



hp education services
education.hp.com

Основы операционной системы UNIX

Учебное пособие

Версия H.01
51434S

Notice

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD PROVIDES THIS MATERIAL "AS IS" AND MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HEWLETT-PACKARD SHALL NOT BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS IN CONNECTION WITH THE FURNISHING, PERFORMANCE OR USE OF THIS MATERIAL WHETHER BASED ON WARRANTY, CONTRACT, OR OTHER LEGAL THEORY).

Some states do not allow the exclusion of implied warranties or the limitations or exclusion of liability for incidental or consequential damages, so the above limitations and exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights, which vary from state to state.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information, which is protected by copyright. All rights reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior consent of Hewlett-Packard Company.

UNIX® UNIX is a registered trademark of the Open Group.

**HP Education Services
100 Mayfield Avenue
Mountain View, CA 94043 U.S.A.**

© Copyright 2001 by the Hewlett-Packard Company

Содержание

Обзор курса

Описание курса.....	1
Цели обучения	1
Предпосылки и требования к студентам.....	6
Использованная документация	6

Модуль 1 – Введение в UNIX

1-1. СЛАЙД: Что такое операционная система?	1-2
1-2. СЛАЙД: История развития операционной системы UNIX	1-4
1-3. ТЕКСТОВАЯ СТРАНИЦА: История ОС UNIX	1-6
1-4. СЛАЙД: Основные компоненты ОС UNIX.....	1-10
1-5. СЛАЙД: Отличительные особенности UNIX	1-14
1-6. СЛАЙД: Система UNIX и стандарты	1-16
1-7. СЛАЙД: Что такое HP-UX?	1-21

Модуль 2 – Вход в систему и простейшие действия

2-1. СЛАЙД: Типичный сеанс работы в системе	2-2
2-2. СЛАЙД: Вход в систему и выход из нее	2-4
2-3. СЛАЙД: Shell — интерпретатор команд	2-6
2-4. СЛАЙД: Формат командной строки	2-7
2-5. СЛАЙД: Вторичное приглашение.....	2-9
2-6. СЛАЙД: Справочник.....	2-10
2-7. СЛАЙД: Содержание статей справочника	2-12
2-8. ТЕКСТОВАЯ СТРАНИЦА: Справочник – пример.....	2-15
2-9. СЛАЙД: Оперативный справочник	2-16
2-10. СЛАЙД: Некоторые начальные команды.....	2-19
2-11. СЛАЙД: Команда id	2-20
2-12. СЛАЙД: Команда who	2-22
2-13. СЛАЙД: Команда date	2-23
2-14. СЛАЙД: Команда passwd	2-24
2-15. СЛАЙД: Команда echo	2-26
2-16. СЛАЙД: Команда banner	2-27
2-17. СЛАЙД: Команда clear	2-28
2-18. СЛАЙД: Команда write	2-29
2-19. СЛАЙД: Команда mesg	2-31
2-20. СЛАЙД: Команда news	2-32
2-21. УПРАЖНЕНИЯ: Вход в систему и простейшие действия	2-34

Модуль 3 – Использование CDE

3-1. СЛАЙД: Элементы передней панели.....	3-2
3-2. СЛАЙД: Всплывающее меню передней панели	3-5
3-3. СЛАЙД: Переключение рабочих областей	3-6
3-4. СЛАЙД: Служба помощи	3-7
3-5. СЛАЙД: Менеджер файлов	3-10
3-6. СЛАЙД: Задачи меню менеджера файлов	3-13
3-7. СЛАЙД: Использование менеджера файлов для поиска файлов	3-16
3-8. СЛАЙД: Удаление объектов.....	3-18
3-9. СЛАЙД: Использование текстового редактора	3-20
3-10. СЛАЙД: Запуск приложений из менеджера приложений	3-25
3-11. СЛАЙД: Использование почтового сервиса	3-27
3-12. СЛАЙД: Отправление сообщений	3-31

Содержание

3-13.	СЛАЙД: Настройка почтового сервиса	3-34
3-14.	СЛАЙД: Использование календаря	3-37
3-15.	СЛАЙД: Планирование встреч	3-39
3-16.	СЛАЙД: Элементы To Do (Сделать)	3-41
3-17.	СЛАЙД: Просмотр календаря по сети	3-43
3-18.	СЛАЙД: Предоставление прав доступа к календарю	3-45
3-19.	Упражнения: Использование CDE	3-47

Модуль 4 – Навигация по файловой системе

4-1.	СЛАЙД: Что такое файловая система?	4-2
4-2.	СЛАЙД: Древоподобная структура	4-3
4-3.	СЛАЙД: Иерархия файловой системы	4-4
4-4.	СЛАЙД: Маршрутные имена	4-7
4-5.	СЛАЙД: Некоторые специальные каталоги	4-10
4-6.	СЛАЙД: Основные команды файловой системы	4-13
4-7.	СЛАЙД: pwd – представить рабочий каталог	4-14
4-8.	СЛАЙД: ls – вывести содержимое каталога	4-15
4-9.	СЛАЙД: cd – изменить каталог	4-18
4-10.	СЛАЙД: Команда find	4-20
4-11.	СЛАЙД: mkdir и rmdir – создать и удалить каталоги	4-21
4-12.	СЛАЙД: Обзор	4-23
4-13.	Слайд: Файловая система: заключение	4-24
4-14.	УПРАЖНЕНИЯ: Файловая система	4-25

Модуль 5 – Работа с файлами

5-1.	СЛАЙД: Что такое файл?	5-2
5-2.	СЛАЙД: Что можно делать с файлами?	5-4
5-3.	СЛАЙД: Характеристики файла	5-5
5-4.	СЛАЙД: cat – вывод содержимого файла	5-7
5-5.	СЛАЙД: more – вывод содержимого файла	5-9
5-6.	СЛАЙД: tail – вывод окончания файла	5-10
5-7.	СЛАЙД: Система спулинга строчных принтеров	5-11
5-8.	СЛАЙД: Команда lp	5-12
5-9.	СЛАЙД: Команда lpstat	5-15
5-10.	СЛАЙД: Команда cancel	5-18
5-11.	СЛАЙД: cp – копирование файлов	5-21
5-12.	СЛАЙД: mv – перемещение или переименование файлов	5-23
5-13.	СЛАЙД: ln – создание ссылки на файл	5-25
5-14.	СЛАЙД: rm – удаление файлов	5-27
5-15.	СЛАЙД: Команды работы с файлами и каталогами: заключение	5-29
5-16.	УПРАЖНЕНИЯ: Работа с файлами и каталогами	5-30

Модуль 6 – Доступ к файлам

6-1.	СЛАЙД: Доступ к файлам	6-2
6-2.	СЛАЙД: Кто имеет доступ к файлу?	6-3
6-3.	СЛАЙД: Типы доступа	6-5
6-4.	СЛАЙД: Права доступа	6-7
6-5.	СЛАЙД: chmod – изменение прав доступа к файлу	6-9
6-6.	СЛАЙД: umask – маскирование прав доступа	6-12
6-7.	СЛАЙД: touch – обновление временных меток файла	6-13
6-8.	СЛАЙД: chown – изменение прав собственности файла	6-15
6-9.	СЛАЙД: Команда chgrp	6-17

6-10.	СЛАЙД: su – переключение идентификатора пользователя	6-19
6-11.	СЛАЙД: Команда newgrp	6-21
6-12.	СЛАЙД: Списки управления доступом ACL	6-23
6-13.	СЛАЙД: Доступ к файлам: заключение	6-26
6-14.	УПРАЖНЕНИЯ: Доступ к файлам	6-27

Модуль 7 – Основы shell

7-1.	СЛАЙД: Что такое shell?	7-2
7-2.	СЛАЙД: Обычно используемые shell'ы	7-4
7-3.	СЛАЙД: Характеристики POSIX shell	7-7
7-4.	СЛАЙД: Установка псевдонимов команд	7-8
7-5.	СЛАЙД: Автоматическое завершение имен файлов	7-10
7-6.	СЛАЙД: Предыстория команд	7-12
7-7.	СЛАЙД: Повторный запуск команды на исполнение	7-14
7-8.	СЛАЙД: Повторный вызов команд	7-15
7-9.	СЛАЙД: Редактирование командной строки	7-17
7-10.	СЛАЙД: Редактирование командной строки (продолжение)	7-19
7-11.	СЛАЙД: Среда пользователя	7-22
7-12.	СЛАЙД: Установка переменных среды	7-24
7-13.	СЛАЙД: Две важные переменные	7-26
7-14.	ТЕКСТОВАЯ СТРАНИЦА: Обычно используемые значения переменных	7-28
7-15.	СЛАЙД: Что происходит при входе в систему?	7-31
7-16.	СЛАЙД: Стартовые файлы shell'a	7-33
7-17.	СЛАЙД: Встроенные команды shell и команды UNIX	7-36
7-18.	СЛАЙД: Поиск команд – команда whereis	7-37
7-19.	ТЕКСТОВАЯ СТРАНИЦА: Пример файла .profile	7-39
7-20.	ТЕКСТОВАЯ СТРАНИЦА: Примеры файлов .kshrc и .logout	7-41
7-21.	УПРАЖНЕНИЯ: Основы shell	7-43

Модуль 8 – Дополнительные возможности shell

8-1.	СЛАЙД: Shell: подстановки	8-2
8-2.	СЛАЙД: Shell: хранение переменных	8-3
8-3.	СЛАЙД: Установка значений переменных shell'a	8-5
8-4.	СЛАЙД: Подстановка переменных	8-6
8-5.	СЛАЙД: Подстановка переменных (продолжение)	8-8
8-6.	СЛАЙД: Подстановка команд	8-9
8-7.	СЛАЙД: Тильда подстановка	8-11
8-8.	СЛАЙД: Вывод значений переменных	8-13
8-9.	СЛАЙД: Экспортирование локальных переменных в среду	8-14
8-10.	СЛАЙД: Передача переменных в приложение	8-15
8-11.	СЛАЙД: Наблюдение за процессами	8-17
8-12.	СЛАЙД: Порожденные процессы и среда	8-19
8-13.	УПРАЖНЕНИЯ: Среда shell	8-20

Модуль 9 – Генерация имен файлов

9-1.	СЛАЙД: Генерация имен файлов: введение	9-2
9-2.	СЛАЙД: Символы генерации имен файлов	9-3
9-3.	СЛАЙД: Генерация имен файлов и файлы с точкой	9-4
9-4.	СЛАЙД: Генерация имен файлов: символ ?	9-5
9-5.	СЛАЙД: Генерация имен файлов: символы []	9-6
9-6.	СЛАЙД: Генерация имен файлов: символ *	9-7
9-7.	СЛАЙД: Генерация имен файлов: заключение	9-8
9-8.	УПРАЖНЕНИЯ: Генерация имен файлов	9-9

Содержание

Модуль 10 – Экранирование

10–1. СЛАЙД: Экранирование: введение	10-2
10–2. СЛАЙД: Символы экранирования	10-3
10–3. СЛАЙД: Экранирование: символ \	10-4
10–4. СЛАЙД: Экранирование: символ '	10-5
10–5. СЛАЙД: Экранирование: символ "	10-6
10–6. СЛАЙД: Экранирование: заключение	10-8
10–7. УПРАЖНЕНИЯ: Экранирование	10-9

Модуль 11 – Перенаправление ввода/вывода

11–1. СЛАЙД: Перенаправление ввода/вывода: введение	11-2
11–2. СЛАЙД: stdin , stdout и stderr	11-3
11–3. СЛАЙД: Перенаправление ввода: <	11-5
11–4. СЛАЙД: Перенаправление вывода: > и >>	11-7
11–5. СЛАЙД: Перенаправление протокола: 2> and 2>>	11-9
11–6. СЛАЙД: Что такое фильтр?	11-10
11–7. СЛАЙД: wc – подсчет количества слов	11-11
11–8. СЛАЙД: sort – алфавитная или числовая сортировка	11-13
11–9. СЛАЙД: grep – поиск по модели	11-15
11–10. СЛАЙД: Перенаправление ввода/вывода: заключение	11-17
11–11. УПРАЖНЕНИЯ: Перенаправление ввода/вывода	11-18

Модуль 12 – Конвейеры

12–1. СЛАЙД: Конвейеры: введение	12-2
12–2. СЛАЙД: Почему используются конвейеры?	12-3
12–3. СЛАЙД: Символ 	12-4
12–4. СЛАЙД: Конвейеры или перенаправление ввода/вывода	12-6
12–5. СЛАЙД: Перенаправление в конвейере	12-7
12–6. СЛАЙД: Фильтры	12-9
12–7. СЛАЙД: Команда cut	12-10
12–8. СЛАЙД: Команда tr	12-12
12–9. СЛАЙД: Команда tee	12-13
12–10. СЛАЙД: Команда pr	12-14
12–11. СЛАЙД: Печать из конвейера	12-16
12–12. СЛАЙД: Конвейеры: заключение	12-17
12–13. УПРАЖНЕНИЯ: Конвейеры	12-18

Модуль 13 – Использование сетевых возможностей

13–1. СЛАЙД: Что такое локальная сеть?	13-2
13–2. СЛАЙД: Сервисы локальной сети (LAN)	13-4
13–3. СЛАЙД: Команда hostname	13-5
13–4. СЛАЙД: Команда telnet	13-6
13–5. СЛАЙД: Команда ftp	13-7
13–6. СЛАЙД: Команда rlogin	13-10
13–7. СЛАЙД: Команда rcp	13-11
13–8. СЛАЙД: Команда remsh	13-13
13–9. СЛАЙД: Команда rwho	13-15
13–10. СЛАЙД: Команда ruptime	13-16
13–11. УПРАЖНЕНИЯ	13-17

Модуль 14 – Введение в редактор vi

14-1. СЛАЙД: Что такое vi?	14-2
14-2. СЛАЙД: Почему vi?	14-4
14-3. СЛАЙД: Начало сеанса работы в vi	14-6
14-4. СЛАЙД: Режимы vi	14-7
14-5. СЛАЙД: Сеанс работы vi	14-9
14-6. СЛАЙД: Завершение сеанса работы vi	14-11
14-7. СЛАЙД: Команды управления курсором	14-12
14-8. СЛАЙД: Команды управления курсором (продолжение)	14-14
14-9. СЛАЙД: Режим ввода: i , a , O , o	14-16
14-10. СЛАЙД: Удаление текста: x , dw , dd , dG	14-18
14-11. УПРАЖНЕНИЯ: Ввод и удаление текста, перемещение курсора	14-20
14-12. СЛАЙД: Перемещение текста: p , P	14-23
14-13. СЛАЙД: Перемещение текста: p , P (продолжение)	14-24
14-14. СЛАЙД: Копирование текста: yw , yy	14-25
14-15. СЛАЙД: Изменение текста: r , R , cw , .	14-27
14-16. СЛАЙД: Поиск текста: / , n , N	14-29
14-17. СЛАЙД: Поиск текста по модели	14-31
14-18. СЛАЙД: Глобальный поиск и замена — команды ex	14-33
14-19. СЛАЙД: Другие команды ex	14-35
14-20. ТЕКСТОВАЯ СТРАНИЦА: Команды vi: обзор	14-37
14-21. УПРАЖНЕНИЯ: Модификация текста	14-38

Модуль 15 – Управление процессами

15-1. СЛАЙД: Команда ps	15-2
15-2. СЛАЙД: Фоновые процессы	15-4
15-3. СЛАЙД: Перевод заданий в фоновый/оперативный режим	15-6
15-4. СЛАЙД: Команда nohup	15-7
15-5. СЛАЙД: Команда nice	15-9
15-6. СЛАЙД: Команда kill	15-11
15-7. УПРАЖНЕНИЯ: Управление процессами	15-14

Модуль 16 – Введение в shell программирование

16-1. СЛАЙД: Shell программирование: введение	16-2
16-2. СЛАЙД: Пример shell процедуры	16-3
16-3. СЛАЙД: Передача данных в shell процедуру	16-5
16-4. СЛАЙД: Аргументы shell процедуры	16-7
16-5. СЛАЙД: Аргументы shell процедур (продолжение)	16-9
16-6. СЛАЙД: Специальные переменные shell'a — # и *	16-10
16-7. СЛАЙД: Специальные переменные shell'a — # и * (продолжение)	16-11
16-8. СЛАЙД: Команда shift	16-12
16-9. СЛАЙД: Команда read	16-14
16-10. СЛАЙД: Команда read (продолжение)	16-16
16-11. СЛАЙД: Дополнительные средства	16-17
16-12. УПРАЖНЕНИЯ: Введение в shell программирование	16-19

Модуль 17 – Shell программирование: ветвление

17-1. СЛАЙД: Коды завершения	17-2
17-2. СЛАЙД: Команда test	17-3
17-3. СЛАЙД: Команда test : сравнение чисел	17-4
17-4. СЛАЙД: Команда test : сравнение строк	17-6
17-5. СЛАЙД: Команда test : анализ файлов	17-8

Содержание

17-6. СЛАЙД: Команда test : другие операции	17-10
17-7. СЛАЙД: Команда exit	17-12
17-8. СЛАЙД: Конструкция if	17-13
17-9. СЛАЙД: Конструкция if-else	17-15
17-10. СЛАЙД: Конструкция case	17-17
17-11. СЛАЙД: Конструкция case : примеры задания шаблонов	17-19
17-12. СЛАЙД: Shell-программирование: ветвление (заключение)	17-20
17-13. УПРАЖНЕНИЯ: Shell программирование: ветвление	17-21

Модуль 18 – Shell программирование: циклы

18-1. СЛАЙД: Циклы: введение	18-2
18-2. СЛАЙД: Команда let : вычисление арифметических выражений	18-3
18-3. СЛАЙД: Конструкция while	18-5
18-4. СЛАЙД: Конструкция while : примеры	18-7
18-5. СЛАЙД: Конструкция until	18-8
18-6. СЛАЙД: Конструкция until : примеры	18-10
18-7. СЛАЙД: Конструкция for	18-11
18-8. СЛАЙД: Конструкция for : примеры	18-13
18-9. СЛАЙД: Команды break , continue и exit	18-15
18-10. СЛАЙД: Команды break и continue : пример	18-17
18-11. СЛАЙД: Shell программирование: циклы (заключение)	18-18
18-12. УПРАЖНЕНИЯ: Shell программирование: циклы	18-19

Модуль 19 – Хранение файлов во внешней памяти

19-1. СЛАЙД: Сохранение файлов на ленте	19-2
19-2. СЛАЙД: Команда tar	19-4
19-3. СЛАЙД: Команда cpio	19-6
19-4. УПРАЖНЕНИЯ: Хранение файлов во внешней памяти	19-9

Приложение А – Краткий справочник команд

А-1. Краткий справочник команд	А-2
--------------------------------------	-----

Ответы и решения

Обзор курса

Описание курса

Этот курс, представляемый Hewlett-Packard, является первым в цикле учебных курсов по UNIX® и по Linux. Он предназначен для ознакомления пользователей любых категорий (системных администраторов, программистов и обычных пользователей) с системой UNIX. Этот курс построен в предположении, что студенты ничего не знают о системе UNIX. (UNIX – это зарегистрированная торговая марка Open Group в США и других странах).

Цели обучения:

Изучив этот модуль, вы сможете:

Модуль 1 — Введение в UNIX

- Описывать общую структуру и возможности операционной системы UNIX
- Описывать HP-UX

Модуль 2 — Вход в систему и простейшие действия

- Входить в систему UNIX
- Выходить из системы UNIX
- Искать описание команд в *Справочном руководстве HP-UX (HP-UX Reference Manual)*
- Искать описание команд, используя оперативный справочник
- Описывать формат командной строки shell'a
- Использовать команды ОС UNIX для идентификации пользователей системы
- Использовать некоторые простые команды ОС UNIX для взаимодействия с пользователями системы
- Использовать командные строки ОС UNIX для запуска утилит и вывода информации

Модуль 3 — Использование CDE

- Описывать элементы передней панели
- Понимать, как работает всплывающее меню передней панели
- Описывать переключение рабочих областей
- Описывать управление дополнительными панелями
- Понимать, как использовать систему помощи
- Описывать систему управления файлами (менеджер файлов)

Обзор курса

- Понимать, как используется меню менеджера файлов
- Использовать менеджер файлов для поиска файлов
- Удалять файлы
- Печатать файлы, используя переднюю панель, менеджер файлов и менеджер печати
- Выводить информацию о системе буферизации принтера
- Понимать управление принтером
- Использовать текстовый редактор
- Использовать менеджер приложений для запуска приложения
- Использовать почтовую службу и ее опции, а также знать, как создаются почтовые ящики (Mailboxes)
- Использовать менеджер календаря для планирования встреч и создания элементов "выполнить"
- Описывать, как можно просматривать другие календари в сети
- Описывать, как разрешать или запрещать доступ к своему календарю

Модуль 4 — Навигация по файловой системе

- Описывать структуру файловой системы UNIX
- Понимать, чем отличаются друг от друга файлы и каталоги
- Перемещаться по файловой системе UNIX
- Создавать и удалять каталоги
- Представлять различие между абсолютным и относительным маршрутным именем
- Использовать (когда это возможно) относительные маршрутные имена для уменьшения объема ввода

Модуль 5 — Работа с файлами

- Использовать общие команды манипулирования файлами системы UNIX
- Объяснить назначение системы спулинга строчных принтеров
- Идентифицировать и использовать команды, используемые для взаимодействия с системой спулинга строчных принтеров
- Наблюдать за состоянием системы спулинга строчных принтеров

Модуль 6 — Доступ к файлам

- Описывать и изменять атрибуты владельца и группы для файла
- Описывать и изменять права доступа к файлу
- Описывать и устанавливать права доступа, используемые по умолчанию при создании новых файлов
- Описывать, как изменить идентификацию пользователя и группы

Модуль 7 — Основы shell

- Описывать назначение shell'a
- Описывать, что происходит, когда пользователь входит в систему
- Описывать переменные среды пользователя и их назначение
- Устанавливать и модифицировать переменные shell'a
- Понимать и изменять специфические переменные среды, такие как *PATH* и *TERM*
- Настраивать среду пользователя в соответствии с потребностями конкретного приложения

Модуль 8 —Дополнительные возможности shell

- Использовать механизм подстановок shell'a, включающий в себя подстановки переменных, команд и тильда подстановки
- Устанавливать и изменять значения переменных shell'a
- Экспортировать локальные переменные в среду
- Обеспечивать доступность переменных из порожденных процессов
- Объяснять, как создаются процессы

Модуль 9 — Генерация имен файлов

- Использовать специальные символы для генерации имен файлов в командной строке
- Экономить время при вводе с клавиатуры за счет использования символов генерации имен файлов
- Выбирать такие имена файлов, чтобы использование символов генерации имен было наиболее эффективным

Обзор курса

Модуль 10 — Экранирование

- Использовать механизм экранирования для отмены особого смысла специальных символов, задаваемых в командной строке

Модуль 11 — Перенаправление ввода/вывода

- Изменять получателя результатов выполнения команд системы UNIX
- Изменять получателя сообщений об ошибках, генерируемых командами системы UNIX
- Изменять источник ввода для команд системы UNIX
- Определять фильтр
- Использовать некоторые простые фильтры, такие как **sort**, **grep** и **wc**

Модуль 12 — Конвейеры

- Объяснить использование конвейеров
- Создавать конвейеры, направляя стандартный вывод одной команды на стандартный ввод другой
- Использовать фильтры **tee**, **cut**, **tr**, **more** и **pr**

Модуль 13 — Использование сетевых возможностей

- Описывать различные сетевые сервисы, используемые в HP-UX
- Объяснить назначение и функции локальной сети (LAN)
- Определить имя локального компьютера и других компьютеров, включенных в локальную сеть
- Использовать ARPA/Berkeley сервисы для входа на удаленную машину, передачи данных между удаленными машинами и выполнения команд на удаленной машине

Модуль 14 — Введение в редактор vi

- Эффективно использовать vi для редактирования текстовых файлов

Модуль 15 — Управление процессами

- Использовать команду **ps**
- Запускать процесс в фоновом режиме
- Наблюдать за исполняющимися процессами с помощью команды **ps**

- Запускать фоновый процесс, который не реагирует на сигнал **hangup**, генерируемый при завершении сеанса работы в системе (log off)
- Переводить процесс из фонового режима в оперативный
- Приостанавливать процесс
- Завершать выполнение процессов, посылая им соответствующие сигналы

Модуль 16 — Введение в shell программирование

- Создавать простые shell процедуры
- Передавать аргументы в shell процедуры, используя переменные среды
- Передавать аргументы в shell процедуры, используя позиционные параметры
- Использовать специальные переменные shell'a * и #
- Использовать команды **shift** и **read**

Модуль 17 — Shell программирование: ветвление

- Описывать использование кодов завершения для ветвления по условию
- Использовать команду **test** для анализа кода завершения
- Использовать инструкции **if** и **case** для организации ветвления в shell процедурах

Модуль 18 — Shell программирование: циклы

- Использовать конструкцию **while** для организации повторного выполнения участка программы, пока некоторое условие остается истинным
- Использовать конструкцию **until** для организации повторного выполнения участка программы до тех пор, пока некоторое условие не станет истинным
- Использовать итерационную конструкцию **for** для последовательной обработки элементов строки, разделенных пробельными символами

Модуль 19 — Хранение файлов во внешней памяти

- Использовать команду **tar** для резервного копирования файлов на ленту
- Использовать команды **find** и **cpio** для резервного копирования файлов на ленту
- С помощью команд **tar** и **cpio** восстанавливать файлы, сохраненные на ленте

Обзор курса

Предпосылки и требования к студентам

Для изучения данного курса нет никаких предпосылок. Тем не менее, предполагается, что студенты работают с компьютерами и хорошо знают клавиатуру.

Использованная документация

- *HP-UX Reference*, P/N B2355-90033.
- *Shells: User's Guide*, P/N B2355-90046.



Этот рисунок пингвина используется везде в материалах данного курса, чтобы идентифицировать информацию, специфическую для системы Linux.

Модуль 1— Введение в UNIX

Цели:

Изучив этот модуль, вы сможете:

- Описывать общую структуру и возможности операционной системы UNIX
- Описывать HP-UX

1–1. СЛАЙД: Что такое операционная система?



Пояснения

Операционная система (ОС) представляет собой специальное программное обеспечение (software), которое управляет работой вычислительной системы (аппаратным обеспечением – hardware). ОС осуществляет взаимодействие между потребителями и ресурсами, распределяя ограниченные ресурсы среди многочисленных потребителей. Ресурсами являются, например, центральный процессор (CPU), диски, оперативная память и принтеры, а потребителями – исполняющиеся программы, требующие доступ к ресурсам. Например, пользователь (или программа) требуют записать файл на диск, и в этот момент вмешивается ОС для управления распределением дискового пространства и передачи информации из оперативной памяти на диск.

Когда пользователь требует выполнения какой-либо программы, ОС должна выделить пространство в оперативной памяти для ее загрузки и исполнения. При выполнении программы ей разрешается доступ к центральному процессору (CPU). В системах разделения времени часто возникает ситуация, когда в один и тот же момент времени несколько программ пытаются получить доступ к процессору.

Операционная система управляет тем, какая программа и в какой момент времени получит доступ к процессору, подобно тому, как регулировщик управляет движением на сложном перекрестке. Здесь перекресток является аналогом процессора: он доступен только для одного варианта движения.

Каждая дорога, идущая через перекресток, аналогична программе. В каждый момент времени движение через перекресток возможно только по одной дороге, и регулировщик указывает, по какой именно дороге возможно движение через перекресток в данный момент времени, давая возможность пересечения перекрестка по всем дорогам.

1–2. СЛАЙД: История развития операционной системы UNIX

История развития операционной системы UNIX

Конец 1960-х г.г.	AT&T разрабатывает MULTICS
1969	AT&T Bell Labs начинает разработку ОС UNIX
Начало 1970-х г.г.	AT&T создает ОС UNIX
Середина 1970-х г.г.	Калифорнийский университет в Беркли (BSD) и другие университеты включаются в исследование и разработку ОС UNIX
Начало 1980-х г.г.	Появляется коммерческий интерес к ОС UNIX DARPA проявляет интерес к BSD Hewlett-Packard представляет HP-UX
Конец 1980-х г.г.	Закладываются основы разработки стандартов открытых систем (OSF)
Начало 1990-х г.г.	Появление POSIX, стандартизация интерактивного интерфейса пользователя

Пояснения

Разработка операционной системы UNIX началась в Bell Laboratories в 1969 г. Кен Томпсон (Ken Thompson), при участии Рада Кэнэдея (Rudd Canaday), Дуга Маклроя (Doug McIlroy), Джо Осанны (Joe Ossana) и Денниса Ритчи (Dennis Ritchie), написали небольшую универсальную операционную систему, поддерживающую режим разделения времени, которая привлекла определенное внимание. Пообещав административному руководству Bell Labs подготовить хорошо документированное программное обеспечение, разработчики получили в свое распоряжение большой компьютер и продолжили разработку.

В середине 1970-х г.г. ОС UNIX была распространена по университетам и получила большую популярность в академических кругах, поскольку была:

- небольшой – ранние версии системы требовали всего 512 Кб дискового пространства, используя 16 Кб для ядра системы, 8 Кб для программ пользователей и 64 Кб для файлов;
- гибкой – был доступен исходный текст, система была написана на языке высокого уровня, что обеспечивало переносимость ОС;
- дешевой – университеты имели возможность получать систему UNIX практически по цене магнитной ленты, на которой была записана дистрибутивная версия ОС. Ранние версии ОС UNIX обеспечивали достаточно мощные возможности, которые были

реализованы только в операционных системах, функционировавших на более дорогих аппаратных платформах.

Наряду с указанными выше преимуществами, имелись в то же время и некоторые недостатки:

- не обеспечивалась должная поддержка ОС – AT&T вложила немалые средства в MULTICS и не была заинтересована в продолжении работ по ОС UNIX;
- в ней имелись ошибки – и, поскольку не обеспечивалась поддержка, не было гарантий по исправлению выявленных ошибок;
- документация была недостаточной или отсутствовала совсем, но всегда имелась возможность обратиться к исходным текстам.

Когда ОС UNIX появилась в Калифорнийском университете в Беркли, пользователи университета в Беркли создали свою собственную версию системы. Поддержанные министерством обороны, они включили в свою версию много новых возможностей. Будучи исследовательским центром, университет в Беркли предложил фирменную политику поддержки системы, тогда как у AT&T она отсутствовала совсем.

AT&T признала потенциальные возможности системы и начала ее коммерческое распространение. Для повышения результативности работы она объединила все разработки по ОС UNIX, выполненные в разных отделах AT&T, а также начала включать в свою версию лучшие решения, реализованные в Беркли.

Последовавший затем успех был связан со следующими свойствами ОС UNIX:

- гибкий интерфейс пользователя, а также операционная среда, которая *включает* большое число утилит;
- модульное построение системы, что позволяет включать новые утилиты;
- возможность поддержки параллельного выполнения многих процессов и работы многих пользователей;
- поддержка Беркли со стороны DARPA;
- возможность использования относительно дешевых и не очень мощных микрокомпьютеров;
- возможность использования ОС UNIX на широком спектре аппаратных платформ;
- стандартизация программного интерфейса, обеспечивающая переносимость приложений.

1–3. ТЕКСТОВАЯ СТРАНИЦА: История ОС UNIX

Приведенные ниже сведения дают более подробное представление об истории развития ОС UNIX.

- 1956 AT&T Consent Decree – антитрестовский процесс AT&T, который исключил ее участие в ряде направлений деятельности, не относящихся к сферам ее непосредственных интересов.
- 1965 Bell Labs, MULTiplexed Information and Computing System (MULTICS) – начаты исследования в области *оконечной* многопользовательской среды; совместный проект с Массачусетским технологическим институтом (Massachusetts Institute of Technology and General Electric).
- 1969 Bell Labs, рождение ОС UNIX – Кен Томпсон (Ken Thompson), исследуя систему разработки файлов, создает программу Space Travel – программу моделирования движения тел в пространстве для миникомпьютера PDP-7. Он также создал файловую систему, ассемблер, редактор и простой shell. Почему для PDP7? Этот компьютер имел *хорошую графику*, он был *дешевле* по сравнению с DEC-10, которая поддерживала интерактивный интерфейс в режиме разделения времени; он хотел обеспечить *удобное интерактивное взаимодействие с компьютером*. Предыдущая разработка была сделана для мейнфрейма GE645, который работал в пакетном режиме и был дорогим. Первоначально программы подвергались кросс компиляции для PDP-7 и загружались с перфоленты. В соответствии с Согласительным декретом (Consent Decree), Bell Labs могла проводить исследования ОС UNIX, но не могла продавать, рекламировать или поддерживать любые программные продукты, базирующиеся на ОС UNIX. Она могла также распространять программное обеспечение в университетах только в целях образования.
- 1970 Появление версии ОС UNIX на ассемблере для PDP-11/20 (16-разрядный миникомпьютер) для исследования возможностей обработки текстов.
- 1971 1-я редакция – патентный отдел Bell Labs (Bell Labs Patent Office) стал первым покупателем ОС UNIX. Большое преимущество для пользователей, которые не имели возможности обслуживания через большой вычислительный центр. Кен Томпсон (Ken Thompson) разрабатывает интерпретирующий язык B, базирующийся на языке BCPL, разработанном Мартином Ричардсом (Martin Richards), а затем – язык NB (новый – new B).
- 1972 2-я редакция – появление каналов, поддержка языков, попытка написать ядро на языке NB (предшественнике языка C). 10 систем.
Деннис Ритчи (Dennis Ritchie) создает язык C.
- 1973 4-я редакция – ядро и shell переписываются на языке C. Для внутренней поддержки системы в Bell Labs создается UNIX Systems Group. 25 систем.
Первое неофициальное распространение системы в университетах.
- 1974 5-я редакция – официально доступная университетам, но *только для учебных целей*. AT&T НЕ обеспечивает поддержку и сопровождение системы, НЕ дает никаких гарантий, НЕ исправляет ошибки; вы ДОЛЖНЫ за все платить авансом.
- 1975 6-я редакция – появляется возможность приобретения системы государственными и коммерческими пользователями. Томпсон (Thompson)

посещает Калифорнийский университет в Беркли (University of California at Berkeley – UCB).

Начало разработок в Беркли.

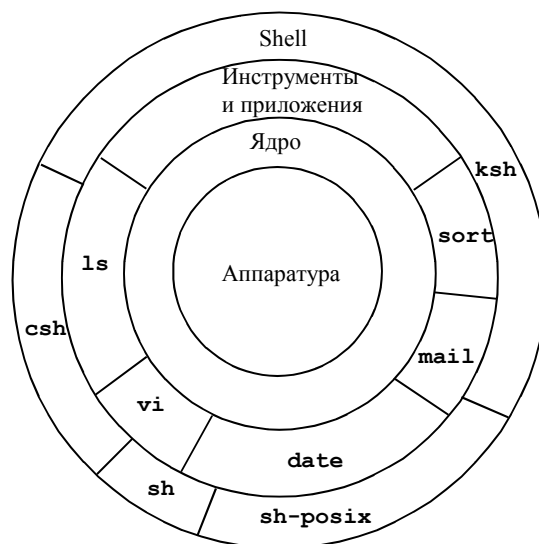
- 1977 500 систем, главным образом, в 125 университетах. BSD разрабатывает первую версию для PDP-11.
Созданы первые порты для подключения внешних устройств, имеющих не DEC архитектуру.
- 1978 7-я редакция – главной целью проекта становится переносимость. Поддерживаются свопинг, компилятор языка C от Кернигана и Ритчи (K&R C), Bourne Shell и большие файлы. Система UNIX переносится на VAX 11/780 (32-х разрядная адресация, 4 Гб виртуального адресного пространства). В результате появилась UNIX/32V.
- 1979 3.0 BSD – улучшение UNIX/32V за счет объединения виртуальной памяти и поддержки страничной организации. Главная цель проектирования – обеспечение возможности исполнения процессов, которые требуют больше памяти, чем физически имеется в наличии.
- 1980 4.0 BSD – включает в себя управление заданиями, виртуальную память, страничную организацию памяти, драйверы для периферийных устройств третьих фирм, использующих не DEC архитектуру. Обеспечивается поддержка независимости от терминала для приложений, использующих экран, таких как, например, vi. К системе проявляет интерес Управление перспективных исследований и разработок Министерства обороны США (Department of Defense Advanced Research Projects Agency – DARPA), которое ищет независимый стандарт на ОС сетевых исследовательских систем для приложений в областях автоматизированного проектирования и производства (CAD/CAM – computer-aided design / computer-aided manufacturing), искусственного интеллекта и распознавания образов. Разработки в области виртуальной памяти в Беркли оказались более продвинутыми, чем у AT&T.
- 1981 Основано объединение пользователей /usr/group – первая организация, которая инициирует определение стандарта для среды ОС UNIX.
- 1982 System III – объединяет свойства нескольких вариантов ОС UNIX, разработанных AT&T, а также включает некоторые возможности BSD, такие как пакет curses, управление заданиями, база данных терминалов termcap и vi. Представлена версия HP-UX.
- 1983 System V Release (версия) 1 – AT&T анонсирует официальную поддержку системы и снижает цену. AT&T предоставляет право производителям микропроцессоров на поддержку системы UNIX. Реализация версии BSD 4.2, основанной на исследованиях DARPA; включение IPC, виртуальной памяти, высокопроизводительной файловой системы, сетевой архитектуры (TCP/IP). Выпуск 16-ти и 32-х разрядных микрокомпьютеров. Включение средств BSD-IPC, сетевых возможностей, высокопроизводительной файловой системы. 100,000 систем UNIX.
- 1984 Consent decree аннулируется, Bell лишается монопольных прав – это позволяет AT&T конкурировать в компьютерном бизнесе.
System V Release 2 – поддержка страничной организации, разделяемая память.

- Стандарт, выработанный объединением пользователей /usr/group, представляется на рассмотрение в рамках POSIX.
- Ричард Сталлмен (Richard Stallman) создает проект GNU (этот проект становится основой для утилит, которые вошли в состав ОС Linux).
- 1985 System V Interface Definition (SVID) – определяет интерфейс системных вызовов.
System V Verification Suite (SVVS) – определяет набор тестов, которые должны быть выполнены для того, чтобы подтвердить соответствие SVID.
- 1986 4.3 BSD – устранение обнаруженных ранее ошибок, управление заданиями, введение сигналов диагностики.
- 1987 System V Release 3 (SVR3) – потоки (STREAMS), IPC, управление заданиями.
X/Open Portability Guide (XPG) – определяет интерфейс ядра и большое количество утилит, которые обеспечивают переносимость приложений между различными реализациями систем UNIX. Установлено еще 300,000 систем UNIX. Общее количество – 750,000 систем UNIX.
- 1988 SVID, выпуск 2 (SVID2) – механизм блокировки файлов.
Для разработки и обеспечения компьютерной среды создана независимая организация по программному обеспечению открытой архитектуры (OSF – Open Software Foundation), которая базируется на промышленных стандартах и новейших доступных технологиях.
- 1989 System V Release 4 (SVR4) – согласованность с POSIX.1. XPG/3 – поддержка POSIX.1 и общей среды приложений (Common Application Environment); выбираются стандарты, которые будут объединены для различных видов компьютерных сред, а не только для интерфейса с ОС, обеспечивающего переносимость программного обеспечения.
- 1990 SVID, выпуск 3 (SVID3) – включает POSIX.1, FIPS 151-1 и стандарт языка C.
- 1991 HP-UX 8.0 – развитие SVR3, удовлетворяющее требованиям SVID2 и объединяющее расширения систем BSD4.2 и BSD4.3, ставших де-факто промышленным стандартом; объединение POSIX-, FIPS-, XPG2-, и XPG3-совместимых интерфейсов.
- Линус Торвалдс (Linus Torvalds) создает ядро ОС **Linux**. Возможности этого ядра позволяют компоновать открыто распространяемое в исходных кодах программное обеспечение GNU как завершенную операционную систему. ОС "GNU/Linux" позднее становится известной под сокращенным именем "Linux".
- 1992 HP-UX 9.0 – развитие SVR3, удовлетворяющее требованиям SVID2, а также требованиям со стороны X/Open Portability Guide, выпуск 3, POSIX 1003.1 и POSIX 1003.2, X11R5, FIPS-2 и FIPS-3, POSIX.1, OSF/Motif 1.2, и других.
- 1995 HP-UX 10.0 – по организации ядра система соответствует SVID3, удовлетворяет требованиям со стороны X/Open Portability Guide, выпуск 4, POSIX.4 Realtime Phase 1 и других, гарантирует переносимость от версии от 9.0 к версии 10.0. Главное отличие заключается в том, что структура файловой системы была изменена в соответствии с парадигмой AT&T SVR4 и OSF/1.

- 1997 HP-UX 11.0 – удовлетворяет требованиям SVID3, версия 4 – POSIX.2. IA64 - совместимость (64-х разрядная адресация) для 64-х битных приложений. Реализованы нити ядра.
- 1999 Red Hat Inc. создает Red Hat Linux, версия 6.0. Эта версия ОС Linux быстро становится коммерческим стандартом и устанавливается на разных промышленных вычислительных системах.

1–4. СЛАЙД: Основные компоненты ОС UNIX

Основные компоненты ОС UNIX



Пояснения

ОС UNIX представляет собой операционную систему разделения времени, которая управляет работой и ресурсами компьютера и предоставляет интерактивный, гибкий интерфейс. Она была спроектирована с целью обеспечения мультипроцессной параллельной обработки и обслуживания многих пользователей, обеспечивая разделение данных между членами коллектива, ведущими совместную работу. Операционная среда на всех уровнях была спроектирована в соответствии с модульной архитектурой. При установке ОС UNIX следует устанавливать только те ее части, которые необходимы для работы, и опускать все лишнее. Например, система UNIX имеет большой набор утилит, необходимых для разработки программ. Если же работа не связана с разработкой программных средств, необходимо установить только компилятор. Интерфейс пользователя также эффективно поддерживает модульную структуру. Команды, которые совершенно независимы друг от друга, легко комбинируются с помощью конвейеров для выполнения сложной обработки данных.

Операционная система

Собственно операционной системой является **ядро**. Оно управляет всеми доступными ресурсами и осуществляет доступ к аппаратуре. Ядро состоит из модулей, каждый из которых обеспечивает интерфейс с одним из компонентов аппаратных средств. Эти модули обеспечивают программам доступ к процессору, оперативной памяти, дискам, терминалам, сети и т.д. Как только в систему включаются новые типы устройств, в ядро включаются новые программные модули.

Операционная среда

Инструментальные средства и приложения

Модульная структура ОС UNIX наиболее ярко проявляется именно на этом уровне. Принцип построения команд системы UNIX заключается в том, что каждая команда хорошо делает что-то одно, а совокупность команд создает некоторый "инструментальный ящик" (tool box). Когда нужно выполнить некоторое задание, следует просто извлечь соответствующие инструменты. Сложные задачи могут быть решены путем объединения соответствующих инструментов.

С самого начала "инструментальный ящик" системы UNIX включал в себя много больше, чем просто основные команды, необходимые для взаимодействия с системой. Система UNIX также имеет утилиты для:

- электронной почты (**mail, mailx**)
- редактирования файлов (**ed, ex, vi**)
- обработки текста (**sort, grep, wc, awk, sed**)
- форматирования текста (**nroff**)
- разработки программ (**cc, make, lint, lex**)
- управления программами (SCCS, RCS)
- внутрисистемных коммуникаций (**uucp**)
- сбора учетной информации о процессах и пользователях (**ps, du, acctcom**)

Так как пользовательская среда системы UNIX была разработана и реализована интерактивной, программируемой и модульной, в инструментальный ящик пользователя могут быть легко добавлены новые утилиты, а ненужные средства могут быть опущены без какого-либо ущерба для системы.

Например, ОС UNIX используют прикладной программист и технический редактор (technical writer). Они будут использовать много общих команд, несмотря на то, что их приложения сильно отличаются друг от друга. Кроме того, они будут использовать утилиты, ориентированные только на их разработки. Система прикладного программиста включает утилиты для разработки программ и управления ими, тогда как система технического редактора будет содержать утилиты для форматирования и обработки текста и управления документами. Необходимо отметить, что утилиты, которые разработчик приложений использует для управления версиями программы, могут также использоваться техническим редактором для управления версиями документа. Поэтому их системы будут очень похожими, хотя каждый пользователь выберет модули, которые соответствуют потребностям его приложений, и отбросит все ненужное.

Популярность системы UNIX в большой степени определяется следующими факторами:

- завершенностью и гибкостью системы UNIX, что делает ее пригодной к использованию во многих предметных областях;
- многочисленностью утилит, включенных в операционную среду и повышающих производительность труда пользователей;
- доступностью и переносимостью системы на многочисленные аппаратные платформы.

Shell

Shell является *интерактивным* командным интерпретатором. Команды вводятся в ответ на приглашение shell'a и сразу же выполняются. Взаимодействие пользователя с компьютером осуществляется через shell. Shell воспринимает команду, которую пользователь вводит с клавиатуры, и преобразует ее к виду, понятному ядру системы. Затем система выполняет команду.

Важно отметить, что shell *отделен* от ядра системы. Если пользователя не устраивает интерфейс, предоставляемый имеющимся shell'ом, он может легко заменить его другим. В настоящее время доступно много shell'ов. Некоторые из них управляются командами, другие – через меню. В общем случае shell'ы, поставляемые с системой UNIX, предоставляют и интерпретатор команд, и программный интерфейс.

Наиболее часто в среде UNIX доступны следующие четыре shell'a:

- Bourne Shell (**/usr/old/bin/sh**) – первый shell, разработанный Стефаном Борном (Stephen Bourne) из Bell Laboratories, который поставлялся с системами, базировавшимися на версии AT&T. Он выполняет интерпретацию команд и поддерживает программный интерфейс для разработки shell программ, которые обычно называют shell процедурами (scripts). Программный и интерактивный интерфейсы предоставляют такие возможности, как определение и подстановки переменных, анализ значений переменных и характеристик файлов, ветвление и циклы.
- C Shell (**/usr/bin/csh**) – shell, разработанный Биллом Джоем (Bill Joy) в Калифорнийском университете в Беркли и поставляемый с системами, базирующимися на версии BSD. Сначала этот shell назывался California Shell, а затем его название было сокращено до C Shell. Он рассматривается как улучшение Bourne shell'a, так как предоставляет такие интерактивные возможности, как стек команд, который позволяет повторно вызывать и редактировать введенные ранее команды, и использование псевдонимов, которые позволяют присваивать собственные альтернативные имена существующим командам.
- Korn Shell (**/usr/bin/ksh**) – более поздняя разработка Bell Laboratories, автор которой – Давид Корн (David Korn). Этот shell рассматривается как усовершенствованный Bourne shell, так как поддерживает простой программный интерфейс Bourne shell'a и обладает удобными интерактивными особенностями, присущими C shell'у. Код был оптимизирован, что делает shell более быстрым и эффективным. GNU версия Korn shell'a доступна и в ОС Linux.
- POSIX Shell (**/usr/bin/sh**) – POSIX-совместимый командный язык программирования и интерпретатор команд, размещенный в файле **/usr/bin/sh**. Во многих отношениях этот shell похож на Korn shell: он поддерживает буфер команд, управление заданиями и обладает многими другими полезными особенностями.

Система Linux



Bash shell (**/usr/bin/bash**) – вариант Bourne shell'a, который был модифицирован путем введения POSIX-совместимых опций и управления. Название bash представляет собой аббревиатуру для **B**ourne **A**gain **S**hell. Этот shell предоставляет функции и механизмы, аналогичные имеющимся в POSIX shell'е.

Таблица 1. Сравнительные характеристики shell'ов

Характеристика	Описание	Bourne	Bash	Korn	C	POSIX
Поддержка буфера команд	Возможность запоминания команд в буфере; в дальнейшем эти команды могут модифицироваться и использоваться вновь	Нет	Да	Да	Да	Да
Редактирование командной строки	Возможность модификации текущей или ранее введенной команды с помощью текстового редактора	Нет	Да	Да	Нет	Да
Завершение имени файла	Возможность автоматически завершать ввод имени файла в командной строке	Нет	Да	Да	Да	Да
Команда alias	Возможность, позволяющая пользователю переименовывать команды, автоматически включая опции команды, или использовать аббревиатуры для длинных командных строк	Нет	Да	Да	Да	Да
Ограниченный shell	Средство защиты, обеспечивающее настройку среды пользователя с ограниченными возможностями	Да	Нет	Да	Нет	Да
Управление заданиями	Средства для трассировки и организации доступа к процессам, которые выполняются в фоновом режиме	Нет	Да	Да	Да	Да

1–5. SLIDE: Отличительные особенности UNIX

Отличительные особенности UNIX

- Иерархическая файловая система
- Мультизадачный режим работы
- Обслуживание многих пользователей

Пояснения

Иерархическая файловая система

Информация хранится на диске в виде **файлов**. Каждый файл имеет имя, и пользователь получает доступ к файлу, обращаясь к нему по имени. В файлах обычно содержатся числовые данные, тексты, программы и т.д. Обычно в системе UNIX хранятся сотни файлов, поэтому в системе существуют **каталоги**, с помощью которых пользователи могут логически группировать файлы. В ОС UNIX каталог может быть использован для хранения файлов и других каталогов.

Структура файловой системы очень гибкая, так что если потребности пользователей в организации его файлов изменились, можно с помощью простых команд системы UNIX переместить файлы и каталоги, переименовать их, сгруппировать файлы в новых или разместить в разных каталогах. Файловая система позволяет пользователям разделять свою информацию и размещать ее в разных каталогах, наилучшим образом соответствующих решаемым задачам и приложениям.

Мультизадачный режим работы

В системе UNIX в одно и то же время могут выполняться несколько задач. Пользователь со своего терминала может запустить на выполнение несколько программ, и будет казаться, что все программы выполняются одновременно. Это значит, что пользователь может

редактировать текстовый файл, тогда как другой файл в это время форматируется, а третий файл распечатывается.

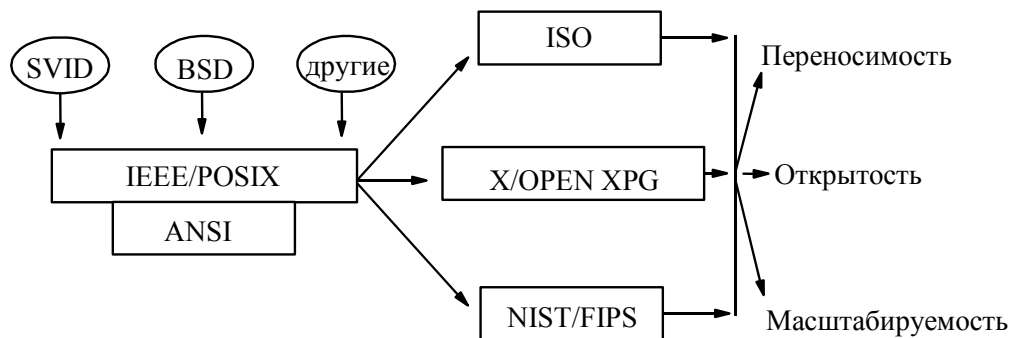
В действительности, на центральном процессоре в каждый момент времени может выполняться только одна программа, но ОС UNIX имеет возможность разделять время процессора между многими процессами, которые должны выполняться в одно и то же время. Поэтому пользователю кажется, что все процессы выполняются одновременно.

Обслуживание многих пользователей

Способность обслуживать многих пользователей позволяет в одно и то же время входить в систему и использовать ее более чем одному пользователю. К одному компьютеру можно подключить несколько терминалов. Эта способность является естественным расширением мультизадачного режима работы. Если система может одновременно выполнять несколько программ, то некоторые из этих программ могут относиться к сеансам работы с системой других пользователей. Кроме того, один и тот же пользователь может многократно входить в систему под одним именем с разных терминалов. Большим преимуществом такой архитектуры является то, что члены одной рабочей группы могут получать доступ в одно и то же время к одним и тем же данным, либо по логике выполняемой ими разработки, либо с пользовательской точки зрения.

1–6. СЛАЙД: Система UNIX и стандарты

Система UNIX и стандарты



Пояснения

С самого начала система UNIX разрабатывалась с ориентацией на переносимость.

Так как большая часть операционной системы и утилиты написаны на языке C (а не на ассемблере), система UNIX не ограничивается каким-либо одним процессором или аппаратной платформой. С другой стороны, так как система UNIX написана на языке высокого уровня, ее легко модифицировать, что продемонстрировали более чем 100 компаний, предложивших основанные на системе UNIX реализации (лицензированные Open Group) и клоны семейства UNIX (новые реализации с UNIX-подобным интерфейсом, которые не требуют лицензирования со стороны Open Group). Несмотря на то, что большинство систем берут свое начало от AT&T UNIX или BSD UNIX или от обеих систем, каждая конкретная реализация может содержать уникальные расширения операционной системы, такие как, например, возможность функционирования в реальном времени, что может привести к несовместимости различных реализаций системы UNIX. (Действительно, System V уже обладает многими популярными характеристиками BSD). Для поддержки совместимости различных реализаций для операционной среды системы UNIX начали формироваться стандарты.

Целью этих стандартов является обеспечение следующих возможностей:

1. Переносимость – возможность легкого переноса приложения с одной реализации системы UNIX в другую.

2. Открытость – возможность разделять информацию приложениями, выполняющимися на разных реализациях системы UNIX.
3. Масштабируемость – обеспечение широкого диапазона аппаратных платформ, от малых систем до больших, выбираемых пользователями в соответствии с их потребностями, а также возможность наращивать систему в соответствии с изменением требований приложения.

Несмотря на большое количество реализаций системы UNIX, различия на уровне пользователя незначительны, так как большинство реализаций создано исходя из общих принципов. Поэтому вначале стандарты были направлены на определение интерфейса программ с ядром операционной системы, и лишь с недавних пор определяются стандарты на интерактивный интерфейс пользователя.

Цели групп по стандартизации

Определение требований к интерфейсу, а не к реализации

Стандарты не преследуют цель полного создания нового интерфейса, а направлены на выработку хорошо определенного и переносимого интерфейса, базирующегося на существующих в настоящее время реализациях систем UNIX. Важно понимать, что стандарты предназначены для определения интерфейса с операционной средой системы UNIX, но не для определения того, как этот интерфейс будет реализован. Поэтому стандарты системы UNIX не требуют, чтобы все компьютеры, работающие под UNIX, были точными копиями; скорее, они требуют того, чтобы поддерживался некоторый общий набор функций, на основе которого формируется конкретная реализация.

Хорошим примером такого подхода может быть автомобиль. Базовый интерфейс, определяемый автомобилем "стандарт", включает:

Двигаться – нажать на акселератор

Остановиться – нажать на тормоз

Изменить направление – повернуть руль

Запустить двигатель – повернуть ключ

Автомобили, которые поддерживают такой стандартный интерфейс, могут быть разработаны с большим разнообразием реализаций. Например, автомобиль может иметь электрический или газовый двигатель, но нажатие на акселератор заставит автомобиль двигаться.

Интересной стороной такого подхода является то, что с точки зрения поддержки предписываемых стандартом интерфейсов он может быть реализован не только для операционных систем UNIX.

Модульность

Вычислительная среда постоянно изменяется и расширяется. Поэтому стандарты должны быть расширяемыми. Они должны соответствовать достижениям технологии и требованиям пользователей.

Стандарты формируются по модульному принципу, так что их можно расширять или при необходимости заменять, если появляется лучший интерфейс.

AT&T System V Interface Definition (SVID)

Компания AT&T первой разработала стандарт для операционной среды системы UNIX. Этот стандарт, базирующийся на версии AT&T System V, ориентирован на уровень функционального интерфейса с операционной системой (системные вызовы), взаимодействие процессов, shell системы UNIX и некоторые базовые утилиты.

Кроме того, компания AT&T разработала System V Verification Suite (SVVS) – комплекс для проверки соответствия требованиям SVID.

Хотя SVID является первой попыткой разработки стандарта, стандарт не был нейтральным, так как AT&T – компания бескомпромиссная в своих требованиях. Например, в версии System V Release 3 (System V.3), AT&T установила такие жесткие ограничения для реализаций, которые хотели подтвердить System V.3, что некоторые расширения Беркли не были совместимыми с System V.3.

IEEE/POSIX

Институт инженеров по электротехнике и радиоэлектронике (The Institute of Electrical and Electronics Engineers – IEEE) является крестным отцом интерфейса переносимых операционных систем для компьютерных сред (Portable Operating System Interface – POSIX). POSIX ведет свой отсчет с 1984 года от группы /usr/group по стандартам, чьей целью было определение стандартов, независимых от SVID (/usr/group является предшественницей UniForum). POSIX 1003 был учрежден для разработки стандартов для *всей операционной среды*, а не только для интерфейса с ядром системы. В отличие от AT&T, POSIX определяет программный интерфейс, не определяя требований по его реализации. Поэтому могут быть разработаны POSIX-совместимые системы, которые не базируются на кодах AT&T.

POSIX также был передан ISO для включения в международный стандарт. Это связано с проектом, предполагавшим стандарт TC22 WG15.

Для продвижения разработки стандарта, POSIX был разделен на несколько частей, и для разработки каждой части была создана рабочая группа.

- 1003.1 Системный интерфейс (создана в 1981 г.).
Обеспечивает исходный текст, программный интерфейс, связанный с языком высокого уровня, который обеспечивает переносимость приложений. POSIX.1 тесно связан с SVID, выпуск 2 (SVID2), но также включает в себя возможности из BSD 4.3 и дополнительные возможности, которые не поддерживались определениями других интерфейсов.
- 1003.2 Shell'ы и утилиты (создана в 1984 г.).
Определяет командный язык shell и интерактивные утилиты.
- 1003.3 Методы тестирования (создана в 1986 г.).
Определяет общие требования, касающиеся того, какие тестовые комплексы должны быть написаны, и как их администрировать. Предоставляет перечень операторов, точно определяющих, что в стандарте POSIX необходимо тестировать. Эта рабочая группа не уполномочена оценивать пригодность тестов; за методику тестирования отвечают поставщики системы.
- 1003.4 Реальное время
 - 4a Расширения, связанные с нитями (threads)
 - 4b Спецификации, независимые от языка

1003.5	Связь языка Ada с POSIX
1003.6	Безопасность
1003.7	Администрирование системы
1003.8	Сетевые возможности
1003.9	Связь языка FORTRAN с POSIX
1003.10	Параметры выполнения приложений на суперкомпьютерах (Application Execution Profile – AEP)
1003.11	Обработка транзакций в AEP
1003.12	Независимые от протокола интерфейсы
1003.13	Реальное время в AEP
1003.14	Параметры выполнения приложений на мультипроцессорных системах

X/Open и Open Group

X/Open представляет собой международный консорциум, объединивший поставщиков информационных систем, пользователей, системных интеграторов и разработчиков программного обеспечения, задавших целью определить общую среду приложений (Common Application Environment). Их целью было не определить новые стандарты, а выбрать из существующих стандартов те, которые гарантируют переносимость и работу приложений в разных системах, что позволяет пользователям переходить на разные системы без дополнительных усилий. X/Open также берет свое начало из SVID, но является, по сути, существенным расширением POSIX. X/Open's Portability Guide (XPG, в настоящее время версия 4) включает в себя набор соответствующих стандартов, относящихся к единой среде приложений.

В настоящее время X/Open объединилась с Open Software Foundation (OSF), в результате чего появилась Open Group.

Ниже представлены некоторые элементы:

Компонент	Определяющий стандарт
Системные вызовы и библиотеки	POSIX 1003.1
Команды и утилиты	POSIX 1003.2
Язык C	ANSI
Язык COBOL	ANSI/ISO
Язык FORTRAN	ANSI
Язык Pascal	ISO

SQL

ANSI

Менеджер окон

X Window System

Американский национальный институт стандартов (American National Standards Institute - ANSI)

Представляет собой координирующую организацию по стандартизации в США. IEEE является комитетом по стандартам, аккредитованным при ANSI.

Международная организация стандартов (International Standards Organization - ISO)

Координирует принятие международных стандартов для распределенных информационных систем в среде открытых систем (среда неоднородных сетевых систем). Наиболее заметные разработки были сделаны в области сетевых систем и определения эталонной семиуровневой сетевой модели открытых взаимосвязанных систем (Open Systems Interconnection - OSI).

Членами ISO обычно являются национальные организации по стандартизации. От США членом ISO является ANSI.

Национальный институт стандартов и информации федеральной технологии (National Institute of Standards and Technology Federal Information)

Действующий стандарт (NIST/FIPS)

NIST подчиняется Министерству торговли; ранее был известен под названием «Национальное бюро стандартов» (National Bureau of Standards – NBS). Эта организация разрабатывает стандарты для правительственных учреждений. Изначально задача заключалась в оценке предложенных POSIX.1 стандартов; появившийся затем федеральный стандарт на обработку информации (Federal Information Processing Standard – FIPS) объединил POSIX и ряд дополнительных возможностей POSIX.1, рассматриваемых как не обязательные или не специфицированные.

В задачу NIST/FIPS входит также оценка других компонентов POSIX по мере того, как они становятся доступными.

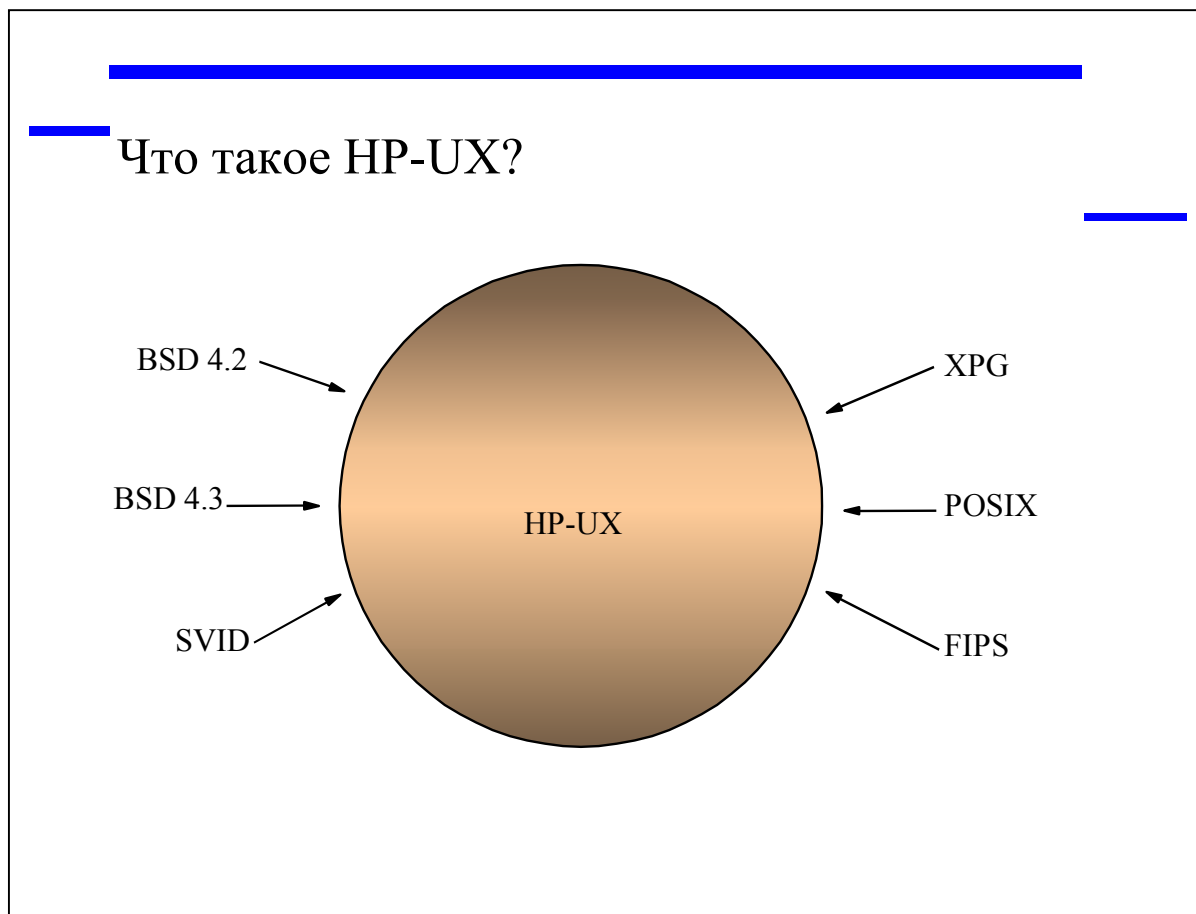
Стандарт Linux



За последнее время были созданы несколько стандартов для контроля над разработкой операционной системы Linux и соответствующих утилит. Подробную информацию можно найти на следующих сайтах (URL):

www.linux.org
www.gnu.org
www.redhat.com

1–7. СЛАЙД: Что такое HP-UX?



Пояснения

HP-UX 11.0 является реализацией операционной системы UNIX AT&T System V, удовлетворяющей следующим стандартам:

- POSIX.1:1996 (IEEE стандарт 1003.1:1996)
- POSIX.2:1992 (IEEE стандарт 1003.2:1992)
- System V Interface Definition (SVID3)
- OSF/Motif 1.2
- X Window System, версия 11, редакция 4
- X Window System, версия 11, редакция 5
- X Window System, версия 11, редакция 6.2
- OSF AES OS Component, исправленная версия A (S300, S400 и рабочая станция HHP 9000)
- X/Open Portability Guide, выпуск IV
- FIPS 151-1, FIPS 151-2 и FIPS 189
- ANSI C (ANS X3.159-1989)
- X/OPEN UNIX95 – зарезервированная марка
- LP64 Industry 64-разрядный стандарт
- SPEC 1170

Доступны также следующие дополнительные возможности, реализованные Hewlett-Packard:

- X Windows и Motif – графический интерфейс пользователя
- Common Desktop Environment – интерфейс, основанный на Motif (CDENext)
- Визуальный редактор (Visual Editor) – основанный на Motif текстовый редактор
- Графические языки
- Поддержка естественных языков
- Система администрирования, основанная на меню (SAM)
- Документация и инсталляционная копия системы на CD ROM
- BIND 4.9
- POSIX.3 (IEEE стандарт 1003.3c) – использование нитей в ядре системы
- Транспортный потоковый стек для TCP/IP
- BSD 4.3

Система Linux



Операционная система Red Hat Linux 6.x во многом аналогична HP-UX и другим коммерческим версиям ОС UNIX. Red Hat, вместе с версиями пакета Linux для рабочих станций и серверов, включает в себя следующее:

- Gnome RPM – Red Hat Package Manager – инструментальные программные средства для управления
- Графическая среда пользователя GNOME
- Графическая среда пользователя KDE
- SAMBA – файловый сервер и сервер печати для систем, основанных на Windows 95/98/NT
- Программное обеспечение web-сервера APACHE
- Средства администрирования, основанные на меню (linuxconf)

Модуль 2 — Вход в систему и простейшие действия

Цели:

Изучив этот модуль, вы сможете:

- Входить в систему UNIX
- Выходить из системы UNIX
- Искать описание команд в *Справочном руководстве HP-UX (HP-UX Reference Manual)*
- Искать описание команд, используя оперативный справочник
- Описывать формат командной строки shell'a
- Использовать команды ОС UNIX для идентификации пользователей системы
- Использовать некоторые простые команды ОС UNIX для взаимодействия с пользователями системы
- Использовать командные строки ОС UNIX для запуска утилит и вывода информации

2–1. СЛАЙД: Типичный сеанс работы в системе

Типичный сеанс работы в системе

- Вход в систему: аутентификация пользователя и получение доступа к системе
- Выполнение команд, производящих необходимые действия
- Выход из системы: завершение сеанса работы

Пояснения

Для установления связи с компьютером необходимо иметь:

- терминал, поддерживающий полный набор символов американского стандартного кода для обмена информацией (American Standard Code for Information Interchange – ASCII);
- линию передачи данных в компьютер;
- входное имя (login ID) пользователя;
- пароль.

Сеанс работы начинается входом в систему с терминала, подключенного к системе и опознанного ею, и завершается выходом из системы. Компьютер будет выполнять требуемые действия в ответ на команды, вводимые в течение сеанса работы.

Система UNIX идентифицирует каждого своего пользователя с помощью **имени пользователя** (которое обычно называют **login ID – регистрационным идентификатором**, или **входным именем пользователя**). В качестве входного имени, назначаемого пользователю администратором системы, обычно используется имя пользователя или его инициалы. В учетную запись пользователя может быть введен и пароль. Администратор системы может определить для пользователя начальный пароль, который он в дальнейшем может изменить; пользователь может установить для себя свой пароль. Каждый пользователь имеет собственный пароль; только пользователь решает, каким будет его пароль, и *никто другой* не сможет узнать или найти чужой пароль. Если пользователь

забудет свой пароль, он должен обратиться за помощью к администратору системы. Только администратор системы имеет право удалить пароль из учетной записи пользователя.

В ответ на приглашение **login:**, которое выводится на терминал, пользователь вводит свое входное имя и, если есть, пароль.

Войдя в систему, пользователь может вводить команды. В ответ на требование пользователя shell интерпретирует их, а операционная система выполняет. Любой ответ, генерируемый во время выполнения команды, будет выведен на экран.

По окончании работы пользователь завершает сеанс работы, выходя из системы. Это приводит к освобождению терминала, так что другой пользователь может войти с него в систему. Пользователю настоятельно рекомендуется выходить из системы, когда он оставляет терминал, чтобы исключить возможность другим пользователям воспользоваться его сеансом работы и учетной записью.

2–2. СЛАЙД: Вход в систему и выход из нее

Вход в систему и выход из нее

```
login: user1
Password:           
Welcome to HP-UX
Erase is Backspace
Kill is Ctrl-U
$ date
Fri Jul 1 11:03:42 EDT 1994

$ другие команды

$ exit
login:
```

Вход в систему

Входные сообщения

Выполнение
требуемых действий

Выход из системы

Пояснения

Для входа в систему нужно выполнить следующие действия:

- Включить терминал. На некоторых терминалах для вывода установлен режим таймаута, поэтому достаточно нажать какую-либо клавишу (например, **[Shift]**), чтобы активизировать дисплей.
- Если на экране нет приглашения **login:** или же выведена посторонняя информация ("мусор"), следует нажать клавишу **[Return]**. Если это не поможет, нужно нажать клавишу **[Break]**. Мусор на экране обычно означает, что компьютер пытался установить соединение с терминалом не на нужной скорости. Нажатие клавиши **[Break]** сообщает компьютеру, чтобы он попытался использовать другую скорость обмена. Возможно, придется нажимать клавишу **[Break]** несколько раз, чтобы проверить разные скорости обмена; при этом следует дожидаться ответной реакции после каждого нажатия клавиши.
- Когда на экране появится приглашение **login:**, нужно ввести свое входное имя.
- Если появилось приглашение **password:**, нужно ввести свой пароль. Для обеспечения секретности при вводе пароля он на экране не отображается. При вводе входного имени и пароля клавиша **[#]** действует как забой (**[Backspace]**), а клавиша **[@]** удаляет всю строку. Следует обратить внимание на то, что при входе в систему во время ввода входного имени и пароля клавиша **[Backspace]** не выполняет функции удаления.

Символ `$` является стандартным приглашением для командных интерпретаторов Bourne shell (`/usr/old/bin/sh`), Korn shell (`/usr/bin/ksh`) и POSIX shell (`/usr/bin/sh`). Символ `%` обычно обозначает C shell (`/usr/bin/csh`). В данном курсе используется POSIX shell, поэтому вы увидите приглашение `$`. Приглашение `#` обычно резервируется для администратора системы. Это служит полезным визуальным напоминанием того, что пользователь вошел в систему как ее администратор, так как администратор системы может модифицировать (или удалять) в системе все.

Задание пароля

Учетная запись пользователя может быть установлена таким образом, что при первом входе в систему пользователь должен будет определить свой пароль. Пароль должен удовлетворять следующим требованиям:

- должен состоять не менее чем из шести символов;
- по крайней мере, два из первых шести символов должны быть буквами;
- по крайней мере, один из первых шести символов должен быть не буквой.

После первого ввода пароля система попросит пользователя ввести пароль повторно для проверки правильности его ввода. Затем система повторно выведет приглашение для входа в систему, и пользователь сможет завершить процедуру входа в систему со своим новым паролем.

ПРИМЕЧАНИЕ: Когда вход в систему осуществляется через CDE или HP-VUE, пользователь может с помощью мыши выбрать поле login и ввести с клавиатуры свое входное имя. После этого автоматически будет выбрано поле пароля (т.е. курсор переместится в это поле). Если пользователь имеет пароль, он должен ввести его с клавиатуры, при этом пароль не будет отображаться в поле пароля. Для исправлений во время ввода входного имени и пароля можно использовать клавишу `Backspace`. Исправления осуществляются процессом входа в CDE или HP-VUE.

2–3. СЛАЙД: Shell — интерпретатор команд



Пояснения

В процессе входа пользователя в систему для него запускается shell (в нашем случае POSIX shell). Shell отвечает за вывод приглашения и интерпретацию вводимых команд. В оставшейся части данного модуля будут рассмотрены некоторые команды, с помощью которых можно будет обращаться к справочному руководству, выводить информацию о пользователях, работающих в системе, и устанавливать с ними связь.

Как видно из слайда, shell выполняет много других функций, которые дополняют интерпретатор команд.

2–4. СЛАЙД: Формат командной строки

Формат командной строки

Синтаксис:

\$ команда [-опции] [аргументы] Return

Примеры:

\$ date Return

Без аргументов

Fri Jul 1 11:10:43 EDT 1994

\$ banner hi Return

Один аргумент

#

#

#

#

#

\$ bannerHi Return

Неправильный синтаксис

sh: bannerHi : not found

\$ ls -F Return

Одна опция

dira/ dirb/ f1 f2 prog1* prog2*

Пояснения

После того как на экране появится приглашения shell'a (\$), можно вводить команды. Имя команды должно быть всегда первым словом командной строки. Многие команды допускают задание опций, расширяющих функциональные возможности команды, и аргументов, часто задающих текстовые строки или имена файлов или каталогов, с которыми команда должна работать. Перед опциями обычно ставится знак минус (-).

Для разделения имени команды, опций и аргументов используется **пробельный символ (white space)**. Пробельный символ обычно определяется как один или несколько расположенных подряд кодов, соответствующих нажатию клавиш пробела (Space) или табуляции (Tab). Отсюда, имеется большая разница между строками **banner Hi** и **bannerHi**. Первая строка будет распознана как команда **banner** с одним аргументом, переданным ей (**Hi**). Вторая строка будет интерпретироваться как команда **bannerHi**, имя которой, скорее всего, не является верным.

Ввод каждой команды должен завершаться символом возврата каретки (клавиша Return). Нажатие этой клавиши приводит к передаче команды в компьютер для обработки. На следующих слайдах завершающий Return будет подразумеваться, но изображаться обычно не будет.

Организация каналов ввода/вывода терминалов допускает упреждающий ввод. Это позволяет ввести первую команду и затем вводить следующую команду (одну или несколько) до того, как на экране появится соответствующее приглашение. Введенная таким

образом команда будет запомнена в буфере команд и выполнена после того, как завершится выполнение первой.

В одной командной строке может быть введено несколько команд, разделенных символом "точка с запятой".

ПРИМЕЧАНИЕ: Ввод команд в системе UNIX является *чувствительным к регистру клавиатуры*. Большинство команд и опций вводятся в нижнем регистре (используются строчные буквы). Поэтому **banner hi** является правильной командой, тогда как команда **BANNER hi** не будет понята системой.

ПРИМЕЧАНИЕ: В одной командной строке можно ввести две команды, разделив их символом "точка с запятой". Например:

\$ ls ; pwd

2–5. СЛАЙД: Вторичное приглашение

Вторичное приглашение

```

$ banner 'hi' Return Введен открывающий апостроф
> there' Return Введен закрывающий апостроф

#      #      #      ##### # # ##### ##### #####
#      #      #      #      # # #
##### #      #      ##### # # #####
#      #      #      #      # # ##### #
#      #      #      #      # # #
#      #      #      #      # # # ##### #
#      #      #      #      # # # #####

$ ( Return Введена левая круглая скобка
> Ctrl + c

$ if Return Введено предложение if
> Ctrl + c

$

```

Пояснения

Bourne, Korn, и POSIX shell'ы позволяют интерактивно вводить команды, размещенные на нескольких строках экрана. Если shell требует для завершения команды продолжить ее ввод, на экране после нажатия клавиши "возврат каретки" появится вторичное приглашение (>). Некоторые команды требуют ввода "закрывающих" команд, так же как некоторые символы требуют ввода "закрывающих" символов. Например, команда **if** требует для завершения ввода команды **fi**, открывающая (левая) круглая скобка требует ввода правой скобки, а открывающий апостроф требует ввода закрывающего апострофа.

Если команда введена неправильно, как показано на слайде, shell выводит вторичное приглашение. Определена специальная последовательность клавиш для прерывания текущей исполняющейся программы. Обычно одновременное нажатие клавиш Ctrl и c (в дальнейшем одновременное нажатие клавиш будет обозначаться символом «+»): Ctrl + c приводит к завершению выполняющейся программы и выводу первичного приглашения shell'a. Чтобы определить, какие клавиши используются для прерывания программ в текущем сеансе работы, следует выполнить команду **stty -a**.

2–6. СЛАЙД: Справочник

Справочник

Справочник HP-UX содержит:

Раздел	Описание
Раздел 1:	Команды пользователя
Раздел 1m:	Команды обслуживания системы (ранее раздел 8)
Раздел 2:	Системные вызовы
Раздел 3:	Функции и библиотеки функций
Раздел 4:	Форматы файлов
Раздел 5:	Смешанные темы
Раздел 7:	Файлы устройств (специальные файлы)
Раздел 9:	Глоссарий

Пояснения

Термин "Справочник" означает *Справочное руководство HP-UX*. *Справочник* очень полезен для выяснения синтаксиса команд, но не может быть использован в качестве учебника. Также справочник не очень подходит для изучения, как использовать ОС UNIX. Опытные пользователи системы UNIX обращаются к справочнику для того, чтобы разобраться с отдельными деталями команд и их использованием. *Справочник* разделен на разделы, что показано на слайде.

Ниже приводится краткое описание каждого раздела.

- Раздел 1 Команды пользователя
В этом разделе приводится описание программ, выполняемых непосредственно пользователями или вызываемых из shell процедур. В общем случае они доступны для исполнения любому пользователю системы.
- Раздел 1M Обслуживание системы
В этом разделе приводится описание команд, которые использует администратор системы для ее обслуживания. В общем случае они доступны только пользователю **root** (только **root** может их исполнять) – это входное имя ассоциируется с администратором системы.
- Раздел 2 Системные вызовы
В этом разделе приводится описание функций, которые определяют программный интерфейс с ядром системы UNIX (интерфейс на языке C).

- Раздел 3 Функции и библиотеки функций
В этом разделе приводится описание функций, которые представлены в системе в виде объектных модулей и которые не являются системными вызовами. Эти функции обычно можно использовать в программах, написанных на языке С. Примерами таких функций являются функции ввода/вывода и математические функции.
- Раздел 4 Форматы файлов
В этом разделе приводится описание полей конфигурационных файлов системы (таких как **/etc/passwd**), а также определяется структура файлов различных типов (таких как **a.out**).
- Раздел 5 Смешанные темы
В этот раздел включена разнообразная информация, например, описание заголовков файлов, наборов символов, пакетов макросов и другие темы.
- Раздел 7 Специальные файлы (файлы устройств)
В этом разделе обсуждаются характеристики специальных файлов, которые обеспечивают связь между системой UNIX и системными устройствами ввода/вывода (диски, ленты, принтеры).
- Раздел 9 Глоссарий
В этом разделе дается определение терминов, используемых в *Справочнике*.

В каждом разделе команды перечисляются в алфавитном порядке. Чтобы найти интересующую его команду, пользователь должен обратиться к индексному указателю *Справочника*.

Система Linux



В справочнике системы Linux используется несколько иное распределение разделов, что показано ниже:

- | | |
|----------|------------------------------|
| Раздел 1 | Команды пользователя |
| Раздел 2 | Системные вызовы |
| Раздел 3 | Функции и библиотеки функций |
| Раздел 4 | Специальные файлы устройств |
| Раздел 5 | Форматы файлов |
| Раздел 7 | Смешанные темы |
| Раздел 8 | Обслуживание системы |
| Раздел 9 | <Не используется> |

2–7. СЛАЙД: Содержание статей справочника

Содержание статей справочника

NAME	EXAMPLES
SYNOPSIS	WARNINGS
DESCRIPTION	DEPENDENCIES
EXTERNAL INFLUENCES	AUTHOR
NETWORKING FEATURES	FILES
RETURN VALUE	SEE ALSO
DIAGNOSTICS	BUGS
ERRORS	STANDARDS CONFORMANCE

Пояснения

Важно понимать формат статей *Справочника*. Везде в документации системы UNIX ссылки даются в виде **команда(n)**, где **команда** – это имя команды, а **n** – номер раздела справочника. Так, **date(1)** ссылается на команду **date** из раздела 1 *Справочника*. В каждом разделе команды представлены в алфавитном порядке. В соответствии с принятым порядком работы со *Справочником*, сквозная нумерация страниц в нем не используется. Описание каждой команды начинается со страницы 1.

Каждая статья *Справочника* (описание некоторых команд размещается более чем на одной странице) имеет несколько основных заголовков, каждый из которых начинает соответствующий раздел статьи. Статьи *Справочника* не всегда имеют все перечисленные заголовки (и, соответственно, разделы).

Ниже перечислены все основные заголовки и дается краткое описание соответствующих разделов статей *Справочника*.

NAME	НАЗВАНИЕ: здесь приводится имя команды и ее краткое описание. Текст данного раздела используется для создания индекса.
SYNOPSIS	СИНОПСИС: здесь приводится описание формата вызова команды. Части команды, выделенные полужирным шрифтом , должны вводиться в точности

так, как они представлены в описании. Части, заключенные в квадратные скобки ([]), не обязательны. Части команды, представленные обычным шрифтом, заменяются пользователем конкретным текстом в соответствии с его потребностями. Круглые скобки с многоточием внутри (...) используются для того, чтобы показать, что предшествующий аргумент может быть повторен. Если есть сомнения в правильности понимания информации данного раздела, следует обратиться к разделу DESCRIPTION.

DESCRIPTION	ОПИСАНИЕ: здесь приводится подробное описание назначения команды и каждой ее опции.
EXTERNAL INFLUENCES	ВНЕШНИЕ ВЛИЯНИЯ: здесь приводится информация о программировании с учетом особенностей различных национальных языков.
NETWORKING FEATURES	СЕТЕВЫЕ ВОЗМОЖНОСТИ: здесь приводится описание функционирования в зависимости от характеристик той или иной сети.
RETURN VALUE	КОД ЗАВЕРШЕНИЯ: здесь приводится описание значений, возвращаемых по завершении работы вызванной программы.
DIAGNOSTICS	ДИАГНОСТИКА: здесь поясняются сообщения об ошибках, которые могут возникнуть при выполнении команды.
ERRORS	ОШИБКИ: здесь перечислены возможные ошибки и соответствующие им сообщения об ошибках или возвращаемые значения.
EXAMPLES	ПРИМЕРЫ: здесь приводятся примеры использования команд.
WARNINGS	ПРЕДУПРЕЖДЕНИЯ: здесь описываются возможные "подводные камни", которые могут возникнуть при выполнении команд.
DEPENDENCIES	ЗАВИСИМОСТИ: здесь описываются особенности функционирования системы UNIX, связанные с использованием специфических аппаратных средств.
AUTHOR	АВТОР: здесь приводятся сведения о разработчике команды.
FILES	ФАЙЛЫ: здесь описываются специальные файлы, используемые командой.
SEE ALSO	СМ. ТАКЖЕ: здесь перечисляются ссылки на другие статьи <i>Справочника</i> или другую документацию, где можно найти дополнительную информацию.

BUGS

ДЕФЕКТЫ: здесь обсуждаются известные ошибки и недостатки, выявленные при тестировании, а также ошибки, которые маловероятны, но могут возникнуть.

STANDARDS CONFORMANCE

СООТВЕТСТВИЕ СТАНДАРТАМ: здесь приводятся стандарты, которым соответствует данная команда.

2–8. ТЕКСТОВАЯ СТРАНИЦА: Справочник – пример

banner (1)

banner (1)

NAME

banner - make posters in large letters

SYNOPSIS

banner strings

DESCRIPTION

banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

Each argument is printed on a separate line. Note that multiple-word arguments must be enclosed in quotes in order to be printed on the same line.

EXAMPLES

Print the message `'Good luck Susan'` in large letters on the screen:

```
banner "Good luck" Susan
```

The words Good luck are displayed on one line, and Susan is displayed on a second line.

WARNINGS

This command is likely to be withdrawn from X/Open standards. Applications using this command might not be portable to other vendors' platforms.

SEE ALSO

`echo(1)`.

STANDARDS CONFORMANCE

banner: SVID2, SVID3, XPG2, XPG3

2–9. СЛАЙД: Оперативный справочник

Оперативный справочник

Синтаксис:

man [**-k** | **X**] *ключевое_слово* | *команда*

где **X** — номер одного из разделов *Справочника*

Примеры:

\$ man date	<i>Вывести статью Справочника "date"</i>
\$ man -k copy	<i>Вывести статьи с ключевым словом "copy"</i>
\$ man passwd	<i>Вывести статью "passwd" - Раздел 1</i>
\$ man 4 passwd	<i>Вывести статью "passwd" - Раздел 4</i>

 Вывести следующую страницу

 Вывести следующую строку

 Выйти из команды **man**

Пояснения

В большинстве систем UNIX *Справочник* доступен в оперативном режиме. Для доступа к нему используется команда **man**.

Синтаксис команды:

man -k *ключевое_слово*

или

man [*номер_раздела*] *команда*

Примеры использования:

man -k *ключевое_слово* выводится перечень всех команд, в описании которых встречается строка *ключевое_слово*.

man [*номер_раздела*] *команда* выводится статья *Справочника* с именем *команда* из указанного раздела.

man *команда* по умолчанию поиск статьи с указанным именем *команда* выполняется последовательно во всех разделах *Справочника*, начиная с первого; выводится

статья, найденная первой (статьи с одинаковым именем могут встретиться в нескольких разделах *Справочника*).

Все эти форматы команд требуют, чтобы системный администратор корректно установил оперативный *Справочник*. В третьем примере, представленном на слайде, команда **man passwd** покажет описание команды изменения пароля. В последнем примере команда **man 4 passwd** выведет описание файла паролей. Когда будет выведена первая страница статьи справочника для указанной команды, команда ожидает от пользователя определенной реакции, вводимой со стандартного ввода; могут быть использованы следующие клавиши:

Return	Вывести следующую строку
Space	Вывести следующую страницу
Q или q	Завершить выполнение команды man и вернуться в shell

Иногда при доступе к оперативному *Справочнику* может появиться сообщение:

```
Reformatting Entry.  Wait...
```

Это сообщение означает, что статья *Справочника* для указанной команды должна быть распакована, так как это первое обращение к данной статье в течение сеанса связи с системой. При следующих обращениях этой же статье *Справочника* такое сообщение уже не появится на экране.

Управление экраном

На клавиатуре терминалов Hewlett-Packard доступны специальные клавиши, помогающие просматривать текст, выводимый командой **man** или другими командами. ПРИМЕЧАНИЕ: эти клавиши являются функциями клавиатур и эмуляторов терминалов HP, а не возможностями системы UNIX.

Prev	Возврат к предыдущему экрану.
Next	Переход к следующему экрану.
Shift + ↓	Перемещение вниз на одну (следующую) строку.
Shift + ↑	Перемещение вверх на одну строку.
Home	Перемещение курсора в первую позицию первой строки.
Clear Display	Очистка экрана от текущего положения курсора до конца экрана.
Home Clear Display	Очистка всего экрана.

Множественность статей Справочника

Статьи, относящиеся к некоторым командам, представлены в нескольких разделах Справочника. Для того, чтобы вывести номера разделов Справочника, в которых есть статьи, относящиеся к интересующей команде, можно использовать команду **whereis**. Например:

```
$ whereis passwd
```

```
passwd: /sbin/passwd /usr/bin/passwd  
/usr/share/man/man1.Z/passwd.1  
/usr/share/man/man4.Z/passwd.$
```

```
whereis nothere
```

```
nothere:
```

Приведенные результаты выполнения этой команды показывают, что статьи, относящиеся к команде **passwd**, имеются в первом и четвертом разделах *Справочника*, а для команды с именем **nothere** статей в *Справочнике* нет.

2–10. СЛАЙД: Некоторые начальные команды

Некоторые начальные команды

id	Вывести идентификатор пользователя и группы
who	Идентифицировать пользователей, работающих в системе
date	Вывести системные дату и время
passwd	Установить пароль пользователя
echo	Вывести на экран простое сообщение
banner	Вывести аргументы большими буквами
clear	Очистить экран терминала
write	Послать сообщение на терминал другого пользователя
mesg	Разрешить/запретить вывод сообщений на свой терминал
news	Вывести системные новости

Пояснения

Здесь представлены несколько простых команд, которые позволят попрактиковаться в передаче команд shell'у системы UNIX. Большинство из представленных команд имеют много других опций кроме тех, которые представлены в данных материалах. Если потребуются и другие опции, следует обратиться к статьям *Справочника* для соответствующих команд.

2–11. СЛАЙД: Команда **id**

Команда **id**

Синтаксис:

id Выводит идентификатор пользователя и группы
 для текущей сессии

Пример:

```
$ id  
  
uid =303 (user3) gid =300 (class)
```

Примечание: Параметр **gid** определяет первичную группу. Если пользователь включен еще и в дополнительные группы, эти группы перечисляются после параметра **groups**

Пояснения

Для организации доступа к файлам и выполнения программ система UNIX должна знать **идентификатор пользователя** и **идентификатор группы**, в которую включен пользователь. Компьютер работает с числовыми идентификаторами; соответствующие им текстовые имена предусмотрены только для удобства пользователей. Числовой идентификатор пользователя устанавливается, когда пользователь начинает сеанс работы в системе. После входа в систему пользователь имеет возможность изменить свой идентификатор или идентификатор группы. Команда **id** выводит текущие идентификаторы пользователя и группы.

Все идентификаторы пользователей, с которыми имеет дело компьютер, хранятся в файле **/etc/passwd**, а все идентификаторы групп – в файле **/etc/group**.

Группы

Группы предоставляют собой средство, позволяющее некоторому подмножеству пользователей разделять доступ к файлам.

Пользователи, которые должны быть включены в определенную группу, определяются администратором системы; каждый пользователь может быть членом одной или нескольких групп. Группы обычно формируются по профессиональному принципу, на основе рабочих подразделений, уже существующих в организации.

Например, в некоторой организации могут существовать производственный отдел (manufacturing), инженерный отдел (engineering) и бухгалтерия (accounting). Назначение пользователей в эти группы может быть представлено следующим образом:

группа:	manufacturing			engineering		accounting		
	-----			-----		-----		
пользователи:	chris*	pat*	mike	chris	joe*	chris	mike*	terry*

Символом * помечены идентификаторы пользователей в их входных группах, т.е. в тех, которые устанавливаются при входе пользователя в систему.

- **chris** является членом всех трех групп, его входная группа **manufacturing**.
- **mike** является членом двух групп, его входная группа **accounting**.
- Остальные пользователи являются членами только одной группы – своей входной.

При такой организации пользователь **chris** имеет право доступа к файлам, которые ассоциированы со всеми тремя группами – **manufacturing**, **engineering** и **accounting**. Пользователь **mike** имеет право доступа к файлам, которые ассоциированы с группами **manufacturing** и **accounting**. Все остальные пользователи имеют право доступа только к тем файлам, которые ассоциированы с их входными группами.

2–12. СЛАЙД: Команда **who**

Команда **who**

Синтаксис:

who [am i] Выводит информацию о пользователях, которые
whoami в данный момент времени работают в системе

Примеры:

```
$ who
root      tty1p5  Jul 01 08:01
user1     tty1p4  Jul 01 09:59
user2     tty0p3  Jul 01 10:01

$ who am i
user2     tty0p3  Jul 01 10:01

$ whoami
user2
```

Пояснения

Команда **who** выводит информацию о тех пользователях, которые в данный момент времени работают в системе, с каких терминалов, и в какое время они вошли в систему. Команда **who am i** выводит имя пользователя и информацию о терминале и времени входа в систему для локальной терминальной сессии. Наконец, команда **whoami** выводит только имя пользователя, которое система ассоциировала с данным терминалом. Право выполнения команд зависит от идентификатора пользователя. Пользователь может интерактивно изменить свой идентификатор, для того чтобы иметь доступ к другим командам и программам.

Система Linux



В среде Linux команда **whoami** перед именем пользователя выводит также имя хост-машины, за которым следует восклицательный знак (!).

2–13. СЛАЙД: Команда **date**

Команда **date**

Синтаксис:

date Выводит текущие дату и время

Пример:

```
$ date  
Fri Jul 1 11:15:55 EDT 1998
```

Пояснения

Команда **date** используется для вывода системных даты и времени. Эта команда имеет аргументы, которые позволяют форматировать вывод.

Обычно команда **date** используется без опций и аргументов, так, как показано на слайде.

В статье *Справочника* **date (1)** упоминается еще и аргумент, который может быть указан для установки даты. *Только администратор системы имеет право модифицировать системные дату и время.*

2–14. СЛАЙД: Команда **passwd**

Команда **passwd**

Синтаксис:

passwd Устанавливает пароль для входа в систему

Пример:

```
$ passwd
Changing password for user3
Old password:
New password:
Re-enter new password:
```

Ограничения на значение пароля:

- не менее шести символов
- по крайней мере, две буквы
- по крайней мере, одна не буква

Пояснения

На многих системах паролями управляет администратор системы. В системе UNIX администратор системы позволяет самому пользователю осуществлять прямое управление своим собственным паролем. Для изменения пароля можно использовать команду **passwd**

Синтаксис команды:

```
passwd
```

При выполнении команды будет запрошен текущий пароль пользователя (старый пароль – old password). Это делается для того, чтобы запретить кому-нибудь другому изменение пароля, если пользователь оставил свой терминал без внимания во время сеанса работы в системе. Если текущий пароль введен верно, команда попросит ввести новый пароль, а затем повторить ввод нового пароля. Повторный ввод нового пароля позволяет защититься от случайных ошибок и опечаток при вводе пароля. Новый и старый пароли должны отличаться друг от друга, по крайней мере, тремя символами.

Когда старый и новый пароли вводятся с клавиатуры, они не отображаются на экране.

Ограничения на значение пароля

Пароль должен состоять, по крайней мере, из шести символов. По крайней мере, два символа из первых шести должны быть буквами и один символ из первых шести не должен быть буквой.

На администратора системы указанные ограничения не распространяются, поэтому если администратор системы устанавливает пароль для какого-либо пользователя, он может не следовать указанным правилам.

2–15. СЛАЙД: Команда **echo**

Команда **echo**

Синтаксис:

echo [*аргумент* ...] Вывести аргумент(ы) на экран терминала

Примеры:

```
$ echo how are you
how are you
```

```
$ echo      123      abc
123 abc
```

Пояснения

Команда **echo** позволяет вывести на экран аргументы командной строки. Например, команда

```
echo hello
```

выведет на экран строку:

```
hello
```

Эта команда может показаться очень простой; она обычно используется в shell процедурах для вывода на экран каких-либо сообщений пользователю или значений переменных shell'a. В разделах, посвященных shell программированию, команда **echo** будет активно использоваться.

2–16. СЛАЙД: Команда **banner**

Команда **banner**

Синтаксис:

banner *аргумент* [*аргумент* ...] Вывести аргументы
большими буквами

Пример:

```
$ banner hello
```

```
#   #   #####   #   #   #####  
#   #   #       #   #   #   #  
#####   #####   #   #   #   #  
#   #   #       #   #   #   #  
#   #   #       #   #   #   #  
#   #   #####   #####   #####
```

Пояснения

Команда **banner** была изначально разработана (и до сих пор так и используется) для пометки вывода на принтер. Команда **banner** выводит аргументы командной строки большими заглавными буквами, располагая по одному аргументу в строке.

Система Linux



В системе Linux команда **banner** предназначена только для использования при выводе на принтер. По умолчанию, пользователь системы не может выполнить команду **banner**, если только он не укажет полное маршрутное имя файла, ассоциированного с этой командой (например, **/usr/games/banner**).

2–17. СЛАЙД: Команда `clear`

Команда `clear`

Синтаксис:

`clear` Очистить экран терминала

Пояснения

Команда `clear` очищает экран терминала, если это можно сделать. Эта команда очищает только текущий экран, так что пользователь (с помощью прокрутки) может восстановить предыдущий экран. Чтобы стереть весь экран, нужно позиционировать курсор в начало экрана, нажав клавишу `HOME`, и затем выполнить команду `clear`.

2–18. СЛАЙД: Команда **write**

Команда **write**

Синтаксис:

```
write имя_пользователя[tty]
```

Послать сообщение пользователю с указанным именем, если этот пользователь работает в системе

Пример:

```
user3
$ write user4
Are you going to the meeting?
```

Message from user4 (tty52)
I will be there.

```
I won't be there. Take good notes!
Ctrl + d
```

user4

Message from user3 (tty05)
Are you going to the meeting?

```
$ write user3
I will be there.
```

Message from user3 (tty05)
I won't be there. Take good notes!

Пояснения

Для того чтобы вывести сообщение на экран терминала другого пользователя, который в данное время работает в той же системе UNIX (на том же компьютере), может быть использована команда **write**. Команда **write** в процессе работы предоставляет пользователю возможность ввести свое сообщение. При каждом нажатии клавиши **Return** соответствующая строка передается на экран терминала адресата – пользователя, входное имя которого указано аргументом команды. Пользователь, получивший сообщение, может в свою очередь выполнить команду **write**, чтобы ответить отправителю. Таким образом, можно с помощью своих терминалов вести интерактивный разговор. После того как сообщение введено полностью, следует нажать клавиши **CTRL** + **d**. Нажатие этих клавиш означает завершение своей части разговора.

ПРИМЕЧАНИЕ: Сообщения могут быть посланы на терминал в *любой момент времени*, если пользователь не запретит эту возможность. Поэтому, если пользователь в данный момент времени работает с такими утилитами, как **man**, **mail** или редактор, и кто-то другой выводит на его экран сообщение, это сообщение будет выведено на экран, и вывод от утилит может быть разделен на части.

Если пользователь, которому с помощью команды **write** посылается сообщение, не работает в системе, то при выполнении команды выводится следующее сообщение:

`user is not logged on`

где **user** – имя пользователя-адресата.

2–19. СЛАЙД: Команда **mesg**

Команда **mesg**

Синтаксис:

mesg [**y**|**n**]

Разрешает или запрещает вывод на терминал

Примеры:

```
$ mesg  
is y
```

```
$ mesg n  
$ mesg  
is n
```

```
$ mesg y  
$ mesg  
is y
```

Пояснения

Чтобы запретить другим пользователям вывод сообщений на свой терминал, пользователь может использовать команду **mesg**. Если пользователь посылает сообщение кому-то, кто запретил появление чужих сообщений на своем терминале, то получит сообщение об ошибке: **Permission Denied**.

mesg n Запрещает "запись" на терминал. Это состояние устанавливается по умолчанию в системах HP-UX 10.0 и HP-UX 11.00

mesg y Разрешает "запись" на терминал

mesg Сообщает, разрешена ли "запись" на терминал (**y**) или запрещена (**n**)

Однако даже если пользователь запретит посылать сообщения на свой терминал, администратор системы по-прежнему сможет это сделать.

2–20. СЛАЙД: Команда **news**

Команда **news**

Синтаксис:

news [-a] [-n] [*заголовок*] Выводит системные новости

Примеры:

\$ news	<i>Выводит последние новости</i>
\$ news -a	<i>Выводит все новости</i>
\$ news -n	<i>Выводит заголовки последних новостей</i>

Пояснения

Сообщения, представляющие интерес для всех пользователей системы, могут быть предоставлены им с помощью команды **news**. Администратор системы часто использует команду **news**, чтобы информировать всех пользователей о событиях, имеющих значение для системы, таких как время выгрузки системы, время выполнения резервного копирования, информация о новой аппаратуре, которая будет доступна пользователям.

Прочитать системные новости можно, выполнив команду **news**. Когда выполняется команда без опций, на экран будут выводиться только те сообщения, которые пользователь еще не читал.

В команде **news** возможны следующие опции:

- a** Прочитать все новости независимо от того, были они прочитаны или нет.
- n** Вывести на экран только заголовки тех новостей, которые пользователь еще не читал.

Каждый пользователь, который выполнял команду **news**, в своем входном каталоге имеет файл **.news_time**. Каждый файл в системе UNIX имеет временную метку, которая определяет время последней модификации файла. Временная метка файла **.news_time** обновляется в соответствии со временем последнего прочтения новостей. Если в средства

управления новостями добавляются новые сообщения, **news** знает, что они еще не прочитаны, так как временная метка файла **.news_time** предшествует времени появления нового сообщения в пакете news.

Система Linux



В системе Linux нет команды, эквивалентной команде **news**.

2–21. УПРАЖНЕНИЯ: Вход в систему и простейшие действия

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы. Для выполнения некоторых упражнений вам может понадобиться *Справочник HP-UX*.

1. Войдите в систему, используя входное имя и пароль, выданные вам преподавателем. Возникли ли у вас какие-либо трудности?
2. Выйдите из системы, используя клавиши `CTRL` + `d` или команду **exit**. Заметили ли вы что-нибудь? Вновь войдите в систему.
3. Какие из приведенных ниже команд синтаксически правильны? Попробуйте выполнить их, чтобы увидеть, что будет выведено на экран – результаты выполнения команды или сообщения об ошибках.

```
$ echo
$ echo hello
$ echohello
$ echo HELLO WORLD
```

4. Только для системы HP-UX

Попробуйте выполнить следующие команды, чтобы увидеть, что именно будет выведено на экран:

```
$ banner
$ banner hello
$ BANNER hello
```

5. Установите для себя пароль (или измените пароль, если он уже установлен). Не забудьте правила формирования пароля.
6. Используя варианты команды **who**, ответьте на следующие вопросы. Какие команды вы использовали?
Кто работает в системе?
С какого терминала вы вошли в систему?
Кто вы для системы (ваш входной идентификатор)?
7. Могут ли другие пользователи послать сообщение на ваш терминал? С помощью какой команды вы сможете узнать это?
8. Проверьте, вошел ли в систему ваш коллега, и пошлите сообщение на его терминал. Организуйте диалог с ним. Пошутите друг с другом.

Что случится, если вы попытаетесь послать сообщение своему коллеге, когда он еще не вошел в систему? Что произойдет, если ваш коллега запретил вывод чужих сообщений на свой терминал?
9. Прочитайте системные новости. Какую команду вы использовали? Сможете ли вы повторно вывести новости, которые вы уже прочитали? (Обратите внимание на то, что это упражнение можно выполнить только в системе HP-UX)

10. Выполните команду **date** с соответствующими аргументами, так чтобы дата была выведена в формате *mm-dd-yy*. ПОДСКАЗКА: посмотрите примеры, приведенные в статье **date (1)** *Справочника HP-UX*.
11. Используя команду **man**, найдите описание команды **cp**. Что это за команда? Какое минимальное количество аргументов она требует?
12. Используя команду **man**, найдите описание команды **ls**. Что это за команда? Какое минимальное количество аргументов она требует?
13. Выполните команду **ls -l /usr/bin**. Вы увидите несколько экранов, для просмотра которых потребуется использовать прокрутку.

Только для HP-UX:

С помощью клавиши "стрелка вверх" переместите курсор в начало экрана. Выполните команду **clear**. Осталось что-нибудь на экране? Нажмите вместе с клавишей **[Shift]** клавишу "стрелка вниз", чтобы выполнить прокрутку вниз. Появится ли на экране фрагмент результатов команды **ls -l**?

14. Выйдите из системы. Вновь войдите в систему при включенном режиме CAPS Lock. Как можно исправить эту ситуацию в текущем сеансе работы, т.е. без того чтобы выйти из системы и потом вновь войти в нее? (ПОДСКАЗКА: посмотрите статью **stty** *Справочника*.)

Модуль 3 — Использование CDE

Цели:

Изучив этот модуль, вы сможете:

- Описывать элементы передней панели
- Понимать, как работает всплывающее меню передней панели
- Описывать переключение рабочих областей
- Описывать управление дополнительными панелями
- Понимать, как использовать систему помощи
- Описывать систему управления файлами (менеджер файлов)
- Понимать, как используется меню менеджера файлов
- Использовать менеджер файлов для поиска файлов
- Удалять файлы
- Печатать файлы, используя переднюю панель, менеджер файлов и менеджер печати
- Выводить информацию о системе буферизации принтера
- Понимать управление принтером
- Использовать текстовый редактор
- Использовать менеджер приложений для запуска приложения
- Использовать почтовую службу и ее опции, а также знать, как создаются почтовые ящики (Mailboxes)
- Использовать менеджер календаря для планирования встреч и создания элементов "выполнить"
- Описывать, как можно просматривать другие календари в сети
- Описывать, как разрешать или запрещать доступ к своему календарю

Календарь	Выводит текущую дату. Эта иконка активизирует специальное приложение – календарь, которое позволяет пользователю управлять, планировать и просматривать встречи. Эта утилита позволяет также просматривать другие календари по сети и планировать по сети встречи, если установлены соответствующие права доступа.
Менеджер файлов	Выводит в виде икон файлов, папки и приложения системы. Пользователь может работать с файлами без необходимости заучивать сложные команды. Такие операции, как копирование файлов, их перемещение, удаление, вывод на печать и изменение прав доступа к файлам (перечислены далеко не все возможности), могут быть легко выполнены с помощью меню менеджера файлов.
Текстовый редактор	Запускает простой текстовый редактор с общими возможностями, включающими и взаимодействие с другими приложениями через буфер. Это приложение может быть также использовано в качестве зоны перетаскивания из менеджера файлов. По умолчанию на передней панели выводится иконка текстового редактора. Здесь есть еще и дополнительная панель Personal Application (личные приложения), которая может активизировать терминал или иконку редактора.
Почтовый сервис	Активизирует графический интерфейс (GUI) со средствами электронной почты. Этот пакет может быть использован в качестве зоны перетаскивания файлов или календарей, которые должны быть отправлены по почте другим пользователям сети.
Кнопка блокировки	Позволяет пользователям блокировать экран, если он остается какое-то время без внимания. Можно так настроить работу, что блокировка через какое-то время будет установлена автоматически, если в течение этого времени не проводилась работа с экраном. Чтобы восстановить доступ к компьютеру, пользователь должен ввести свой пароль.
Рабочие области	По умолчанию доступны четыре изолированных экрана (рабочие области), на которых могут быть открыты окна, однако количество рабочих областей является настраиваемым параметром. Приложения могут быть организованы в особую, обычным способом именованную рабочую область. Кроме того, окна, представленные в одной рабочей области, могут быть скопированы в другую рабочую область.
Exit (выход)	Позволяет пользователям завершить работу на компьютере. Все не сохраненные работы будут потеряны. По умолчанию пользователю выводится сообщение, требующее подтвердить завершение работы.
Менеджер печати	Система управления печатью с простым графическим интерфейсом (GUI), которая позволяет планировать и управлять заданиями на печать на любой доступный принтер.
Менеджер стиля	Позволяет пользователям легко настраивать такие свойства компьютера, как цвета, фон, размер шрифта и поведение системы.

Менеджер приложений (Application Manager). Предоставляет доступ к приложениям с помощью иконок. Для того чтобы выполнить некоторое приложение, пользователь может щелкнуть мышью по его иконке. Менеджер приложений состоит из групп приложений (Application Groups), каждая из которых включает в себя приложения, соответствующие определенной функции. Пользователь имеет возможность создать свою собственную группу приложений, а также включать свои собственные приложения как в новую, так и в существующие группы.

Менеджер помощи Для любого стандартного приложения в CDE доступна оперативная помощь. Всплывающее дополнительное меню предоставляет доступ к менеджеру помощи – специальному тому помощи, который перечисляет все средства оперативной помощи, которые зарегистрированы в системе: Desktop Introduction помогает пользователю понять, как перемещаться в системе; Front Panel Help предоставляет содержание помощи о передней панели; On Item Help – интерактивная система, позволяющая пользователю переместить указатель на определенную тему и, щелкнув по нему мышью, вывести на экран соответствующую информацию. Кроме того, другие приложения, установленные на рабочем столе, могут воспользоваться системой помощи рабочего стола.

Корзина Собирает файлы и папки, удаленные пользователем. Эти файлы и папки в действительности не удаляются из системы до тех пор, пока корзина не будет опустошена. До этого момента пользователь может восстановить те файлы, которые он удалил. Этот элемент управления может быть использован в качестве зоны перетаскивания.

Справочник CDE

- B1171-90101 *CDE 1.0 User's Guide*

Система Linux

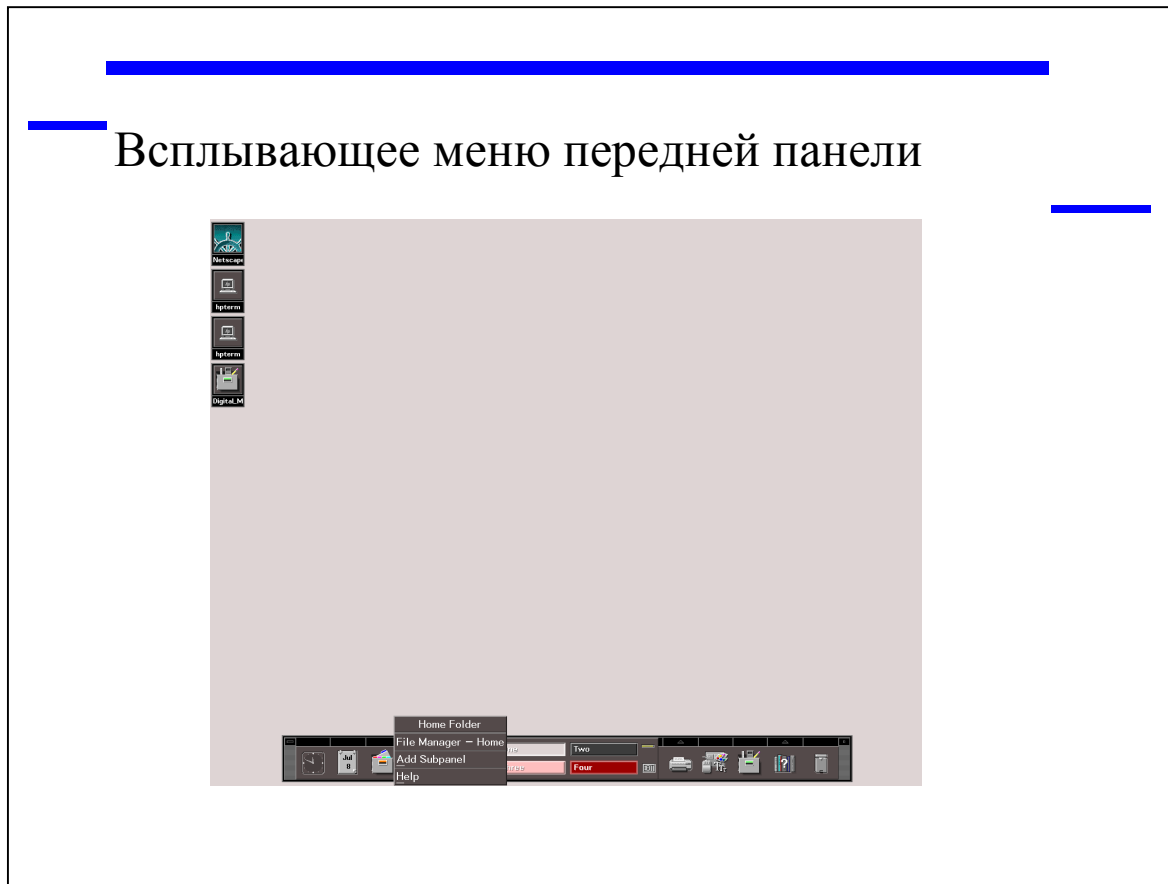


Система Red Hat Linux предусматривает две графические среды пользователя – Gnome и KDE.

KDE в большой степени похожа на CDE. Предполагается, что она будет использована конечными пользователями системы Linux. Однако, различия между средами CDE и KDE слишком значительны, чтобы о них говорить в данном примечании. Правило "look and feel" (смотри и действуй) среды KDE в большой степени похоже на CDE, так что конечные пользователи после изучения данного модуля не будут испытывать больших затруднений, знакомясь с различными приложениями и утилитами, предоставляемыми KDE.

В данном курсе среда CDE будет рассмотрена очень подробно.

3–2. СЛАЙД: Всплывающее меню передней панели



Пояснения

Всплывающее меню – это то, что "всплывает", когда вы щелкаете правой кнопкой мыши в окне приложения или на объекте рабочей области. Каждый элемент управления на передней панели имеет всплывающее меню, свое для каждого элемента управления. Чтобы увидеть всплывающее меню передней панели, нужно установить указатель мыши на элемент управления и щелкнуть правой кнопкой мыши.

В зависимости от элемента управления, содержание всплывающего меню будет разным. Например, если элементом управления является приложение, первым элементом меню будет команда, которая запускает это приложение. Если объект не является приложением, в зависимости от назначения данного объекта будет доступен различный набор альтернатив.

Элементы дополнительной панели, как и главная панель, также имеют всплывающие меню. Например, если щелкнуть мышью по стрелке, расположенной над элементом управления "Текстовый редактор", появится дополнительная панель личных приложений. Теперь установите указатель мыши на элемент Терминал и нажмите правую кнопку. Вы получите всплывающее меню, которое позволит вам скопировать элемент управления на главную панель, удалить элемент управления или получить справку.

Всплывающие меню доступны также и из приложений. Например, в менеджере файлов можно выполнять какие-либо действия над выделенными объектами (например, файлами или каталогами), просто установив курсор на объект и нажав правую кнопку мыши.

3–3. СЛАЙД: Переключение рабочих областей

Переключение рабочих областей



Пояснения

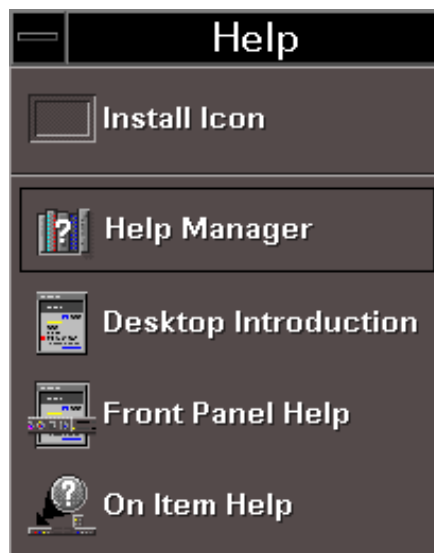
По умолчанию определены четыре рабочие области. Каждая рабочая область занимает весь экран дисплея. Переключатель рабочих областей состоит из кнопок, используемых для перехода от одной рабочей области к другой. Кнопка, которая соответствует текущей рабочей области, отображается в нажатом виде. Чтобы переключиться на другую рабочую область, достаточно просто щелкнуть мышью по кнопке, которая соответствует новой рабочей области. Действия, выполняемые в одной рабочей области, сохраняются при переключении на другую рабочую область.

Переключатель рабочих областей также имеет всплывающее меню, которое появляется, если щелкнуть правой кнопкой мыши на части переключателя рабочих областей, которая не занята другими элементами управления или кнопками переключателя.

Кроме этого, каждая кнопка рабочей области имеет собственное всплывающее меню, которое можно использовать для добавления новой рабочей области, переименования или удаления той рабочей области, на которую указывает указатель мыши.

3–4. СЛАЙД: Служба помощи

Служба помощи



Пояснения

Оперативная помощь доступна для каждого стандартного приложения в рамках CDE. Линейка меню для каждого приложения имеет в крайней правой позиции элемент **Help** (Помощь), или же для большинства приложений пользователь может нажать клавишу F1, чтобы получить контекстно-зависимую помощь. Кроме того, на передней панели есть элемент управления Менеджер помощи (**Help Manager**), который представляется в виде специального тома, в котором перечислены все темы оперативной помощи, зарегистрированные в системе. Выбрав любой подчеркнутый заголовок, можно увидеть дополнительный список тем, включенных в систему помощи для данного заголовка.

Дополнительная панель системы помощи предоставляет также доступ к элементам **Desktop Introduction** и **Front Panel Help**, а также к элементу **On Item Help**, который позволяет с помощью мыши указать на конкретный элемент и щелчком вывести соответствующие страницы информации (помощи).

Окно помощи

Система помощи, которая встроена в каждое приложение CDE, предоставляет два типа окон помощи – общая помощь и быстрая помощь. Быстрая помощь имеет только тематическую область вывода, в которой выводится информация по запрошенному предмету. Окно общей помощи имеет две области – дерево тем и тематическая область вывода. Дерево тем в основном представляет список главных тем объема помощи. Под главными темами выводятся вложенные темы. Тему можно выбрать, щелкнув по ней в дереве тем мышью.

Чтобы открыть окно общей помощи, нужно активизировать приложение CDE (например, менеджер файлов). В верхнем правом углу линейки меню можно увидеть элемент **Help**. Чтобы открыть общее окно помощи, щелкните мышью по **Help**, а затем по **Overview**. Над текстом помощи и под линейкой меню можно увидеть дерево тем системы помощи для менеджера файлов.

Другой способ перехода между темами помощи – использование гиперссылок. Если выведенная тема содержит подчеркнутые слова, система помощи "перепрыгнет" на соответствующую тему в случае, если пользователь щелкнет мышью по подчеркнутому слову.

Поиск тем с помощью индекса помощи

Как только пользователь откроет общее окно помощи, он может выполнять поиск на основе ключевых слов или шаблонов, используя индекс помощи следующим образом:

1. Чтобы открыть индекс в рамках приложения, щелкните по кнопке **Index**. Индекс позволяет просматривать все элементы текущего тома помощи, всех томов или только выбранного тома.
2. Выберите поле **Entries with**, введите слово или фразу, которые нужно найти, и нажмите клавишу **Return**.
3. Выберите элемент индекса, который нужно просмотреть. Если этот элемент имеет префикс +, этот список можно раскрыть, чтобы показать дополнительные элементы для выбора. Выберите нужную тему для просмотра.

Поиск по шаблону

Система помощи позволяет задавать тему для поиска в виде некоторого шаблона поиска. При задании шаблона поиска допускается использовать следующие специальные символы:

- | | |
|----------------------|--|
| * (звездочка) | Соответствует любой строке символов (в том числе и пустой) |
| ? (знак вопроса) | Соответствует одному любому символу |
| . (точка) | Соответствует любому символу |
| (вертикальная черта) | Указывает два шаблона поиска, соответствие устанавливается с одним из них (любым – логическое ИЛИ) |
| () (круглые скобки) | Задаёт выражение в шаблоне |

Вывод статей оперативного *Справочника*

Вывод статей оперативного *Справочника* осуществляется из менеджера приложений, а не из системы помощи.

1. Щелкните по управляющему элементу менеджера приложений на передней панели.
1. Дважды щелкните на иконке **Desktop Apps**.
2. Щелкните на иконке **Man Page Viewer**.

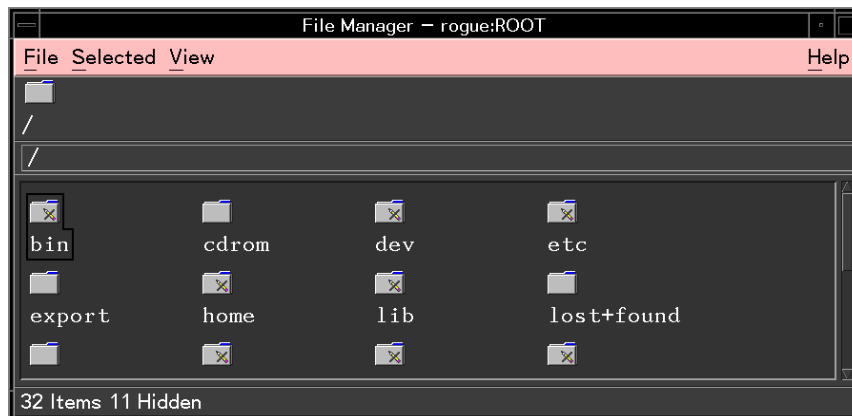
3. Введите название статьи *Справочника*, которую вы хотите прочесть, и нажмите Return.
4. Выберите **close**, чтобы убрать статью *Справочника*.

Вывод на печать тем помощи

После того, как вы вывели текст помощи по интересующей вас теме, вы можете напечатать его, выбрав альтернативы **File->Print** на линейке меню.

3–5. СЛАЙД: Менеджер файлов

Менеджер файлов



Пояснения

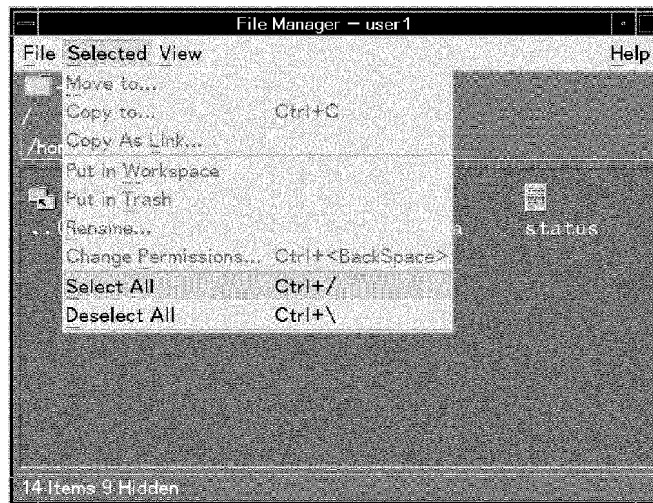
Менеджер файлов позволяет манипулировать файлами (перемещать, копировать, печатать и т.д.), без необходимости заучивать сложные команды. Файлы, папки (каталоги) и приложения выводятся в виде иконок. Активизировать менеджер файлов можно, щелкнув по управляющему элементу менеджера файлов на передней панели.

При первом обращении к менеджеру файлов пользователь попадает во входной (*HOME*) каталог. На каталоги здесь ссылаются как на папки. Папка, в которой пользователь находится в данный конкретный момент времени, определяется как *текущая папка*, и в области отображения объектов будут показаны объекты (файлы и каталоги) из текущей папки.

Высвечивание объекта

Двойной щелчок по файлу откроет текстовый редактор (если этот файл текстовый), в котором будет открыт для редактирования данный файл. Двойной щелчок по папке приведет к тому, что менеджер файлов переместит вас в эту папку. Другие операции выполняются с помощью линейки меню менеджера файлов. Чтобы выбрать файл или папку, нужно, прежде всего, выбрать их, щелкнув по ним и высветив их. Чтобы высветить несколько файлов или папок, нужно нажать левую кнопку мыши в пустой части области отображения объектов, протащить мышь так, чтобы был нарисована рамка вокруг иконок, которые нужно выбрать, и затем отпустить кнопку мыши. После того, как объект(ы) выбран(ы), с ним(и) можно выполнять требуемые операции, используя меню менеджера файлов. Если не выбран ни один объект, некоторые из элементов меню будут недоступны. Например, в меню

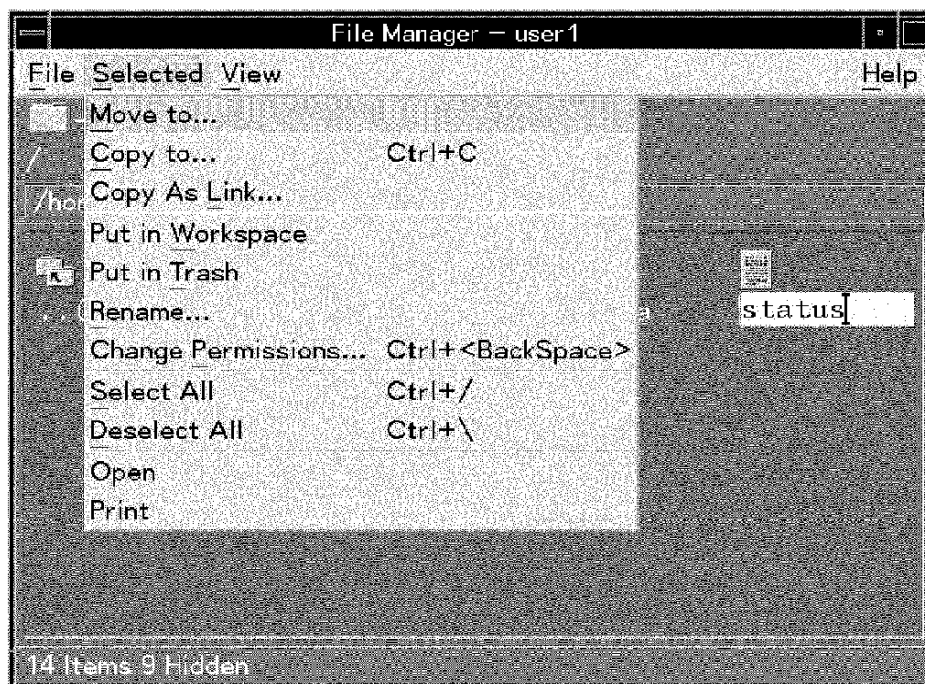
File Manager -> **Selected** многие из действий не высвечиваются, если не выбраны объекты, с которыми нужно выполнить эти действия (Рисунок 1).



a576125

Рисунок 1

Как только будет выбран хотя бы один объект, для выбранных объектов будут высвечены и станут доступными все действия, которые можно выполнять с файлами и/или каталогами (Рисунок 2).



a576126

Рисунок 2

Drag-and-Drop

Объекты можно перетаскивать или в другие области в пределах менеджера файлов (например, в другую папку), или в другую область рабочего стола, например, на принтер, в корзину или рабочую область. Это делается следующим образом:

1. Установите указатель над файлами или папками, которые нужно перетащить.
2. Нажмите и удерживайте левую кнопку мыши.
3. Протащите иконку в ту область, в которой нужно разместить выбранные объекты (т.е. на принтер, в корзину или другую папку и т.д.).
4. Освободите кнопку мыши.

Использование всплывающих меню

Во всплывающих меню могут быть использованы необходимые элементы меню. Установите указатель мыши на иконку, которую надо выбрать, нажмите правую кнопку мыши и выберите элемент меню (Рисунок 3).

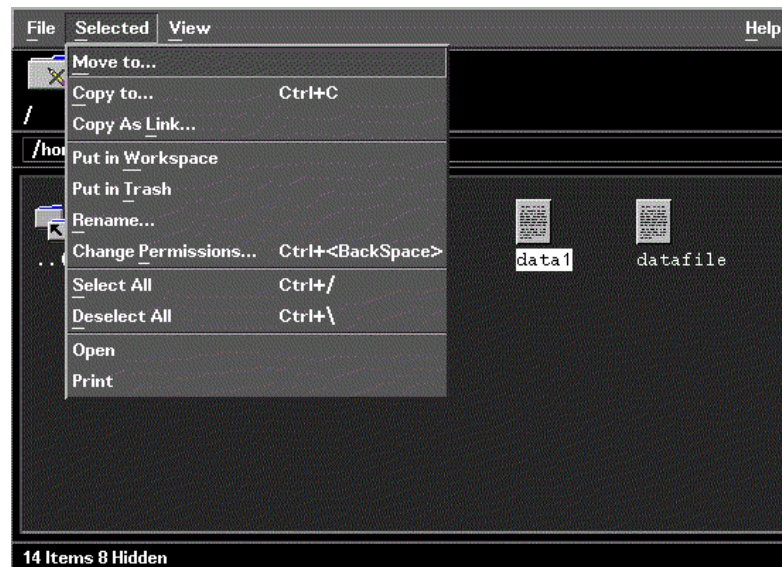


a576127

Рисунок 3

3–6. СЛАЙД: Задачи меню менеджера файлов

Задачи меню менеджера файлов



Пояснения

Операции, относящиеся к файлам, которые можно выполнить из командной строки, могут быть также выполнены с помощью меню менеджера файлов. Для того чтобы выполнить операцию с конкретными файлами или папками, прежде всего их надо выбрать (после того, как файл или папка будут выбраны, их имена будут высвечены). Если будет допущена ошибка при выборе файлов, некоторые операции будут недоступны. После того, как имена требуемых файлов или папок будут высвечены, нужно выбрать опцию **Selected**, чтобы увидеть список доступных операций. Опция меню **View** позволяет управлять тем, какую информацию будет выводить менеджер файлов.

Перемещение или копирование файла или папки

1. Перейдите в ту папку, где находится файл или папка, которые надо копировать или перемещать.
2. Выберите файл или папку, которые надо копировать (перемещать).
3. Выберите операции **Copy to** или **Move to** из меню опции **Selected**. Появится всплывающее диалоговое окно, предлагающее ввести имя папки-адресата, в которую надо скопировать (переместить) файл или папку. Если выполняется операция копирования, будет также предложено ввести имя файла. По умолчанию будет оставлено то же имя, что и у исходного файла.
4. Введите имя адресата.

5. Нажмите .

Копирование файла в виде ссылки

Копирование файла в виде ссылки (будет создана символическая ссылка) в действительности не выполняет копирование данных; вместо копии исходного файла создается копия иконки этого файла. Любые изменения, которые будут сделаны после открытия иконки ссылки, появятся и тогда, когда будет осуществлен доступ к файлу или папке с помощью исходной иконки. Это относится не только к содержимому файла, но также и к характеристикам самого файла. Поэтому если изменяются права доступа или собственности исходного файла, будут точно так же изменены права доступа и собственности ссылки на этот файл. Чтобы создать ссылку на файл:

1. Перейдите в папку, содержащую файл, ссылку на который нужно создать.
2. Выберите файл или папку.
3. Выберите операцию **Copy as Link** из меню **Selected**. Появится всплывающее диалоговое окно, предлагающее ввести имя папки, в которой надо поместить ссылку, и имя ссылки.
4. Введите необходимые имена.
5. Нажмите .

Изменение прав доступа

Чтобы изменить права доступа или собственности файла или папки, нужно быть владельцем этих объектов или администратором системы.

1. Перейдите в ту папку, где находятся файл или папка, характеристики которых вы хотите изменить.
2. Высветите требуемые файл или папку.
3. Выберите операцию **Change Permissions ...** из меню **Selected**.
4. Появится всплывающее диалоговое меню, содержащее текущие значения прав доступа и собственности, а также размер файла и время последней модификации.
5. Введите требуемые данные.
6. Нажмите .

Переименование файла или папки

1. Перейдите в папку, где находится файл или папка, которые вы хотите переименовать.
2. Выберите файл или папку.
3. Выберите операцию **Rename...** из меню **Selected**. Курсор будет установлен в конце текущего имени файла или папки.
4. С помощью клавиши сотрите столько символов имени, сколько нужно, и введите новое имя файла.

5. Нажмите **Return**.

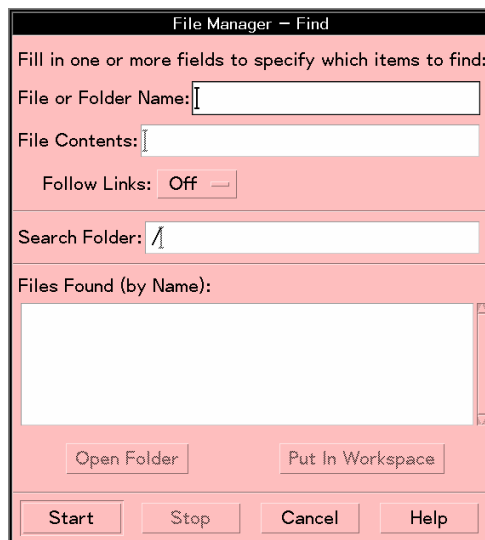
Другие операции меню **Selected**

В зависимости от типа объекта, который представлен иконкой, будут доступны не все из перечисленных ниже операций.

Put on Workspace	Размещает иконку файла или папки на поверхности рабочей области для более легкого доступа к ним. Чтобы удалить эту иконку, нужно переместить на нее курсор мыши, нажать правую кнопку мыши и выбрать Remove From Workspace .
Put In Trash	Перемещает иконку из текущей папки в корзину.
Select All	Выбирает все иконки в текущей папке для выполнения над ними какой-либо операции.
Deselect All	Аннулирует выбор всех иконок в текущей папке.
Open	Предпринимаемое действие зависит от типа выбранного объекта. Если открывается папка, выводится ее содержимое. Если открывается текстовый файл, его содержимое выводится в окне текстового редактора.
Print	Если выбран текстовый файл, его содержимое будет отправлено для печати на принтер, указанный в диалоговом окне. Если выбрана папка, тогда на выбранном принтере будет напечатан список содержащихся в ней файлов и каталогов в длинном формате.
Run	Применяется только для исполняемых файлов. Эта операция приводит к выполнению файла как команды.
Image view	Поместить растровый образ (битовый массив) в иконку редактора.

3–7. СЛАЙД: Использование менеджера файлов для поиска файлов

Использование менеджера файлов для поиска файлов



Пояснения

Менеджер файлов позволяет искать файлы или каталоги по их именам или по содержимому файла.

1. Щелкните по опции меню **File** и выберите **Find**.
2. В поле **File or Folder Name:** введите имя файла или каталога, которые нужно найти. В имени допускается использовать следующие специальные символы:
 - * (звездочка) Соответствует любой (в том числе и пустой) строке символов. Например, если нужно найти все файлы, имена которых начинаются строкой *prog*, нужно ввести **prog***. Это позволит найти файлы с именами *prog*, *progA*, *prog1*, *prog.data*, и т.д.
 - ? (знак вопроса) Соответствует одному любому символу. Если в приведенном выше примере ввести **prog?**, то будут найдены только файлы с именами *progA* и *prog1*.
3. В поле **File Contents** введите текстовую строку, которую надо искать (регистр игнорируется).

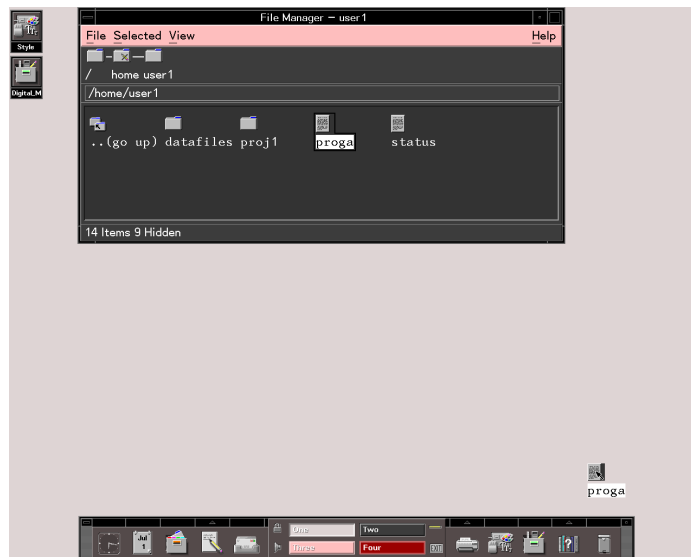
ПРИМЕЧАНИЕ: При поиске файлов не требуется, чтобы были заполнены оба поля – File Name и File Contents. Однако заполнение обоих полей ускоряет поиск, так как критерий поиска сужается.

4. Введите имя папки, с которой вы хотите начать поиск. Операция поиска просмотрит указанную папку и все папки, вложенные в нее.
5. Щелкните по Start.

Когда обнаружено соответствие критерию поиска, имя файла помещается в окно **Files Found**. Как только файлы найдены, можно высветить имя файла и нажать **Open Folder** или **Put in Workspace**, что позволит разместить найденную иконку на поверхности рабочей области.

3–8. СЛАЙД: Удаление объектов

Удаление объектов



Пояснения

Объекты можно удалять двумя способами: с помощью меню **Selected** менеджера файлов или перетаскивая объект в корзину и оставляя его в ней. На самом деле файлы не удаляются из системы, а остаются в корзине до тех пор, пока корзина не будет опустошена. (Управляющий элемент корзины показывает, есть ли в корзине файлы для удаления, с помощью кусочка бумаги, высовывающегося из-под крышки корзины). Пока корзина не будет опустошена, объекты могут быть восстановлены (restore) в исходном месте с помощью менеджера файлов.

Перемещение объекта в корзину

1. Перейдите в папку, содержащую объект, который должен быть удален в корзину.
2. Высветите удаляемый объект.
3. Выберите элемент **Put in Trash** из меню **Selected**, *ИЛИ*
4. Нажмите на левую кнопку мыши и, удерживая ее, перетащите иконку в корзину. Отпустите кнопку мыши. Вы увидите, как корзина откроется и закроется, что говорит о том, что файл будет помещен в корзину.

Восстановление объекта из корзины

1. Дважды щелкните по иконке корзины на передней панели, чтобы открыть окно **Trashcan**. Будет выведен список объектов, которые ранее были удалены.
2. Выберите объект, который нужно восстановить.
3. Щелкните по **File**, чтобы открыть меню, затем выберите **Put Back**. Файл будет восстановлен в том месте, где он находился до удаления.

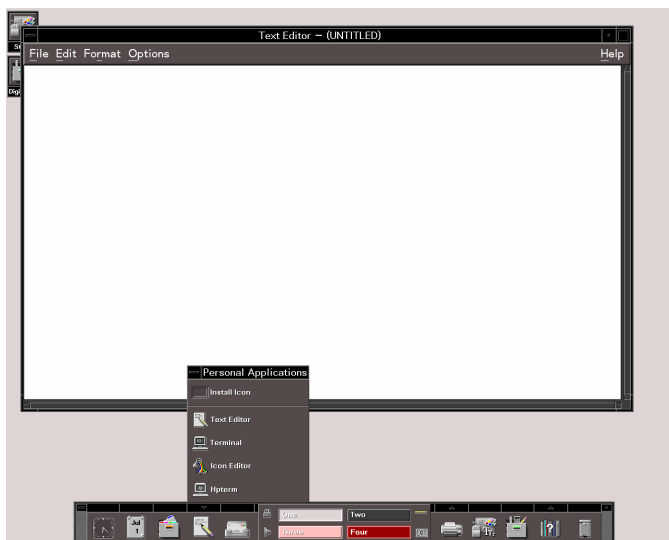
Безвозвратное удаление объектов

Данная процедура действительно удаляет из системы файл или папку. В соответствии с этим, удаленный объект уже не сможет быть восстановлен, если только он не будет сохранен на каком-то внешнем устройстве или скопирован где-то в системе.

1. Откройте окно **Trashcan**, щелкнув по иконке корзины на передней панели.
2. Высветите тот объект, который надо удалить. Если нужно удалить все объекты, откройте меню **File** и выберите **Select All**.
3. В меню **File** выберите **Shred**, чтобы уничтожить объекты. Появится диалоговое окно с просьбой ввести подтверждение выполняемой операции. Если вы уверены в своих действиях, нажмите **OK**. *Помните, что объекты будут удалены безвозвратно!*

3–9. СЛАЙД: Использование текстового редактора

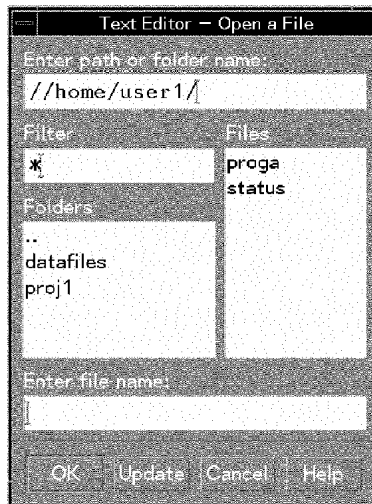
Использование текстового редактора



Пояснения

По умолчанию, управляющий элемент текстового редактора будет доступен из передней панели. Если этот элемент будет изменен (например, на иконку терминала), следует открыть дополнительную панель собственных приложений и выбрать **Text Editor**. Этот элемент запускает программу `/usr/dt/bin/dtpad`, которая может быть также выполнена из командной строки; для этого надо ввести команду **dtpad имя_файла &**.

Как только откроется окно текстового редактора, можно создать новый документ, щелкнув на элементе меню **File** и затем выбрав **New**. Чтобы открыть существующий документ, нужно щелкнуть на элементе меню **File** и выбрать **Open**. Появится окно с приглашением ввести имя файла (Рисунок 4).



a576128

Рисунок 4

Дважды щелкните на имени файла или введите имя файла и нажмите **OK**. В строке заголовка появится имя текущего документа. В качестве имени нового документа выводится **(UNTITLED)**.

Можно включить в текущий документ какой-нибудь другой, если выбрать элемент **Include** из меню **File**. Эта операция модифицирует открытый файл и не влияет на включаемый файл.

Редактирование текста

Перемещение текста (вырезать и вставить – Cut and Paste)

1. Выберите перемещаемый текст; для этого установите курсор мыши на начало того фрагмента, который нужно переместить, нажмите левую кнопку мыши и, удерживая ее, переместите курсор по всему перемещаемому фрагменту текста. Отпустите кнопку.
2. Выберите операцию **Cut** из меню **Edit**. Текст удаляется из документа и запоминается в буфере, откуда в дальнейшем его можно достать.
3. Переместите курсор в ту позицию, куда нужно вставить фрагмент текста.
4. Выберите операцию **Paste** из меню **Edit**.

Копирование текста

1. Выберите копируемый текст; для этого установите курсор мыши на начало того фрагмента, который нужно скопировать, нажмите левую кнопку мыши и, удерживая ее, переместите курсор по всему копируемому фрагменту текста. Отпустите кнопку.
2. Выберите операцию **Copy** из меню **Edit**. Скопированный текст запоминается в буфере.
3. Переместите курсор в ту позицию, куда нужно вставить текст.

4. Выберите операцию **Paste** из меню **Edit**. Если текст нужно скопировать в несколько позиций, достаточно скопировать текст в буфер один раз и затем многократно вставлять содержимое буфера.

Удаление текста

1. Выберите удаляемый фрагмент текста; для этого установите курсор мыши на начало того фрагмента, который нужно удалить, нажмите левую кнопку мыши и, удерживая ее, переместите курсор по всему удаляемому фрагменту текста. Отпустите кнопку.
2. Выберите операцию **Delete** из меню **Edit** или нажмите клавишу **Delete**.

Очистка текста

При очистке текста выбранный фрагмент заменяется пробелами или пустыми строками.

1. Выберите изменяемый текст; для этого установите курсор мыши на начало того фрагмента, который нужно очистить, нажмите левую кнопку мыши и, удерживая ее, переместите курсор по всему изменяемому фрагменту текста. Отпустите кнопку.
2. Выберите операцию **Clear** из меню **Edit**.

Поиск и изменение текста

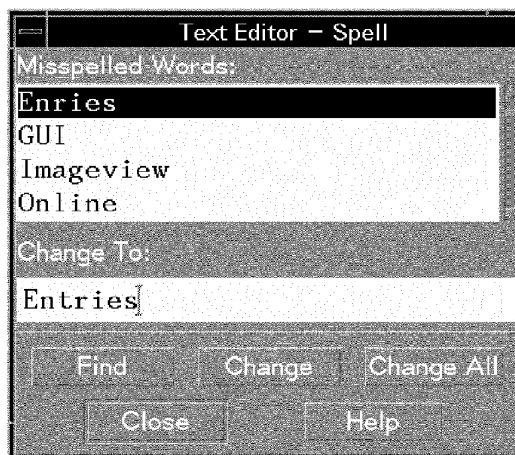
1. Выберите операцию **Find/Change . . .** из меню **Edit**.
2. В поле **Find** введите текст, который нужно найти.
3. В поле **Change To** напечатайте текст, которым надо заменить найденный фрагмент. Если нужно удалить найденный текст, это поле следует оставить пустым.
4. Чтобы начать поиск, нажмите **Return** или щелкните по **Find**.
5. Если соответствующий текст будет найден, курсор будет установлен в соответствующей позиции текста. Чтобы изменить найденный текст, надо щелкнуть по **Change**. Если найденный текст изменять не нужно, но нужно продолжить поиск, нужно щелкнуть по **Find**. Чтобы выполнить все изменения в тексте, щелкните по **Change All**.
6. Чтобы закончить данную операцию, щелкните по **Close**.

Откат редактирования

1. Выберите операцию **Undo** из меню **Edit**. Это приведет к отмене последней операции вырезания, очистки, удаления, изменения, вставки или форматирования.

Исправление опечаток в словах

1. Выберите элемент **Check Spelling** из меню **Edit**. Будет выведено диалоговое окно **Spell**, в котором в поле **Misspelled Words**: находятся слова с ошибками (Рисунок 5).



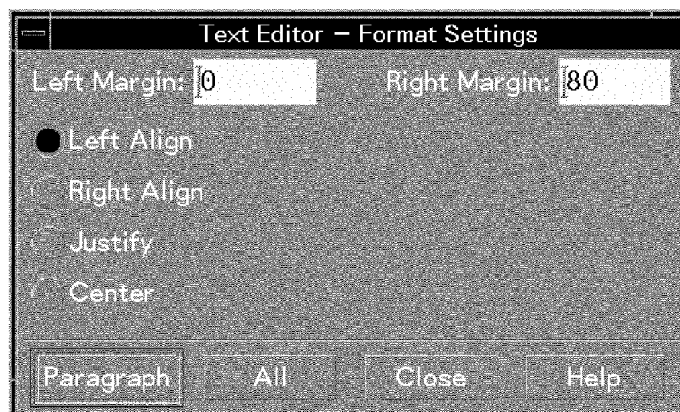
a576129

Рисунок 5

2. Введите правильное слово в поле **Change To**.
3. Щелкните по **Change**, чтобы выполнить одиночное исправление, или по **Change All**, чтобы внести глобальные исправления (во всем тексте). Если нужно только найти слово, содержащее опечатки, и не нужно его исправлять, следует щелкнуть по **Find**.
4. По окончании работы щелкните по **Close**.

Форматирование документа

1. Выберите элемент **Settings** из меню **Format**. Будет выведено диалоговое окно **Format Settings** (Рисунок 6).



a576130

Рисунок 6

2. Введите значения левой и правой границ текста (**Left Margin** – **Right Margin**).
3. Выберите тип выравнивания – по левому (**Left Align**) или правому (**Right Align**) краю, равной ширины (**Justify** – блочный стиль) или по центру (**Center**).
4. Определите область действия форматов:

- чтобы форматировать только параграф, нужно поместить курсор в пределы параграфа, затем щелкнуть по кнопке **Paragraph**.
 - чтобы форматировать весь документ, следует щелкнуть по кнопке **All**.
5. После того, как диалоговое окно закроется, выберите **Paragraph** или **All** из меню **Format**, чтобы применить установки.

Другие опции текстового редактора

Режим замены

Выбор элемента **Overstrike Insert** из меню **Options** позволяет вводить новый текст в режиме замены, а не в режиме вставки. Когда такой режим ввода более не нужен, следует вновь нажать **Overstrike Insert**, чтобы отключить этот режим.

Wrap to Fit

Выбор элемента **Wrap to Fit** из меню **Options** управляет режимом динамического прерывания строк в зависимости от ширины экрана. Когда этот режим включен, строки автоматически прерываются в конце окна. Если размер окна изменяется, соответствующим образом регулируется и разрыв строк.

Строка состояния

Выбор элемента **Status Line** из меню **Options** выводит в нижней части документа строку состояния, в которой указываются текущий номер строки и общее количество строк в документе. В этой строке также появляется соответствующая индикация, когда включается режим замены (**Overstrike mode**).

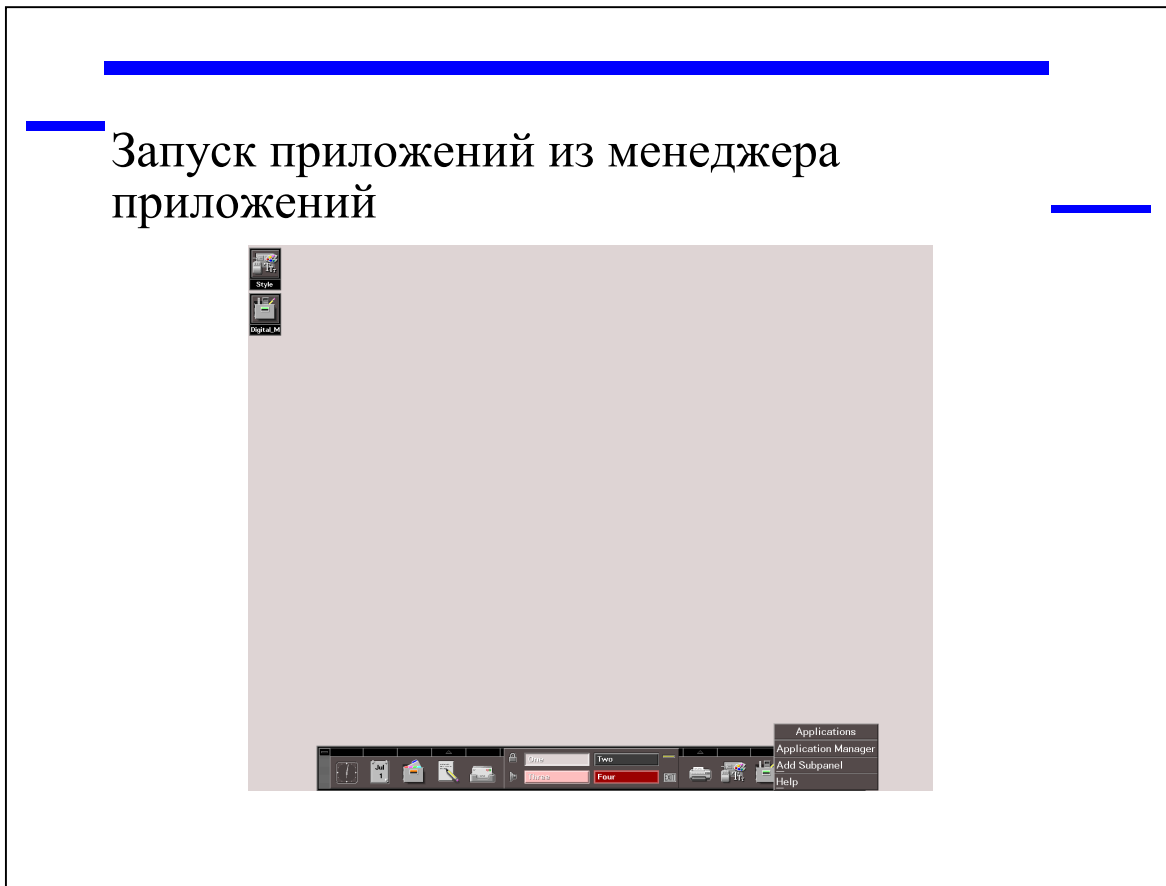
Строка состояния может быть также использована для простого перехода на строку с определенным номером.

1. Выведите строку состояния.
2. Щелкните по полю строки состояния.
3. Введите номер строки, на которую надо перейти, и нажмите **Return**.

Печать документа

1. Выберите элемент **Print** из меню **File**. Появится диалоговое окно принтера, с помощью которого можно решить, какой принтер будет печатать, сколько копий, нужна ли титульная страница, нужна ли нумерация страниц, а также выполнить и другие команды принтера.
2. Щелкните по **Print**.

3–10. СЛАЙД: Запуск приложений из менеджера приложений



Пояснения

Менеджер приложений – это контейнер для приложений и других средств, имеющихся в системе. Большинство приложений и средств встроены в рабочий стол. Настройка системы возможна как системным администратором на системном уровне, так и отдельными пользователями на личном уровне.

Чтобы открыть менеджер приложений, нужно щелкнуть по управляющему элементу менеджера приложений на передней панели.

В верхней части менеджера приложений находятся папки для групп приложений (**Application Groups**), доступных пользователю. Приложения непосредственно хранятся не в верхней части менеджера приложений, а в группах приложений, что позволяет организовать приложения в соответствии с их функциональным назначением.

Чтобы запустить приложение из менеджера приложений, нужно:

1. Открыть менеджер приложений.
2. Дважды щелкнуть по иконке нужной группы приложений, чтобы вывести ее содержимое.
3. Дважды щелкнуть на иконке требуемого приложения, чтобы выполнить его.

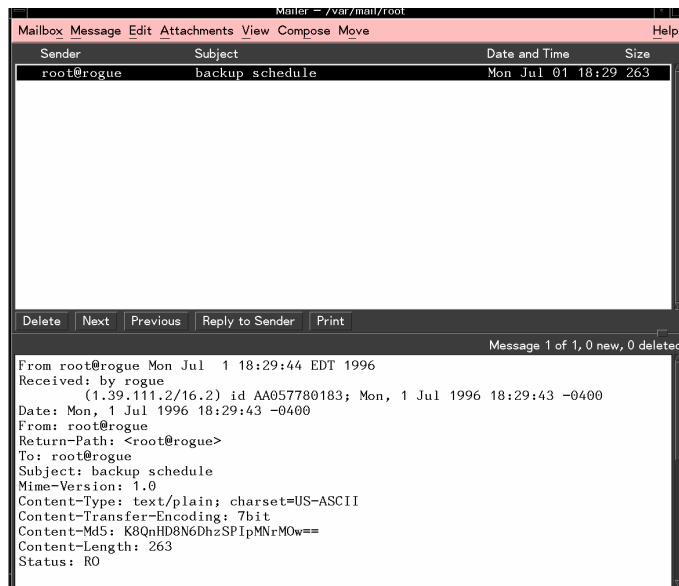
Встроенные группы приложений

Рабочий стол имеет следующие встроенные группы приложений, в которых находятся различные средства и утилиты, доступные в системе:

Desktop_Apps	Приложения рабочего стола, такие как менеджер файлов (File Manager), менеджер стиля (Style Manager) и калькулятор (Calculator)
Desktop_Tools	Администрирование рабочего стола и инструментальные программные средства операционной системы, такие как перезагрузка приложения, редактор vi и проверка орфографии
Information	Иконки, представляющие часто используемые темы помощи
System_Admin	Инструментальные программные средства, используемые системным администратором
Digital_Media	Инструментальные программные средства для аудио, экранных образов и изображений.

3-11. СЛАЙД: Использование почтового сервиса

Использование почтового сервиса



Пояснения

Почтовый сервис в CDE позволяет посылать и принимать сообщения, а также управлять электронной почтой с рабочего стола. Иконка почты на передней панели изменяет свой вид, когда поступает новое сообщение, которое нужно прочитать. Иконка (Рисунок 7) указывает, что в системе нет непрочитанных сообщений, тогда как иконка (Рисунок 8) сообщает, что в системе есть непрочитанные сообщения.

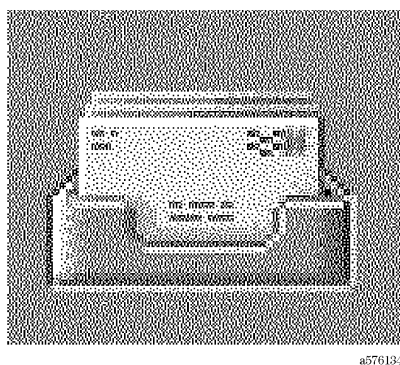


Рисунок 7

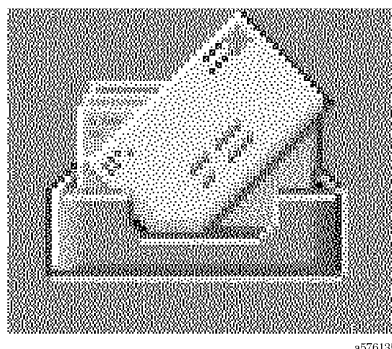


Рисунок 8

В главном окне почтового сервиса выводятся заголовки сообщений независимо от того, были они прочитаны или нет. Перед новым сообщением помещается буква **N**. То сообщение, которое высвечивается, является текущим, и его текст выводится в области **Message View**. Если отправитель включает в сообщение присоединение (attachment), например, календарь или графику, их иконка будет показана в списке присоединений (**Attachment list**).

Чтение сообщений

Чтобы прочитать сообщение, следует выбрать его из списка заголовков сообщений (**Message Header List**). Текст сообщения появляется в области **Message view**. Чтобы открыть для сообщения отдельное окно, нужно дважды щелкнуть по сообщению или выбрать элемент **Open** из меню **Message**.

Сортировка сообщений

По умолчанию сообщения выводятся в том порядке, в каком они поступали (дата/время). Можно переупорядочить их, выбирая соответствующий способ из меню **View**.

Поиск сообщений

Можно найти конкретное сообщение, руководствуясь содержимым полей **To:**, **From:**, **Subject:**, и **CC:**, если выбрать элемент **Find** из меню **Message** (Рисунок 9).

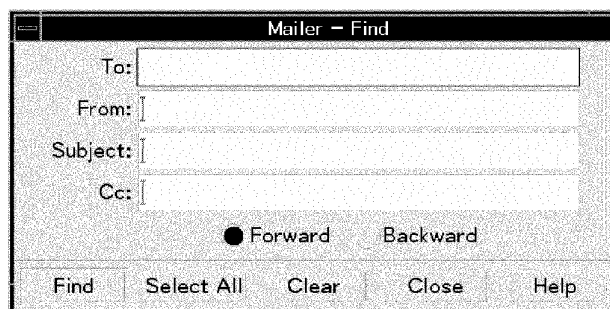


Рисунок 9

Можно сузить поиск, если указать критерии поиска в нескольких полях (например, в полях **From:** и **Subject:**). После ввода критериев поиска нужно щелкнуть по **Find**.

Все сообщения независимо от того, удовлетворяют они критерию поиска или нет, остаются в списке заголовков сообщений. Первое сообщение, удовлетворяющее критерию поиска, будет высвечено. Если еще раз щелкнуть по **Find**, будет высвечено следующее сообщение, удовлетворяющее критерию поиска, и так далее.

Выполнение действий над сообщениями

После того, как сообщение прочитано, с ним, возможно, нужно что-то сделать, например, ответить на него, запомнить в файле, удалить или послать кому-то еще с вашими комментариями.

Ответ на сообщение

1. Выберите сообщение, на которое нужно ответить.
2. Из меню **Compose** выберите одно из следующих:
 - a. **Reply to Sender** – ответить только отправителю сообщения.
 - b. **Reply to All** – ответить отправителю сообщения и всем другим получателям этого сообщения.
 - c. **Reply to Sender, Include** – ответить только отправителю данного сообщения, включив в ответ копию самого сообщения.
 - d. **Reply to All, Include** – ответить отправителю и всем другим получателям сообщения, включив в ответ копию сообщения.
3. Введите ответ.
4. Щелкните по **Send**.

Передача сообщения дальше

1. Выберите сообщение, которое нужно отправить дальше.
2. Из меню **Compose** выберите элемент **Forward Message**. Включается все сообщение вместе с присоединениями. Если нужно удалить присоединение, высветите его и выберите **Delete** из меню **Attachments**.
3. В поля **To:** и **CC:** введите почтовый адрес получателя данного сообщения.
4. При необходимости введите собственный комментарий.
5. Щелкните по **Send**.

Сохранение сообщения в файле

1. Выберите сообщение, которое нужно сохранить, высветив его.
2. Из меню **Message** выберите элемент **Save as Text**.
3. В диалоговом окне введите имя файла и каталога.

- Щелкните по **Save**.

Удаление/восстановление сообщения

Удаление сообщений

- Выберите сообщение, которое надо удалить.
- Из меню **Message** выберите элемент **Delete**.

Восстановление сообщений

Даже если сообщение было удалено, его можно восстановить, если только удаление не было безвозвратным.

- Чтобы восстановить последнее по времени удаленное сообщение, из меню **Message** выберите элемент **Undelete Last**.
- Чтобы восстановить сообщение, которое было удалено ранее, из меню **Message** выберите **Undelete from List**. Выберите одно или несколько сообщений, которые нужно восстановить. Щелкните по **Undelete**.

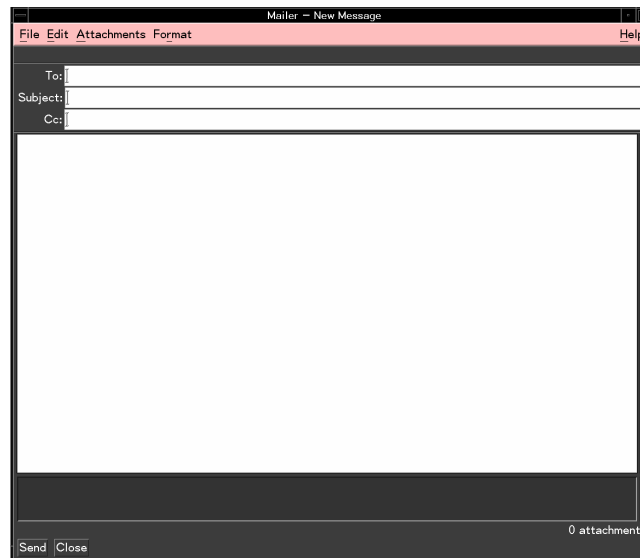
Уничтожение сообщений при закрытии почтового ящика

Можно уничтожить все удаленные сообщения при закрытии почтового ящика. Если сообщение уничтожено, его уже нельзя будет восстановить.

- Выберите элемент **Message Header List** из меню **Category** диалогового окна **Mail Options**.
- Выберите **When I close the mailbox** под **Destroy Deleted Messages**.
- Щелкните по **OK** или **Apply**, чтобы введенные изменения приобрели силу.

3–12. СЛАЙД: Отправление сообщений

Отправление сообщений



a57653

Пояснения

Для того чтобы отправить сообщение, необходимо иметь адрес email получателя сообщения. Формат адреса: *имя_пользователя@хост_имя*. Чтобы отправить сообщение, нужно:

1. Выбрать элемент **New Message** из меню **Compose** (показано на слайде).
2. Ввести в поле **To** адрес email получателя, в поле **Subject** – суть сообщения, и в поле **CC** – адреса email всех, кому надо послать копию данного сообщения.
3. После того как заполнены все поля адреса получателя, следует нажать **[Return]**, чтобы перейти в область текста и ввести текст сообщения. Для редактирования сообщения почтовый сервис использует такое же функциональное меню, как и текстовый редактор.
4. Щелкните по кнопке **[Send]**. Если сообщение нужно отослать позднее, его можно запомнить, выбрав элемент **Save as Text** из меню **Compose**.

В сообщение легко можно включить текстовый файл или создать сообщение на основе шаблона.

Создание шаблона

Можно создать шаблон сообщения, который содержит текст, часто используемый при формировании сообщений. Чтобы создать шаблон, нужно:

1. Набрать текст шаблона, используя текстовый редактор.

2. В почтовом сервисе щелкнуть по элементу **Mail Options** из меню **Mailbox**.
3. Щелкнуть по кнопке **Category** и выбрать **Templates**. Откроется диалоговое окно **Template**.
4. В поле **Menu Label** ввести имя шаблона.
5. В поле **File/Path**: ввести маршрутное имя файла, содержащего текст шаблона.
6. Щелкнуть по **Add**, чтобы добавить шаблон в список шаблонов.

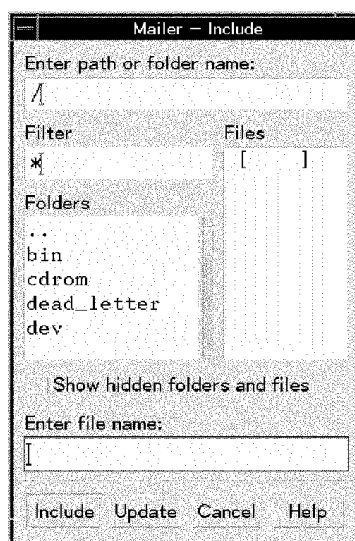
Использование шаблона

1. Выбрать элемент **New Message** из меню **Compose**.
2. Выбрать элемент **Templates** из меню **Format**.
3. Выбрать имя нужного шаблона из списка.

Включение файла в почтовое сообщение

Для того чтобы отправить кому-нибудь существующий файл, нужно:

1. Выбрать элемент **New Message** из меню **Compose**.
2. Выбрать элемент **Include** из меню **File** в окне **Compose** (Рисунок 10).
3. Передвигаясь по файловой системе, выбрать имя файла, который надо включить в сообщение.
4. Щелкнуть по **OK**.
5. Если нужно, добавить необходимый текст. Щелкнуть по **Send**, когда сообщение будет готово к отправке.



a576132

Рисунок 10

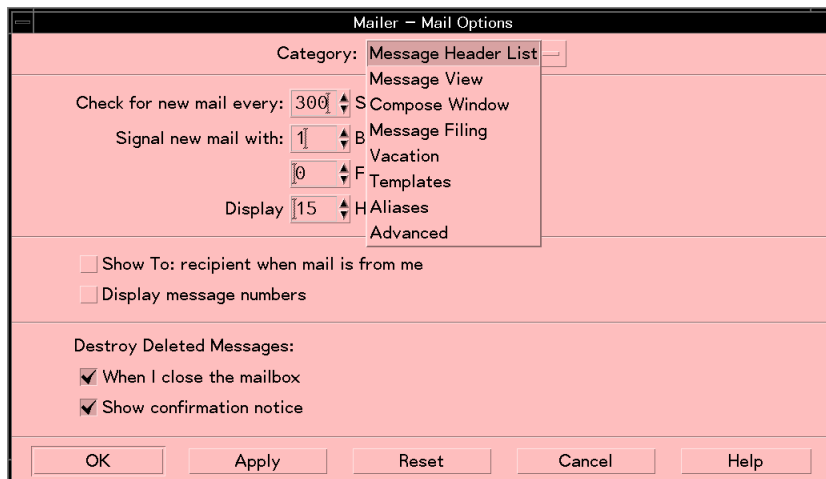
Если нужно включить в сообщение не текстовый файл, этот файл должен быть оформлен как присоединение (attachment). Для этого нужно:

1. Выбрать элемент **New Message** из меню **Compose**.
2. Выбрать элемент **New File** из меню **Attachments**.
3. Выбрать файл, который надо включить в сообщение.
4. Щелкнуть по .
5. Далее отправить сообщение как обычно.

Текст сообщения выводится в области **Message View**; присоединение можно увидеть в нижней части экрана с именем файла. Если дважды щелкнуть по иконке присоединения, откроется сеанс текстового редактора с этим включенным файлом.

3–13. СЛАЙД: Настройка почтового сервиса

Настройка почтового сервиса



Пояснения

Почтовый сервис может быть более точно настроен на основе установок по умолчанию. На слайде представлено диалоговое окно **Mail Options**, которое можно выбрать из меню почтового ящика (Mailbox). Настройка предполагает установку необходимых характеристик для всех или некоторых категорий почтового ящика:

Message Header List (список заголовков сообщений)

- Как часто проверять поступление новых сообщений.
- Нужно или не нужно сигнализировать о поступлении нового сообщения звуковым сигналом и/или мигающей иконкой.
- Сколько заголовков выводить при просмотре сообщений. Будет или не будет отправленное сообщение выводить имя получателя или имя отправителя.
- Выводить или не выводить количество сообщений.
- Уничтожать автоматически удаленные сообщения или выполнять удаление после выхода из почтового сервиса.

Message View (просмотр сообщения)

- Количество строк и символов в строке в области **Message View**.
- Добавлять (**Add**), удалять (**Delete**) или изменять (**Change**) заголовки файлов при выводе сообщений.

Compose Window (структура окна)

- Показывать или не показывать присоединения.
- Настроить отступ строки для ответа.
- Каталог, в котором сохраняется сообщение в случае, если произойдет сбой в системе в тот момент, когда сообщение составляется. Система автоматически сохраняет сообщения каждые 10 минут.
- Настраиваются доступные опции меню **Format** окна **New Message** для включения заказных файлов-заголовков.

Message Filing (хранение сообщений)

- Определяет, где запоминаются сообщения.
- Указывает, откуда начинать поиск сообщений.
- Сколько почтовых ящиков выводить (согласно тому, как давно они просматривались).
- Регистрировать ли копии посланных сообщений.

Vacation (каникулы)

- Позволяет пользователям посылать сообщение всем тем, кто прислал сообщение, а получатель не ответил на него в течение определенного заданного промежутка времени.
- Каникулярные сообщения имеют самый низкий приоритет по сравнению с другими сообщениями.

Templates (шаблоны)

- Используются для создания текста, часто включаемого в сообщения.

Aliases (псевдонимы)

- Позволяет пользователям создавать свой собственный личный список распределений с сокращенными именами. В один псевдоним можно включить много пользователей.

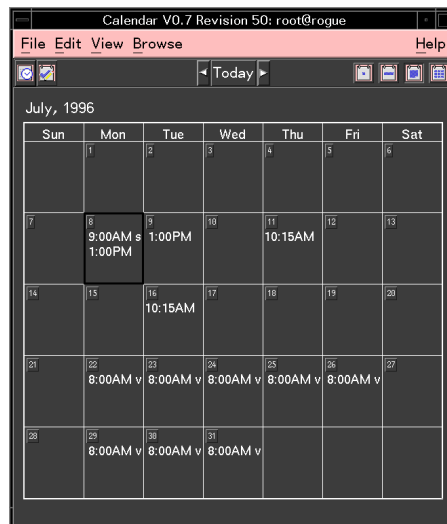
Advanced (дополнительные свойства)

- Управляет тем, как часто обновляется почтовый ящик.

- Показывать или нет подтверждающие примечания, когда делаются изменения (если они не запоминаются). Если эта возможность не выбрана, изменения вносятся автоматически.
- Использовать или нет кодировку MIME, которая необходима, если получатель выполняет пакет, полностью совместимый с MIME. Обычно эта возможность не выбирается, если только вы не уверены в такой ситуации.
- Сетевой файл распознавания блокировок, запрещающий двум экземплярам почтового сервиса открывать одно и то же сообщение.
- Включать или нет адрес отправителя в поле **Reply to All**, а если включать, то включать или нет имя хост-машины.

3–14. СЛАЙД: Использование календаря

Использование календаря



Пояснения

Приложение Календарь позволяет пользователям планировать встречи, создавать списки «Что сделать», делать напоминания, а также просматривать по сети другие календари и планировать групповые встречи, если предоставлены соответствующие права. Иконка календаря обычно расположена на передней панели, и щелкнув по ней, можно получить доступ к календарю.

По умолчанию выводится календарь текущего месяца (**Month view**), в котором выделена текущая дата. Это представление можно изменить, используя меню **View** или управляющие элементы в правой части инструментальной панели календаря (непосредственно под линейкой меню). Кроме **Month view**, можно увидеть:

- **Day view** – выводит встречи, назначенные на данный день, в почасовом режиме. Также предусмотрена возможность вывода мини календаря на три месяца, включающего текущий, предыдущий и следующий месяцы. Любой из дней может быть выведен, если щелкнуть по отдельному дню.
- **Week view** – выводит встречи, назначенные на данную неделю, в режиме по дням. Выводится почасовая сетка, в которой запланированное время представлено затемненной областью, а свободное время – не затемненной. По умолчанию выводится календарь на текущую неделю. Пользователь может вывести любую другую неделю, щелкнув по левой или правой стрелке около кнопки **Today**.

- **Year view** – выводит календарь на год. Так как охватывается очень большой промежуток времени, информация о запланированных встречах не выводится.

3–15. СЛАЙД: Планирование встреч

Планирование встреч

The screenshot shows the 'Calendar Appointment Editor' dialog box. It has a pink background and contains the following elements:

- Date:** 07/08/96
- Start:** 9:00 AM (with AM/PM radio buttons)
- End:** 10:00 AM (with AM/PM radio buttons)
- What:** A text area for describing the appointment.
- Frequency:** One Time (selected)
- Reminders:** Beep (checked), Flash (unchecked), Popup (checked), Mail (unchecked). Each has a time field (5 Mins for Beep and Popup).
- Privacy:** Others See Time And Text (selected)
- Mail To:** root@rogue
- Buttons:** Insert, Change, Delete, Clear, Cancel, Help.

Пояснения

С помощью меню можно включить назначение встреч на тот день, который выводится в данный момент (потому что нельзя сделать, если выводится календарь на год). Чтобы запланировать встречу, надо:

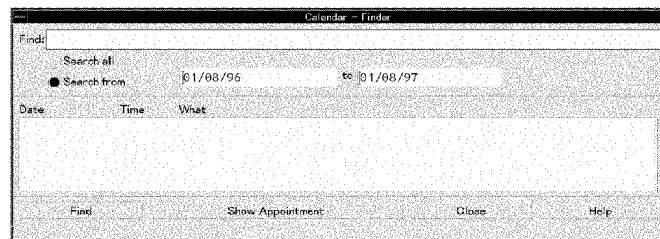
1. Вывести на экран тот день, когда планируется провести встречу.
2. Выбрать **Appointment...** из меню **Edit**, чтобы запустить редактор встреч календаря (**Calendar Appointment Editor**).
3. Выбрать поля **Start** и **End** для указания времени начала и окончания встречи.
4. Указать, что это за встреча (эта информация в укороченном виде будет выведена в области календаря).
5. Если нужны дополнительные опции, нужно щелкнуть по **More**, чтобы вывести доступные варианты, например: как часто будет проходить эта встреча (пусть эта встреча должна проходить в начале каждого месяца); нужно или нет напомнить об этой встрече звуковым сигналом, миганием или почтовым сообщением; в течение какого времени вы хотите сохранить это сообщение.

Изменение или удаление назначенных встреч

1. Вывести день, когда нужно изменить назначенную встречу.
2. Активизировать редактор назначений.
3. Высветить назначение, которое нужно изменить или удалить.
4. Если нужно изменить назначение, следует щелкнуть на области, которую нужно изменить (т.е. на времени начала/окончания).
5. Щелкните либо по **Change** (эта кнопка появится, только если были сделаны изменения), либо по **Delete**.

Поиск назначенных встреч

Предположим, вам известно, что назначена некоторая встреча, но вы не знаете, когда она назначена. Вместо того, чтобы просматривать календарь на каждый месяц, можно использовать элемент **Find** из меню **View** (Рисунок 11).



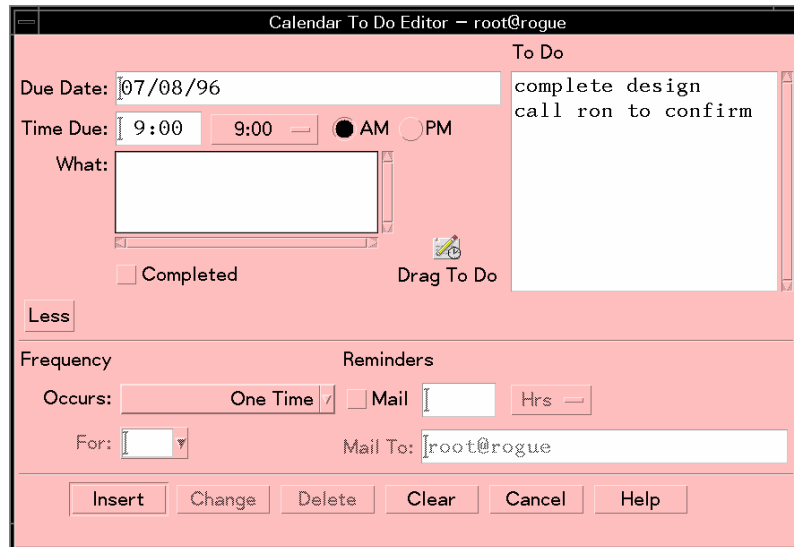
a576133

Рисунок 11

1. Введите ключевое слово в поле **Find**.
2. Если хотите, измените диапазон просматриваемых дат. По умолчанию будет просматриваться прошедший и следующий шестимесячный период.
3. Щелкните по **Find**.
4. Щелкните по требуемому назначению из полученного списка.
5. Щелкните по **Show Appointment**, чтобы вывести назначения.
6. Щелкните по **Close**.

3-16. СЛАЙД: Элементы To Do (Сделать)

Элементы To Do (сделать)



Пояснения

Календарь дает возможность планировать элементы To Do (Сделать); в дальнейшем запланированные элементы (дела) можно пометить как законченные или продолжающиеся. Эти элементы выводятся в хронологическом порядке, и для каждого элемента указываются дата и время его выполнения, а также его описание. Чтобы активизировать соответствующий редактор (**Calendar To Do Editor**), нужно выбрать элемент **To Do ...** из меню **Edit** или щелкнуть по иконке **To Do** инструментальной панели календаря (с меткой проверки и карандашом).

1. По умолчанию, обусловленной датой будет та, которая выделена на календаре, если требуется редактировать обусловленную дату.
2. В поле **What** ввести описание элемента **To Do**.
3. Щелкнуть по **Insert**.
4. Щелкнуть по **Cancel**, чтобы закрыть редактор **To Do**.

Пометка элемента как выполненного

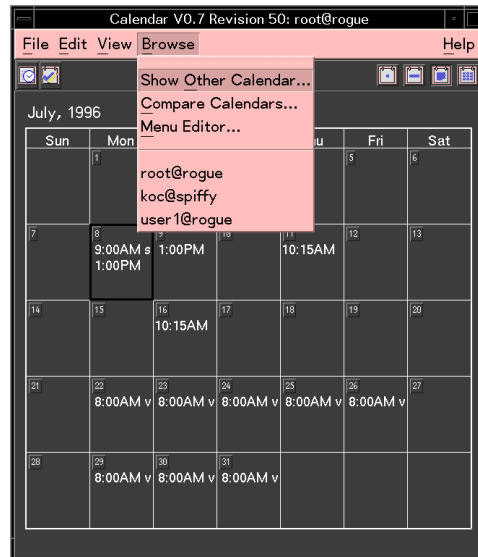
1. Выберите элемент **To Do List...** из меню **View**.
2. В диалоговом окне будут выведены все элементы **To Do**, **Completed To Do** или **Pending To Do**. Выберите то, что вы хотите вывести.
3. Щелкните в поле для вставки проверки, которая отмечает выполнение элемента **To Do**. Чтобы удалить пометку, повторно щелкните в этом поле.
4. Щелкните по .

Другой способ

1. Выберите элемент **To Do ...** из меню **Edit**, чтобы вывести элементы **To Do** заданного дня.
2. Щелкните по , чтобы пометить элемент как выполненный (так можно и включить, и выключить пометку).
3. Щелкните по .
4. Щелкните по , чтобы выйти из редактора.

3-17. СЛАЙД: Просмотр календаря по сети

Просмотр календаря по сети



Пояснения

Приложение рабочего стола Календарь предоставляет возможность просматривать по сети другие календари, если вы знаете их имена (в формате *имя_календаря@хост-имя*) и имеете полномочия для доступа к ним. Это дает пользователям возможность просматривать группу календарей, чтобы найти, например, свободное время для назначения встречи.

Прежде чем просматривать календарь, его нужно включить в список просмотра (**Browse List**). Для этого нужно:

1. Выбрать элемент **Menu Editor** из меню **Browse**.
2. В поле **User Name** ввести *имя_календаря@хост-имя*.
3. Щелкнуть по **Add Name**.
4. Щелкнуть по **OK**.

После того как имя календаря добавлено в список, этот календарь можно просматривать по сети.

1. Выберите элемент **Choose Calendars** из меню **Browse**.
2. Выберите имя календаря (или нескольких календарей), который нужно просмотреть.

3. Календари добавляются сверху, один на другой. Занятые времена затеняются, свободные – нет.

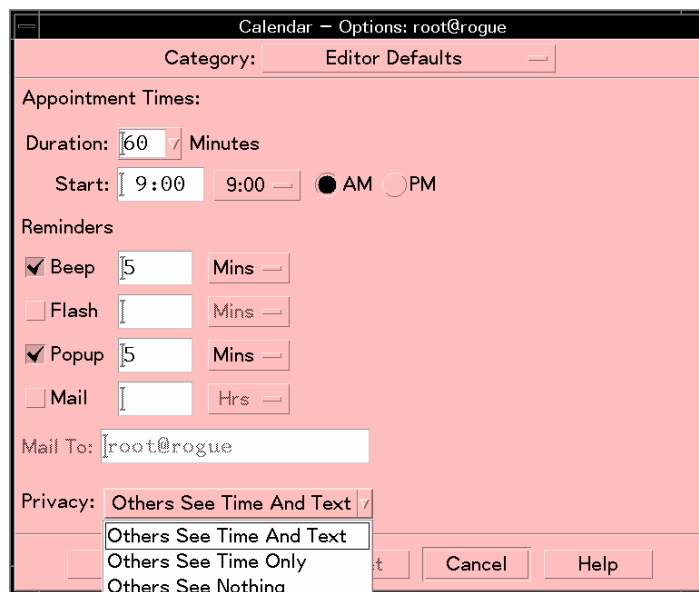
Планирование встречи в других календарях

Если вы находитесь в окне **Compare Calendar**, вы можете планировать встречи, если у вас для этого есть соответствующие полномочия.

1. Выберите один или несколько элементов из **Browse list**.
2. Щелкните по незатененной области, доступной всем элементам.
3. Щелкните по Schedule. Будет выведен редактор **Calendar Group Appointment Editor**, который предоставит список доступных календарей **Calendar Access list**. Буква **Y** в колонке **Access** означает, что вы имеете право доступа к календарю и можете обновлять соответствующее расписание. Буква **N** означает, что вы не имеете такого права. Владелец календаря должен предоставить вам право доступа к календарю для назначения в нем встреч.

3–18. СЛАЙД: Предоставление прав доступа к календарю

Предоставление прав доступа к календарю



Пояснения

По умолчанию календарь настраивается так, чтобы предоставить всем возможность его просмотра. Вставлять и удалять назначения встреч может только владелец календаря. Владелец может изменить эти права следующим образом:

1. Выберите элемент **Options...** из меню **File**.
2. Из меню **Category** выберите **Access List**, чтобы вывести диалоговое окно со списком и правами доступа, как показано на слайде.
3. В поле **User Name** введите *имя_календаря@хост_имя* для пользователя, кому вы хотите предоставить возможность доступа.
4. Выберите для прав доступа **View** (просмотреть), **Insert** (вставить) и/или **Change** (изменить):

Public Разрешает другим календарям выводить время и текст назначенных встреч, отмеченных как **Others See Time and Text** (Другие могут видеть время и текст).

Semiprivate Разрешает другим календарям выводить время и текст назначений, отмеченных как **Others See Time Only** (Другие могут видеть только время).

Private Не разрешает другим календарям выводить время и текст назначений, отмеченных как Others See Nothing (Другие не могут видеть ничего).

5. Щелкните по , чтобы добавить календарь в список доступа с теми правами, которые вы выбрали.
6. Щелкните по или .

3–19. Упражнения: Использование CDE

Цель: Освоить использование основных элементов среды CDE рабочего стола.

Система Linux:



Если вы используете оконный KDE системы Linux, попробуйте найти приложения или опции, эквивалентные тем, которые были описаны для среды CDE.

Система HP-UX:

Указания

1. Используя менеджер файлов, перейдите в папку **class**. Выберите файл **cde_intro** и скопируйте его в файл **cde_intro2**.
2. Переместите файл **cde_intro2** в папку **cde_dir**.
3. Перейдите в папку **cde_dir**. Измените права доступа к файлу **cde_intro2**, так чтобы его можно было только читать.
4. Вернитесь во входную папку. Используйте менеджер файлов, для того чтобы найти все файлы, которые содержат строку *graphical environment*. Выполняйте поиск в папке **class**.

5. С помощью менеджера файлов найдите все файлы, имена которых начинаются символами *data*.
6. Можно выполнить поиск по шаблону, если использовать вопросительный знак (?) для определения одного (любого) символа. С помощью менеджера файлов найдите все файлы, имена которых начинаются символами *data*, за которыми следует один (любой) символ.
7. С помощью менеджера файлов из папки **cde_dir** удалите файл **cde_intro2**.
8. Восстановите файл **cde_intro2** из корзины.
9. С помощью менеджера файлов удалите безвозвратно из папки **cde_dir** файл **cde_intro2**.

10. Используя текстовый редактор, откройте для редактирования файл `$HOME/class/cde_intro`. Скопируйте первый абзац текста и вставьте его в конец документа.
11. Замените все вхождения символов CDE, кроме первого и третьего, текстом *Common Desktop Environment*.
12. Исправьте в файле все слова с опечатками.
13. Используя всплывающее меню переключателя рабочих областей, добавьте еще одну область и назовите ее **CDE Class**.
14. Используя меню дополнительных панелей и всплывающие меню, измените иконку для личных приложений с текстового редактора на терминал.

15. Используйте **Man Page Viewer** из менеджера приложений, чтобы получить статью *Справочника HP-UX* для команды **ls**.

16. Выберите вашего коллегу, которому можно будет отправить почтовое сообщение. Если вы работаете в разных системах, вы должны знать входное имя вашего коллеги и хост имя системы, в которой он работает. Эта информация необходима для задания адреса email вашего коллеги в формате *имя_пользователя@хост_имя*.

Пошлите вашему коллеге сообщение и попросите его прислать сообщение вам. (Если вы испытываете какие-либо затруднения в поиске коллеги, вы можете послать сообщение самому себе.)

17. Когда ваш коллега пришлет вам сообщение, ответьте на него и отошлите исходное сообщение третьему коллеге.

18. Используя текстовый редактор, создайте шаблон, который вы будете использовать для ежемесячной отправки вашему руководителю отчета о положении дел. Сохраните этот шаблон в файле **status**. В почтовом сервисе создайте шаблон с именем **monthly**, содержащий этот вновь созданный файл. Используйте этот шаблон для отправления отчета о положении дел вашему коллеге.

19. Используйте функции календаря для создания пяти или шести назначений встреч в текущем месяце. Вы можете назначить единичную встречу или регулярные встречи, например, ежемесячные.

20. Установите настройки календаря так, чтобы вы могли по сети видеть ваш календарь и календарь вашего коллеги, добавив оба эти календаря в список просмотра (**Browse List**).

21. Вы хотите запланировать важную встречу с вашим коллегой, но сначала хотите проверить его календарь, возможна ли эта встреча. Просмотрите календари таким образом, чтобы можно было видеть одновременно и свой календарь, и календарь вашего коллеги.

22. Предоставьте вашему коллеге права доступа **Insert** и **Change** к вашему календарю, так чтобы он мог планировать встречи с вами, когда это необходимо.

23. Запланируйте встречу с вашим коллегой и отправьте ему по почте напоминание о ней.

Модуль 4 — Навигация по файловой системе

Цели:

Изучив этот модуль, вы сможете:

- Описывать структуру файловой системы UNIX
- Понимать, чем отличаются друг от друга файлы и каталоги
- Перемещаться по файловой системе UNIX
- Создавать и удалять каталоги
- Представлять различие между абсолютным и относительным маршрутным именем
- Использовать (когда это возможно) относительные маршрутные имена для уменьшения объема ввода

4-1. СЛАЙД: Что такое файловая система?

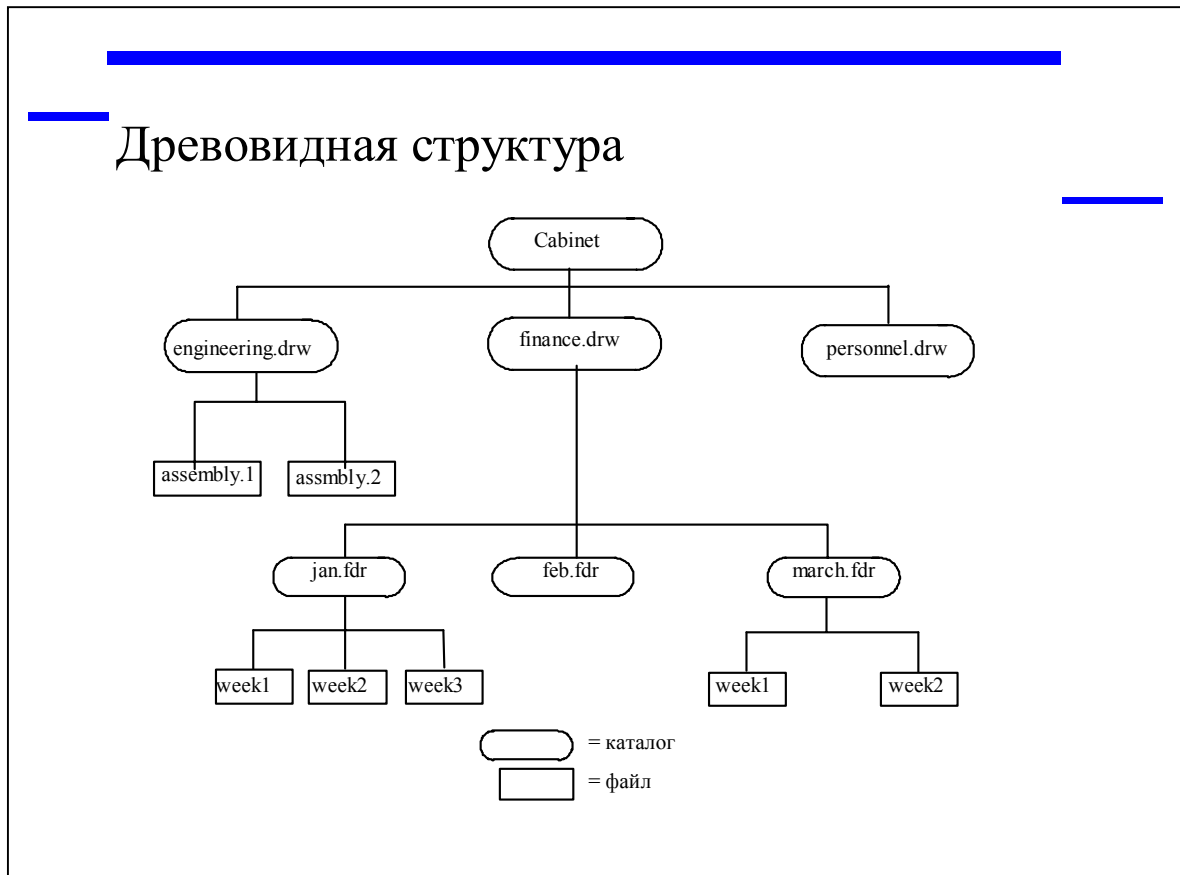


Пояснения

В системе UNIX для организации и управления файлами и каталогами предусмотрена **файловая система**. Обычно **файл** представляет собой хранилище данных, тогда как **каталог** является хранилищем файлов и других каталогов. Каталог, находящийся в другом каталоге, часто называют **подкаталогом**.

Файловая система ОС UNIX очень похожа на картотеку. Всю файловую систему можно рассматривать как картотеку с множеством выдвижных ящиков, папок и отчетных материалов. Выдвижной ящик можно рассматривать как подкаталог, в котором находятся отчеты или папки с отчетами. Папку с отчетами также можно рассматривать как подкаталог, в котором находятся отчеты. Отчет представляет собой файл, и в нем хранятся реальные данные.

4–2. СЛАЙД: Древоподобная структура



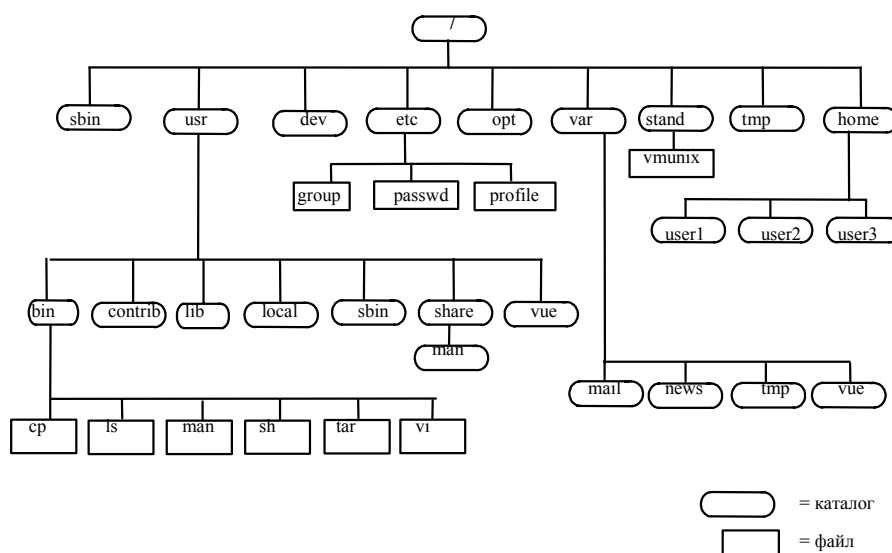
Пояснения

Организация каталога может быть представлена графически в виде иерархической **древоподобной структуры**. Каждый элемент дерева представляет собой каталог или файл. Каталоги на рисунке изображены овалами, а файлы – прямоугольниками, так что на диаграмме их можно легко различить.

Данный слайд графически представляет в виде дерева структуру картотеки, приведенной на первом слайде.

4-3. СЛАЙД: Иерархия файловой системы

Иерархия файловой системы



Пояснения

Подобно картотеке, файловая система ОС UNIX обеспечивает простой и эффективный механизм организации файлов. Так как дистрибутив системы UNIX обычно содержит сотни файлов и программ, было принято решение об использовании иерархической структуры, так что каждая версия ОС UNIX поддерживает единообразную организацию каталогов. Вершина иерархии определяется как **корневой (root)** каталог (потому что она представляет верхушку перевернутого дерева) и обозначается символом "слеш" (/).

В системе UNIX есть команды, которые в соответствии с вашими потребностями позволяют легко создавать новые каталоги, а также команды, которые перемещают или копируют файлы из одного каталога в другой. Это так же легко, как добавить новую папку в один из выдвижных ящиков картотеки или переложить отчет из старой папки в новую.

В версии HP-UX 10.0 файловая система разделена на две основные части: в одной части находятся статические файлы, а в другой – динамические.

Статические файлы (Разделяемые) В этой части находятся три важных каталога: **/opt**, **/usr** и **/sbin**.

/opt

Содержит приложения и программные продукты. Разработчики и администраторы системы HP-UX используют этот каталог для инсталляции новых продуктов или локальных приложений.

/usr/bin	Содержит программы для всех команд, которые включены в раздел 1 <i>Справочника</i> и которые необходимы для выполнения основных операций в системе UNIX и для работы с файлами. Эти команды обычно доступны всем пользователям (bin – это сокращение слова binary – двоичный).
/usr/sbin	Содержит программы для всех команд, которые включены в раздел 1m <i>Справочника</i> . Это команды администратора системы. Многие из этих команд может использовать только суперпользователь (superuser). Это оговаривается в разделе 1m <i>Справочника</i> .
/usr/lib	Содержит архивные и разделяемые библиотеки, используемые в приложениях.
/usr/share	Содержит файлы, независимые от поставщика системы (наиболее важный из них – <i>Справочник</i>).
/usr/share/man	Содержит все файлы, связанные со статьями оперативного <i>Справочника</i> .
/usr/local/bin	Обычно содержит программы и утилиты, разработанные для локального применения.
/usr/contrib/bin	Обычно содержит общедоступные программы и утилиты. Их список можно найти, обратившись к сервисному информационному бюллетеню или группе пользователей.
/sbin	Содержит специальные команды, используемые для запуска и остановки системы.

Динамические файлы (Личные) В этой части находятся семь важных каталогов: **/home**, **/etc**, **/stand**, **/tmp**, **/dev**, **/mnt** и **/var**.

/home Обычно содержит по одному каталогу для каждого зарегистрированного пользователя системы. Каждый пользователь системы UNIX должен иметь учетную запись. Наряду с входным идентификатором и паролем, системный администратор предоставляет каждому пользователю его собственный каталог. Пользователь может управлять содержимым своего каталога. Пользователь отвечает за организацию своей деятельности и управление ею, создавая подкаталоги и файлы в каталоге, ассоциированном с его учетной записью. Когда пользователь входит в систему, он сразу же оказывается в каталоге, ассоциированном с его учетной записью. Поэтому этот каталог часто называют *HOME* каталогом или *входным (login)* каталогом. Из этого каталога пользователь может перейти в любой другой каталог по иерархии файловой системы, если у него есть соответствующие права доступа. Как минимум, у него будет возможность доступа ко всему, что находится в его *входном* каталоге, а как максимум, он может перейти в *любой* каталог по иерархии файловой системы (по умолчанию). Прерогативой администратора системы является ограничение доступа пользователей к особым каталогам системы.

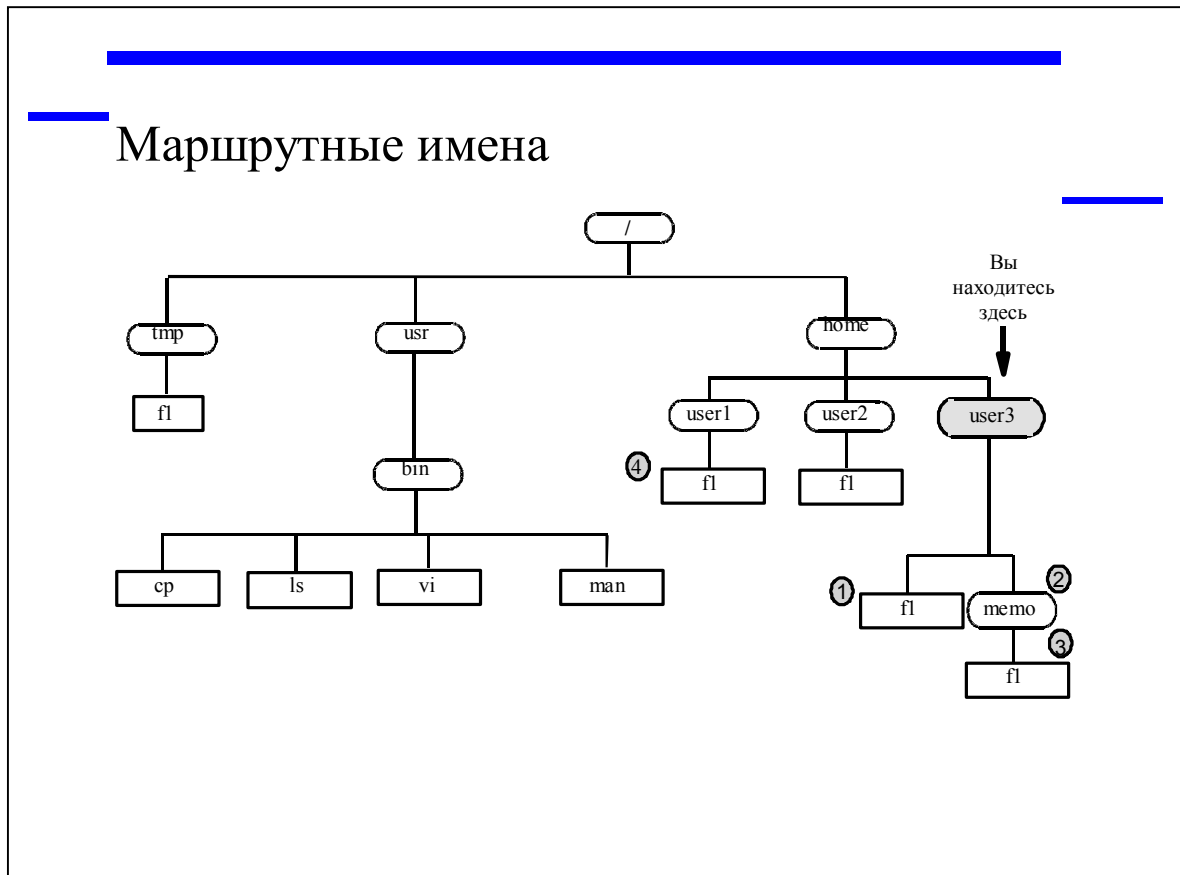
<code>/etc</code>	Содержит большое количество системных конфигурационных файлов. Эти файлы описаны в разделе 4 <i>Справочника</i> .
<code>/stand/vmunix</code>	Хранит программу, представляющую собой ядро системы UNIX. Эта программа загружается в память при загрузке системы и управляет всей деятельностью системы.
<code>/tmp</code>	Обычно используется как рабочее пространство для исполняющихся систем, которым нужно создавать промежуточные или рабочие файлы. Примечание: соглашения, принятые в системе UNIX, гласят, что файлы, находящиеся в каталоге с именем tmp , могут быть удалены в <i>любое время</i> .
<code>/dev</code>	Содержит файлы, представляющие аппаратные устройства, подсоединенные к системе. Так как эти файлы играют роль шлюзов по отношению к устройствам, данные никогда непосредственно не запоминаются в этих файлах устройств. Часто эти файлы называют специальными файлами или файлами устройств .
<code>/mnt</code>	Используется для монтирования других устройств (например, лазерных дисков).
<code>/var/mail</code>	Содержит "почтовые ящики" для каждого пользователя, имеющего поступившие по электронной почте сообщения.
<code>/var/news</code>	Содержит все файлы, содержащие сообщения о текущих новостях системы. Все содержимое этого каталога можно увидеть, выполнив команду news -a .
<code>/var/tmp</code>	Обычно используется как временное рабочее пространство для пользователей.

Система Linux



Хотя иерархия каталогов системы Linux следует тем же соглашениям, что и система HP-UX, между ними все же есть некоторое различие. Тем не менее, имена большинства каталогов, определяющих иерархию каталогов в системе Linux, такие же, как и в HP-UX.

4-4. СЛАЙД: Маршрутные имена



Пояснения

Абсолютное

① **/home/user3/fl**

② **/home/user3/memo**

③ **/home/user3/memo/fl**

Относительное по отношению к /home/user3

① **fl**

② **memo**

③ **memo/fl**

Относительное по отношению к /home/user1

④ **/home/user1/fl**

④ **fl**

Многие команды системы UNIX оперируют с файлами и/или каталогами. Чтобы сообщить команде, где находится нужный файл или каталог, в качестве аргумента команды используется маршрутное имя. Маршрутное имя задает путь по иерархии файловой системы, который нужно пройти для достижения требуемого файла или каталога.

\$ команда [опции] [маршрутное_имя маршрутное_имя ...]

Чтобы продемонстрировать концепцию маршрутных имен, здесь используется аналогия с прослеживанием ветвей дерева файловой системы UNIX с помощью карандаша, чтобы пройти от одного положения в другое. Маршрутное имя представляет собой список всех

каталогов, к которым прикасается кончик карандаша при прослеживании пути по иерархии, включая и требуемый файл или каталог.

Когда определяется маршрутное имя файла или каталога, для разграничения имен файлов и/или каталогов используется символ "прямой слеш" (/).

каталог/каталог/каталог

каталог/файл

Каждый раз, когда пользователь входит в систему, он попадает в некоторый каталог в иерархии файловой системы. Можно изменить свое положение, перейдя в какой-то другой каталог с помощью соответствующих команд системы UNIX, но всегда пользователь находится в каком-нибудь каталоге. Например, когда пользователь входит в систему, он попадает в свой входной (*HOME*) каталог.

Положение файла или каталога может быть определено с помощью или абсолютного, или относительного маршрутного имени.

Абсолютное маршрутное имя

- Однозначно определяет положение файла или каталога
- Всегда начинается с вершины иерархии (корня)
- Всегда начинается символом /
- Не зависит от текущего положения пользователя в иерархии файловой системы
- Всегда уникально в пределах всей файловой системы

Примеры абсолютного маршрутного имени

Следующие маршрутные имена определяют положение всех файлов с именем **f1** в иерархии файловой системы, представленной на слайде. Обратите внимание на то, что в приведенном примере файловой системы существует много файлов с именем **f1**, но каждый из них имеет уникальное абсолютное маршрутное имя.

/tmp/f1

/home/user1/f1

/home/user2/f1

/home/user3/f1

/home/user3/memo/f1

Относительное маршрутное имя

- Всегда начинается с текущего положения пользователя в иерархии файловой системы
- Никогда не начинается символом /
- Уникально только по отношению к текущему положению пользователя
- Часто короче, чем абсолютное маршрутное имя

Примеры относительного маршрутного имени

В следующих примерах также представлена ссылка на файл с именем **f1**, но определение относительного маршрутного имени зависит от текущего положения пользователя в иерархии файловой системы.

Пусть текущий каталог – **/home** :

```
user1/f1  
  
user2/f1  
  
user3/f1  
  
user3/memo/f1
```

Пусть теперь текущим каталогом является **/home/user3**:

```
f1  
  
memo/f1
```

Если же, наконец, текущим каталогом является **/home/user3/memo** :

```
f1
```

Следует обратить внимание на то, что относительное маршрутное имя файла **f1** не является уникальным. Тем не менее, система UNIX знает, какой именно файл нужно найти. Она знает, что нужно найти файл **/home/user1/f1**, если пользователь в данный момент находится в каталоге **/home/user1**, или нужно найти файл **/home/user3/memo/f1**, если пользователь находится в каталоге **/home/user3/memo**. Также следует заметить, что относительное маршрутное имя может быть существенно короче определения абсолютного маршрутного имени. Например, если пользователь находится в каталоге **/home/user3/memo**, ему достаточно в команде напечатать просто **f1**:

Абсолютное маршрутное имя **lp /home/user3/memo/f1**

Относительное маршрутное имя **lp f1**

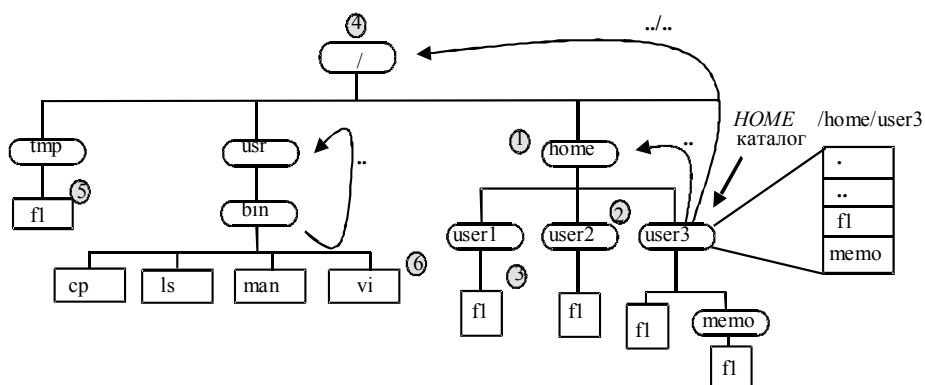
Во втором случае ввод относительного имени существенно короче.

ПРИМЕЧАНИЕ: Когда осуществляется доступ к файлу с помощью относительного маршрутного имени, важно знать, в каком каталоге в данный момент времени находится пользователь, чтобы гарантировать, что осуществляется доступ к нужному файлу, особенно если файлы с одинаковыми именами существуют в более чем одном каталоге системы.

С точки зрения внутренней реализации система UNIX находит все файлы и каталоги, используя абсолютное маршрутное имя. Это разумно, так как абсолютное маршрутное имя абсолютно и уникально идентифицирует файл или каталог (так как существует только один корень). Система UNIX позволяет использовать относительные маршрутные имена только для удобства пользователей.

4–5. СЛАЙД: Некоторые специальные каталоги

Некоторые специальные каталоги



Пояснения

Абсолютное

- ① /home
- ② /home/user2
- ③ /home/user1/fl
- ④ /
- ⑤ /tmp/fl
- ⑥ /usr/bin/vi

Относительное по отношению к /home/user3

- ① ..
- ② ../user2
- ③ ../user1/fl
- ④ ../../
- ⑤ ../../tmp/fl
- ⑥ ../../usr/bin/vi

Когда создается каталог, в нем автоматически создаются два элемента – с именами "точка" (.) и "точка-точка" (..). Эти имена обычно используются при создании относительных маршрутных имен. Можно заметить, что в примерах, приведенных на предыдущем слайде, относительные маршрутные имена позволяют перемещаться только вниз по иерархии файловой системы. С помощью элемента с именем .. можно перемещаться по иерархии файловой системы вверх.

Входной каталог

Когда в системе регистрируется новый пользователь, ему назначаются входное имя и, возможно, пароль. Также будет создан каталог, владеть и управлять которым будет данный пользователь. Обычно этот каталог размещается в каталоге **/home** и имеет такое же имя, как и входное имя пользователя. В дальнейшем в данном каталоге пользователь сможет создавать собственные файлы и подкаталоги.

Когда пользователь входит в систему, система UNIX помещает его в этот каталог. Поэтому этот каталог и называется входным каталогом, или *HOME* каталогом.

Точка (.)

Элемент с именем "**точка**" представляет текущий каталог, в котором в данный момент находится пользователь.

Примеры с (.)

Если текущим каталогом является **/home/user3**:

- .** определяет текущий каталог **/home/user3**
- ./f1** определяет **/home/user3/f1**
- ./memo/f1** определяет **/home/user3/memo/f1**

Точка-точка (..)

Элемент с именем "**точка-точка**" представляет каталог, расположенный непосредственно над текущим каталогом, и часто называется **родительским каталогом**. В каждом каталоге могут находиться несколько файлов и подкаталогов, но каждый каталог имеет только один родительский каталог. Поэтому не возникает никаких конфликтов при перемещении вверх по иерархии файловой системы.

Корневой каталог (**/**) аналогичен любому другому каталогу и также содержит элементы с именами "**точка**" и "**точка-точка**". Но так как корневой каталог не имеет родительского каталога, элемент с именем "точка-точка" определяет опять же сам корневой каталог.

Примеры с (..)

Если текущим каталогом является **/home**:

- ..** определяет **/**
- ../..** также определяет **/**
- ../tmp** определяет **/tmp**
- ../tmp/f1** определяет **/tmp/f1**

Если текущим каталогом является **/home/user3**:

- ..** определяет **/home**
- ../..** определяет **/**

`../user2` определяет `/home/user2`

`../user1/f1` определяет `/home/user1/f1`

`../../tmp/f1` определяет `/tmp/f1`

Следует обратить внимание на то, что в последнем примере абсолютное маршрутное имя короче относительного. Если относительное маршрутное имя проходит через корневой каталог, лучше вместо относительного маршрутного имени использовать абсолютное.

4–6. СЛАЙД: Основные команды файловой системы

Основные команды файловой системы

pwd	Вывести имя текущего каталога
ls	Вывести список файлов и каталогов, находящихся в каталоге
cd	Перейти в другой каталог
find	Найти файлы
mkdir	Создать каталог
rmdir	Удалить каталог

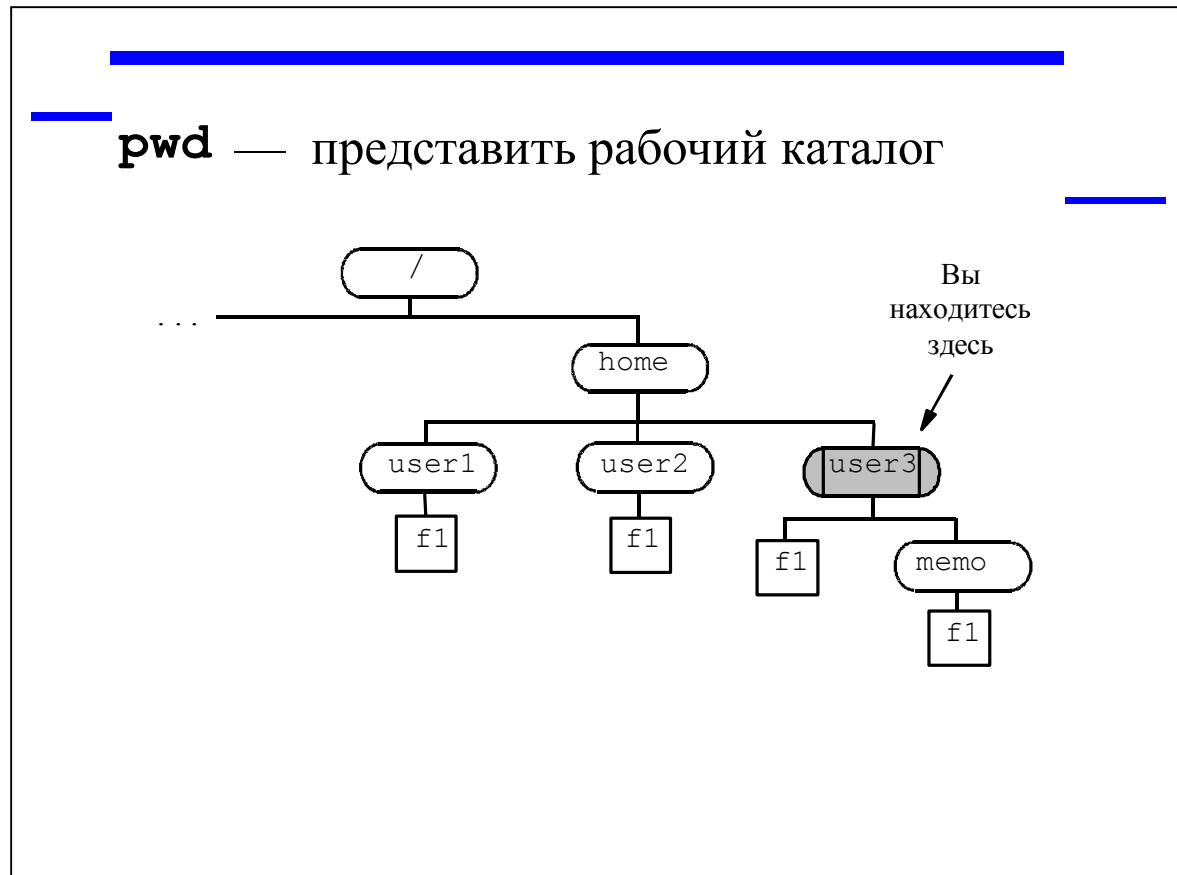
Пояснения

Каталог, по сути, является средством организации файлов. В оставшейся части данного модуля будут рассмотрены основные команды манипулирования каталогами. Будет показано, как:

- выводить имя текущего каталога,
- увидеть, какие файлы и каталоги находятся в текущем каталоге,
- изменить положение в иерархии файловой системы, перейдя в другой каталог,
- создать каталог,
- удалить каталог.

В данном модуле не затрагиваются файлы, находящиеся в каталоге. Здесь обсуждаются только каталоги.

4–7. СЛАЙД: `pwd` – представить рабочий каталог



Пояснения

Каждый раз при входе в систему UNIX пользователь оказывается в некотором каталоге иерархии файловой системы. Каталог, в котором находится пользователь, часто называют рабочим каталогом.

Команда **`pwd`** (**p**resent **w**orking **d**irectory) выводит абсолютное маршрутное имя каталога, в котором находится пользователь в данный момент.

Так как система UNIX позволяет очень легко перемещаться по файловой системе, все пользователи используют эту команду, чтобы проверить свое текущее положение в иерархии файловой системы. Новым пользователям полезно часто использовать эту команду, чтобы вывести на экран свое текущее положение в файловой системе при перемещении по ней.

4–8. СЛАЙД: `ls` – вывести содержимое каталога

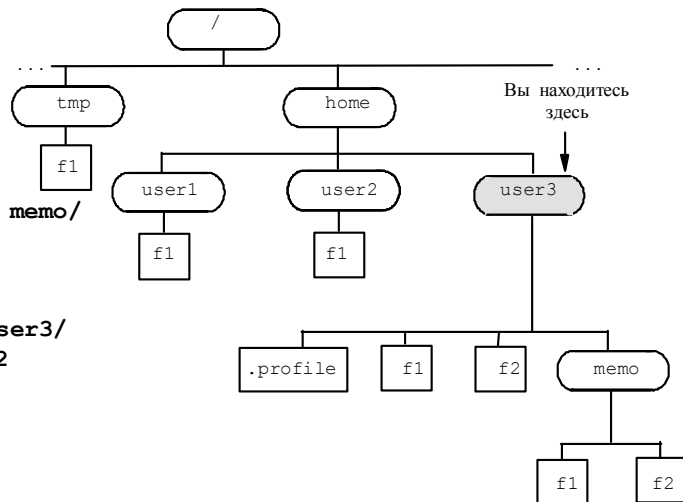
`ls` — вывести содержимое каталога

Синтаксис:

```
ls [-adlFR] [маршрутное_имя]
```

Примеры:

```
$ ls
f1 f2 memo
$ ls -F
f1 f2* memo/
$ ls -aF
.profile f1 f2* memo/
$ ls memo
f1 f2
$ ls -F /home
user1/ user2/ user3/
$ ls -F ../user2
f1
```



Пояснения

Команда `ls` используется для вывода имен файлов и каталогов.

Команда `ls` без аргументов выводит имена файлов и каталогов, размещенных в текущем каталоге.

В команде `ls` допускается задание аргументов, представляющих абсолютное или относительное маршрутное имя файла или каталога. Если задано имя файла, `ls` выводит информацию, ассоциированную с этим файлом. Если задано имя каталога, `ls` выводит содержимое этого каталога.

Команда `ls` допускает также задание многих опций, которые обеспечивают вывод дополнительной информации. В одной командной строке могут быть заданы несколько опций, чтобы вывести более полную информацию о файле или каталоге. Некоторые из наиболее часто используемых опций представлены на слайде. К ним относятся:

- a** Выводит все файлы, включая и те, имена которых начинаются символом "точка" (.). Обычно такие **файлы с точкой скрыты**, т.е. не выводятся при выполнении команды, если только не указана опция **-a**. Файлы с точкой обычно содержат конфигурационную информацию для сеанса пользователя или для приложений.

- d** Выводит характеристики самого каталога, а не его содержимое. Часто используется вместе с опцией **-l**, чтобы вывести подробную информацию о каталоге.
- l** Предоставляет полный перечень атрибутов для каждого файла, включающий его тип, права доступа, количество ссылок на файл, владельца, группу, размер (в байтах), время последней модификации и название файла.
- F** Добавляет символ "слеш" (/) после имени каждого каталога и символ "звездочка" (*) после имени каждого исполняемого файла.
- R** Рекурсивно выводит список файлов из заданного каталога и всех его подкаталогов.

Примеры

```
$ pwd  
/home/user3
```

```
$ ls -F /home  
user1/ user2/ user3/
```

Аргумент представлен абсолютным именем

```
$ ls -F ..  
user1/ user2/ user3/
```

Аргумент представлен относительным именем

```
$ ls -F ../user1  
f1
```

Аргумент представлен относительным именем

```
$ ls -l memo  
-rw-rw-rw- 1 user3 class 27 Jan 24 06:11 f1  
-rw-rw-rw- 1 user3 class 37 Jan 23 19:03 f2
```

Аргументом является относительное имя каталога

```
$ ls -ld memo  
drwxr-xr-x 2 user3 class 1024 Jan 20 10:23 memo
```

Вывод информации о каталоге мемо

```
$ ls -l f1 f2  
-rw-rw-rw- 1 user3 class 27 Jan 24 06:11 f1  
-rw-rw-rw- 1 user3 class 37 Jan 23 19:03 f2
```

Несколько аргументов, задающих относительные имена файлов

```
$ ls -R  
memo f1 f2  
./memo:  
f1 f2
```

Рекурсивный вывод содержимого подкаталогов

```
$ ls user2  
user2 not found
```

в текущем каталоге нет файла или каталога с именем user2

Сокращенные имена команд в HP-UX

В реализации системы UNIX в Hewlett-Packard представлены некоторые сокращенные имена команд для команды **ls** с опциями, используемыми наиболее часто:

Команда системы UNIX/Linux	Эквивалентная команда HP-UX
----------------------------	-----------------------------

ls -F	lsf
ls -l	ll
ls -R	lsr

Система Linux



Система Linux не понимает аббревиатуры **lsf**, **ll** или **lsr**, доступные в системе HP-UX. В системе Linux необходимо использовать полные формы команды **ls** с опциями: **ls -F**, **ls -l** и **ls -R**.

4-9. СЛАЙД: **cd** – изменить каталог

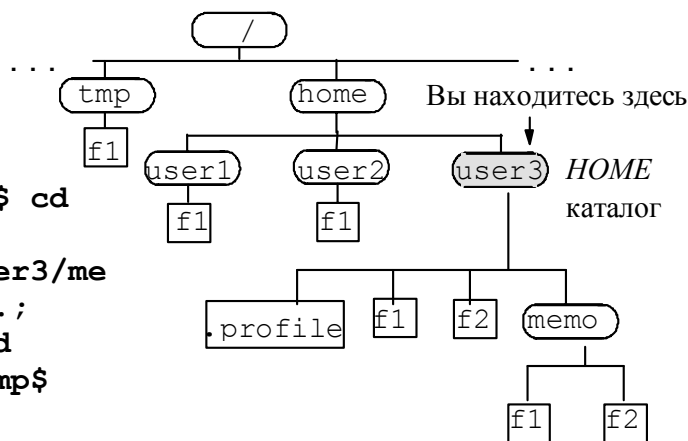
cd — изменить каталог

Синтаксис:

cd [имя_каталога]

Примеры:

```
$ pwd
/home/user3$ cd
memo;
pwd/home/user3/memo$ cd ../..;
pwd/home$ cd
/tmp; pwd/tmp$
cd;
pwd/home/user3
```



Пояснения

Древовидная структура файловой системы является своего рода "дорожной картой", на которой можно увидеть положение всех файлов и каталогов системы. Пользователь всегда находится в каком-то каталоге. Команда **cd** позволяет изменить каталог и тем самым переместиться в другую точку в иерархии файловой системы.

Синтаксис команды:

cd имя_каталога

Здесь *имя_каталога* представляет собой относительное или абсолютное маршрутное имя каталога, в который хочет перейти пользователь. Если выполняется команда **cd** без аргументов, она возвращает пользователя в его входной (*HOME*) каталог. Если пользователь "заблудился" в иерархии файловой системы, он всегда может выполнить команду **cd** и вновь оказаться в своем *HOME* каталоге.

ПРИМЕЧАНИЕ: При использовании команды **cd** для перемещения по файловой системе, для того чтобы определить свое положение в иерархии файловой системы, полезно время от времени выполнять команду **pwd**.

Дополнительные возможности команды **cd** в POSIX shell

POSIX shell имеет возможность запомнить положение (имя) предыдущего каталога. Здесь команда **cd** также изменяет каталоги по указанию пользователя, но, кроме этого, имеет дополнительные возможности, позволяющие сэкономить ввод.

Команда **cd** имеет специальную память для хранения предыдущего каталога (в переменной среды *OLDPWD*), и к ней можно получить доступ с помощью команды **cd -**.

```
$ pwd  
/home/user3/tree
```

```
$ cd /tmp
```

```
$ pwd  
/tmp
```

```
$ cd -          Возвращает в предыдущий каталог  
$ pwd  
/home/user3/tree
```

4–10. СЛАЙД: Команда **find**

Команда **find**

Синтаксис:

find *список_имен* *выражение*

Осуществляет упорядоченный поиск в файловой системе.

список_имен – это список имен каталогов, в которых осуществляется поиск.

выражение – задает критерий поиска и действия

Пример:

```
$ find . -name  
.profile./.profile$
```

Пояснения

Команда **find** – единственная команда, которая выполняет автоматический поиск в файловой системе. Эта команда выполняется очень медленно и требует значительных ресурсов процессора. Поэтому исполнять эту команду следует только при необходимости.

Список_имен – это список маршрутных имен, обычно состоящий из одного имени каталога. Часто указывается имя (.). Рекурсивно отыскиваются маршрутные имена файлов, удовлетворяющих критерию, заданному в **выражении**. Когда команда **find** обнаруживает соответствие критерию, она выполняет действия, заданные в выражении. Одно из наиболее часто выполняемых действий – печать найденных имен файлов.

Выражение состоит из ключевых слов и аргументов, которые определяют критерий поиска и действие, которое требуется выполнить, когда обнаружится соответствие критерию. Один из моментов, затрудняющих использование команды **find**, состоит в том, что ключевым словам, используемым в выражении, должен предшествовать знак "минус" (-), и это выглядит так, будто аргументы предшествуют опциям.

4-11. СЛАЙД: `mkdir` и `rmdir` – создать и удалить каталоги

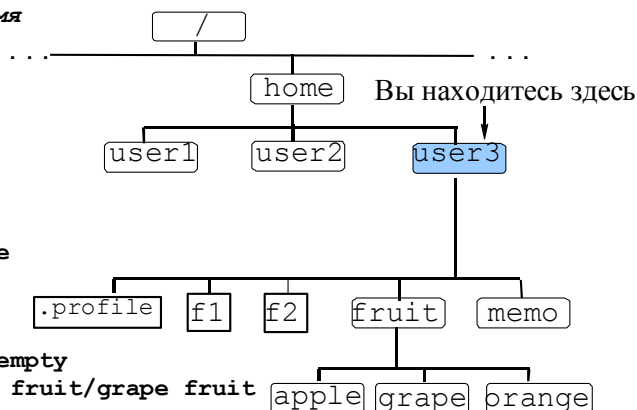
`mkdir` и `rmdir` – создать и удалить каталоги

Синтаксис:

```
mkdir [опции] маршрутное_имя
rmdir маршрутное_имя
```

Примеры:

```
$ pwd
/home/user3
$ mkdir fruit
$ mkdir fruit/apple
$ cd fruit
$ mkdir grape orange
$ rmdir orange
$ cd ..
$ rmdir fruit
rmdir: fruit not empty
$ rmdir fruit/apple fruit/grape fruit
```



Пояснения

Создание каталога

Команда **mkdir** позволяет создавать каталоги. Эти каталоги затем могут быть использованы для организации файлов. Когда создается каталог, в нем автоматически создаются два подкаталога с именами "точка" (.) и "точка-точка" (..), которые представляют текущий и родительский каталоги. Следует отметить, что созданные каталоги не изменяют положения пользователя в иерархии файловой системы.

Команда **mkdir** допускает использование следующих опций:

-m режим После создания каталога права доступа для него должны быть установлены равными значению, заданному параметром **режим**.

-p Создавать промежуточные каталоги, если они не существуют.

По умолчанию, если задается абсолютное или относительное маршрутное имя создаваемого каталога, все промежуточные каталоги, указанные в имени, должны существовать. В противном случае нужно использовать опцию **-p**. Приведенная ниже команда создаст каталог **fruit**, если он еще не существует:

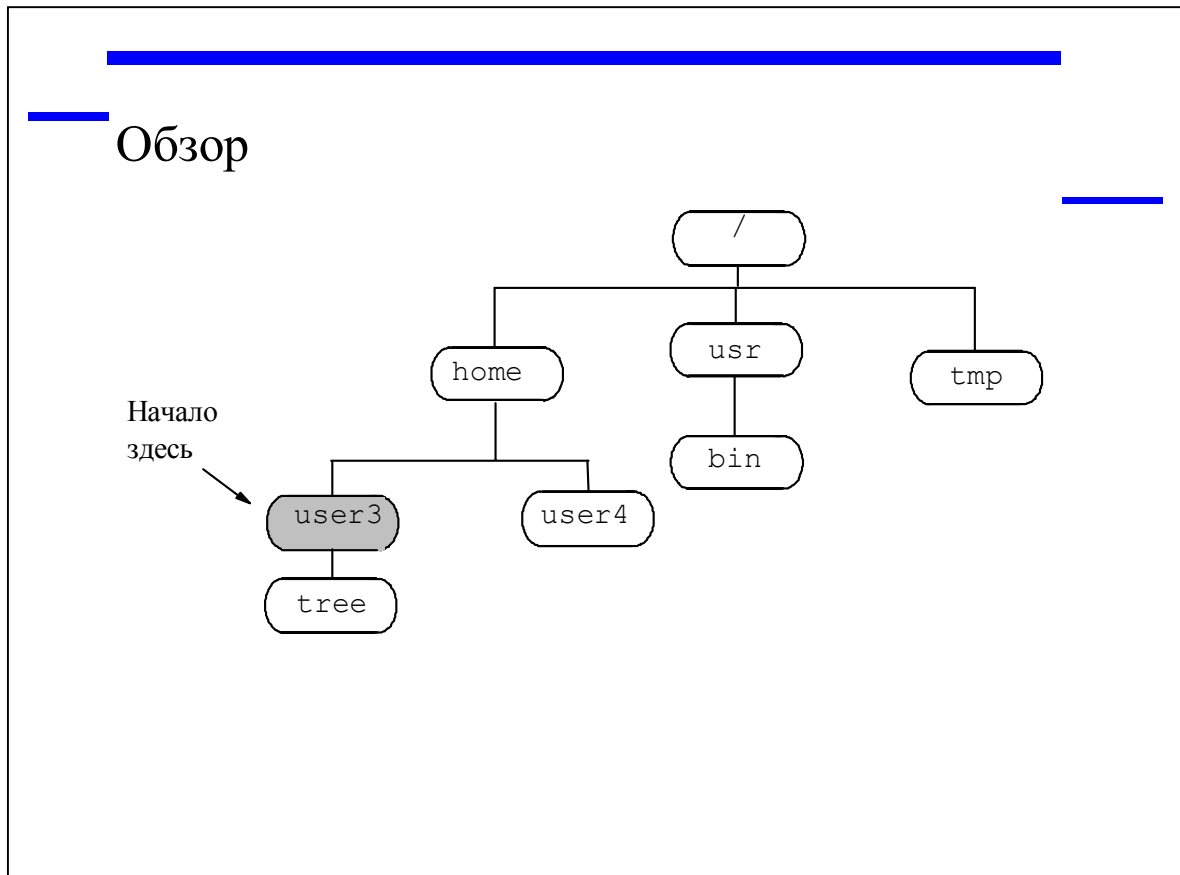
```
$ mkdir -p fruit/apple fruit/grape fruit/orange
```

Удаление каталога

Команда **rmdir** позволяет удалять каталоги. Для того чтобы можно было удалить каталог, он должен быть пустым (т.е. не содержать других элементов, кроме точки и точки-точки). Нельзя удалить текущий каталог и каталог, который находится между корневым и текущим каталогами.

Обе команды могут иметь много аргументов. Аргументы команды **mkdir** представляют имена новых каталогов, не существующих в системе. Аргументами команды **rmdir** должны быть имена существующих каталогов. Как и в любой другой команде, использующей в качестве аргументов имена файлов или каталогов, в этих командах можно задавать и абсолютные, и относительные маршрутные имена.

4–12. СЛАЙД: Обзор



Пояснения

С помощью примера, представленного на слайде, предлагается проработать использование команд **cd** и **pwd**, а также использование относительных и абсолютных маршрутных имен.

Пусть каталог имеет структуру, приведенную на слайде, и текущим является каталог **user3**. Где окажется пользователь после выполнения каждой из приведенных ниже команд **cd**?

\$ pwd	/home/user3
\$ cd ..	
\$ pwd	_____
\$ cd usr	
\$ pwd	_____
\$ cd /usr	
\$ pwd	_____
\$ cd ../tmp	
\$ pwd	_____
\$ cd .	
\$ pwd	_____

4–13. Слайд: Файловая система: заключение

Файловая система: заключение

Файл	Контейнер для данных
Каталог	Контейнер для данных и других каталогов
Дерево	Иерархическая структура файловой системы UNIX
Маршрутное имя	Идентифицирует положение файла или каталога в иерархии файловой системы
HOME	Представляет маршрутное имя входного каталога
pwd	Выводит на экран имя текущего каталога
cd	Изменяет текущий каталог
ls	Выводит содержимое каталога
find	Находит файлы, заданные опциями
mkdir	Создает каталоги
rmdir	Удаляет каталоги

Пояснения

4–14. УПРАЖНЕНИЯ: Файловая система

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Каково имя вашего *HOME* каталога?
2. Находясь в *HOME* каталоге и используя команду **ls**, определите полную древовидную структуру каталогов, начиная с каталога **tree**. Представьте графически структуру дерева каталогов, изобразив каталоги кружками. В случае необходимости, используйте отдельную страницу.
3. Каково абсолютное маршрутное имя файла **labrador** в дереве каталога, полученного в предыдущем упражнении? Каково его относительное маршрутное имя по отношению к вашему *HOME* каталогу?
4. Перейдите из *HOME* каталога в каталог **retriever**. Используя относительное маршрутное имя, перейдите в каталог **shepherd**. Используя опять же относительное маршрутное имя, перейдите в каталог **car.models**. Наконец, вернитесь в *HOME* каталог. Какие команды вы использовали? Каким образом вы узнавали, что находитесь в нужном каталоге?

5. Создайте каталог **junk** в своем *HOME* каталоге. Сделайте этот каталог текущим. Какие команды вы использовали? Каково абсолютное маршрутное имя нового каталога?

6. Находясь в *HOME* каталоге, с помощью одной команды создайте следующие каталоги:

```
junk/dirA/dir1  
junk/dirA  
junk/dirA/dir2  
junk/dirA/dir1/dirc
```

Возникли ли у вас какие-нибудь проблемы? Если возникли, прежде чем выполнять упражнение снова, удалите каталоги, созданные в результате неправильных действий. Какую команду вы использовали?

7. Находясь в *HOME* каталоге, выведите содержимое каталога **dirA**, находящегося в каталоге **junk**. Используйте и относительное, и абсолютное маршрутные имена. Какие команды вы использовали?

8. Находясь в *HOME* каталоге и используя только команду **rmdir**, удалите все подкаталоги, находящиеся в каталоге **junk**. Как можно выполнить это упражнение, используя единственную команду **rmdir**?

9. Вернитесь в *HOME* каталог. С помощью единственной команды выведите подробные характеристики файлов **cp** и **vi** (из каталога **/usr/bin**). Попробуйте использовать и абсолютное, и относительное маршрутное имя.

Модуль 5 — Работа с файлами

Цели:

Изучив этот модуль, вы сможете:

- Использовать общие команды манипулирования файлами системы UNIX
- Объяснить назначение системы спулинга строчных принтеров
- Идентифицировать и использовать команды, используемые для взаимодействия с системой спулинга строчных принтеров
- Наблюдать за состоянием системы спулинга строчных принтеров

5–1. СЛАЙД: Что такое файл?

Что такое файл?

Хранилище данных или ссылка на устройство

- Каждый файл имеет имя и может хранить данные, которые размещаются на диске
- Существует несколько различных типов файлов:
 - обычные файлы
 - текст, числовые и графические данные
 - исполняемые программы
 - каталоги
 - файлы устройств

Пояснения

В системе UNIX все представляется в виде файлов. В системе существуют:

Обычные файлы	Текст, почтовые сообщения, числовые данные, графические данные, исходные коды программ
Программы	Исполняемые программы, такие как ksh , who , date , man и ls .
Каталоги	Специальные файлы, в которых записаны имена и идентификаторы файлов и каталогов, содержащихся в них.
Устройства	Специальные файлы, обеспечивающие интерфейс с устройствами, такими как диски, терминалы, принтеры и оперативная память.

Файл представляет собой имя и ассоциированные с этим именем данные, хранящиеся на устройстве массовой памяти, обычно на диске. Так как это относится к системе UNIX, файл – не более чем поток байтов данных. В них нет предопределенных записей, полей, маркеров конца записи или конца файла. Это дает разработчикам приложений большие возможности с точки зрения определения собственных внутренних характеристик файла.

Обычный файл обычно содержит текст в коде ASCII и создается с помощью текстового редактора в интерактивном режиме.

Файл-программа – это обычный файл, который содержит исполняемые инструкции. Он может содержать скомпилированный код, который нельзя вывести на терминал (**mail**, **who**, **date**), или содержать команды shell'a системы UNIX, которые могут быть выведены на экран терминала (**.profile**, **.logout**); такой файл обычно называют **shell процедурой**.

Каталог – это специальный файл, содержащий имена файлов и каталогов, хранящихся в нем. В нем также для каждого элемента хранится **номер индексного дескриптора (номер inode)**, который определяет, где находятся информация о файле и адреса физического хранения данных в файловой системе. (Следует отметить, что каталог – это не обычный файл.)

Файл устройства – это специальный файл, который предоставляет интерфейс между ядром системы и реальным внешним устройством. Так как эти файлы выполняют только функции интерфейса, они никогда не хранят реальных данных. Эти файлы обычно хранятся в каталоге **/dev**, причем для каждого внешнего устройства, с которым взаимодействует компьютер, предусмотрен свой файл.

5–2. СЛАЙД: Что можно делать с файлами?

Что можно делать с файлами?

ls	Просмотреть характеристики файла
cat	Просмотреть содержимое файла
more	Просмотреть содержимое файла поэкранно
tail	Вывести окончание файла
lp	Напечатать файл
cp	Создать копию файла
mv	Изменить имя файла или каталога
mv	Переместить файл в другой каталог
ln	Создать другое имя для файла
rm	Удалить файл

Пояснения

Известно, что наиболее активная деятельность в системе UNIX сосредоточена вокруг файлов и каталогов; существует большое количество команд, работающих с файлами и каталогами.

Ранее были рассмотрены некоторые начальные команды, позволяющие работать с каталогами. В данном модуле будут представлены дополнительные команды, которые можно использовать для работы с файлами и каталогами.

При изучении данного модуля придется также создавать файлы и манипулировать с их содержимым. Обычно это делается с помощью редактора, например, **vi**.

5–3. СЛАЙД: Характеристики файла

Характеристики файла

```

$ ls -l
-rw-r--r-- 1 user3 class 37 Jul 24 11:06 f1
-rwxr-xr-x 1 user3 class 52 Jul 24 11:08 f2
drwxr-xr-x 2 user3 class 1024 Jul 24 12:03 memo
  
```

↑
Тип
файла

↑
Права
доступа

↑
Ссылки

↑
Владелец

↑
Группа

↑
Размер
файла

↑
Временная
метка

↑
Имя
файла

Пояснения

С файлом связаны разные характеристики. Их можно увидеть, выполнив команду **ls -l**.

Тип файла	Обычный файл или специальный файл
Права доступа	Определяют возможности доступа к файлу
Ссылки	Количество имен файлов, ассоциированных с одним и тем же набором данных
Владелец	Идентификатор пользователя, владеющего файлом
Группа	Идентификация группы для доступа к файлу
Размер	Количество байтов, содержащихся в файле
Временная метка	Дата и время последней модификации файла
Имя файла	Максимум 14 символов (255 символов, если поддерживаются длинные имена файлов)

Задание имени файла

- Максимальная длина имени – 14 символов
- Если поддерживаются длинные имена файлов, тогда максимальная длина имени - 255 символов
- Имя обычно формируется с использованием букв (a-zA-Z), цифр (0-9), точки (.), тире (-), и знака подчеркивания (_)

Многие символы (специальные символы shell'a), такие как *пробел* или *прямой слеш*, имеют для shell'a особый смысл, поэтому эти символы не могут быть использованы в имени файла. Другие примеры специальных символов: ***, *<*, *>*, **, *\$* и *|*. Если попытаться использовать эти символы в имени файла, результат окажется совсем не таким, какой ожидается.

Если имя файла должно состоять из двух слов, слова в имени соединяются символом подчеркивания:

```
$ cd a dir
```

Неправильный синтаксис –

*в команде **cd** указаны два аргумента*

```
$ cd a_dir
```

Правильный синтаксис –

*в команде **cd** указан один аргумент*

В системе UNIX точка (.) представляет собой обычный символ и поэтому может появиться в любом месте (и любое количество раз) в имени файла. Это означает, что имена файлов **a.bcd***efg*, **a.b.c.d** и **a...b** корректны.

Точка имеет особое значение только тогда, когда она является *первым* символом в имени файла; в этом случае она определяет *скрытый файл*. Имена файлов, начинающиеся с точки, можно увидеть с помощью команды **ls -a**.

Типы файлов

Система UNIX поддерживает много типов файлов. Тип файла определяется первым символом в выводе команды **ls -l**. Обычно встречаются файлы следующих типов:

- обычный файл
- d** каталог
- l** символическая ссылка на файл
- n** сетевой специальный файл
- c** байт-ориентированный файл устройства (терминалы, принтеры)
- b** блок-ориентированный файл устройства (диски)
- p** именованный канал (канал для межпроцессного взаимодействия)

5-4. СЛАЙД: **cat** – вывод содержимого файла

cat — вывод содержимого файла

Синтаксис:

cat [файл...] Объединяет и выводит содержимое файлов

Примеры:

```
$ cat remind
Your mother's birthday is November 29.
$ cat note remind
TO:  Mike Smith
The meeting is scheduled for July 29.
Your mother's birthday is November 29.
```

Пояснения

Команда **cat** используется для объединения и вывода содержимого нескольких файлов как единого целого. Она никак не форматирует вывод файлов, в том числе не включает разделитель между концом одного файла и началом другого. Синтаксис команды:

```
cat [файл ...]
```

Обычно команда **cat** используется для просмотра содержимого одного файла, например, команда

```
cat funfile
```

выводит содержимое файла **funfile** на экран. Однако если файл слишком большой и не умещается на одном экране, его содержимое выводится на экран слишком быстро и его невозможно прочитать. Поэтому нужен более удобный способ вывода содержимого файлов на экран.

Если используется команда **cat** без аргументов, она ожидает ввода с клавиатуры. В этом случае команда работает так же, как команды **mail** и **write**. Чтобы завершить ввод, нужно нажать клавиши **Return** и **Ctrl** + **d**. В процессе ввода с клавиатуры вводимые строки выводятся на экран.

ВНИМАНИЕ:

Если файл содержит управляющие символы, что имеет место для скомпилированных программ, и его содержимое выводится на экран с помощью команды **cat**, то терминал может "зависнуть". Вернуть терминал в работоспособное состояние можно одним из двух следующих способов.

Способ 1:

1. Попробуйте выйти из системы – нажмите **Return**, затем введите и выполните команду **exit**.
2. Выключите, а затем вновь включите терминал.
3. Вновь войдите в систему – все должно быть нормально.

Способ 2:

1. Нажмите клавишу **Break**.
2. Нажмите одновременно **Shift** + **Ctrl** + **Reset**.
3. Нажмите **Return**.
4. Выполните команду: **tset -e -k**.
5. Выполните команду: **tabs**.

Если это не помогает, нужно обратиться к системному администратору, чтобы он прервал сеанс связи пользователя с системой.

5–5. СЛАЙД: **more** – вывод содержимого файла

more — вывод содержимого файла

Синтаксис:

more [имя_файла]... Выполняет постранично вывод содержимого файла

Пример:

```
$ more funfile
```

```
.  
. .  
.
```

```
--funfile (20%)--
```

Q или **q** Завершить команду **more**

Return

Вывести следующую строку

Space

Вывести следующую страницу

Пояснения

Команда **more** выводит на экран содержимое указанных файлов. В каждый момент времени команда выводит на экран только часть текста размером с экран. Чтобы увидеть следующий экран с информацией, следует нажать клавишу **Space**. Чтобы увидеть на экране следующую строку текста, следует нажать клавишу **Return**. Чтобы завершить работу команды **more**, следует нажать клавишу **q**.

Команда **more** обладает многими другими возможностями. Чтобы узнать о них, следует обратиться к соответствующей статье оперативного *Справочника HP-UX*.

5–6. СЛАЙД: **tail** – вывод окончания файла

tail — вывод окончания файла

Синтаксис:

```
tail [-n] [имя_файла]...
```

Вывести на экран
окончание файла

Пример:

```
$ tail -1 note  
soon as it is available.
```

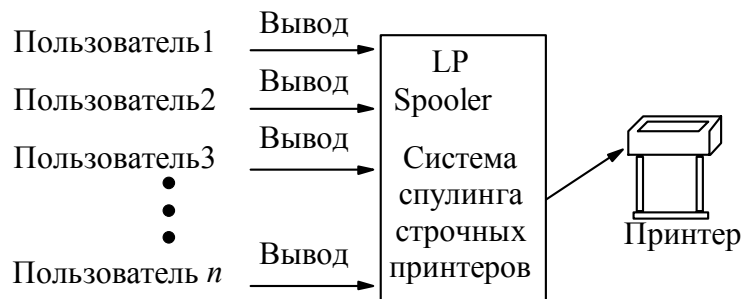
Пояснения

Команда **tail** используется, когда надо вывести на экран последние *n* строк файла. (Если в команде *n* не указано, по умолчанию принимается *n* = 10.) Эта команда особенно полезна для длинных файлов протокола, в конец которых периодически добавляется информация. С ее помощью можно сразу же увидеть последние сообщения в протоколе, не просматривая весь файл с помощью команд **cat** или **more**.

5–7. СЛАЙД: Система спулинга строчных принтеров

Система спулинга строчных принтеров

- Система спулинга строчных принтеров (**lp spooler**) координирует работу принтера.
- Она дает возможность пользователям:
 - организовывать очередь файлов к принтерам
 - определять состояние принтеров и очередей запросов к ним
 - снимать любое задание для принтера



Пояснения

ОС UNIX располагает утилитой, называемой системой спулинга строчных принтеров (или **lp spooler**), которая используется для конфигурирования и управления процессом печати в системе. Система спулинга – это механизм, который принимает запросы на печать от всех пользователей системы, затем конфигурирует соответствующим образом принтер и, наконец, последовательно обрабатывает (печатает) все запросы. Представьте себе те проблемы, с которыми пришлось бы столкнуться при отсутствии такой системы. Каждый раз, когда пользователь захочет напечатать файл, он должен быть уверен в том, что никто больше в данный момент времени не занят тем же. Два пользователя не могут в один и тот же момент времени выводить информацию на принтер.

Система спулинга имеет много возможностей, которые позволяют выводить информацию на печать при минимальном вмешательстве администратора системы в этот процесс. Пользователь передает системе свои запросы на печать, где они ожидают своей очереди. Пользователь может проверить, какие файлы стоят в очереди на печать, и выяснить состояние системы спулинга. Он может также удалить из очереди запрос на печать, если решит, что соответствующая информация не должна быть напечатана.

5–8. СЛАЙД: Команда **lp**

Команда **lp**

- Организует очередь файлов, подлежащих выводу на печать
- Назначает запросу в очереди уникальный идентификатор (request ID)
- Обладает рядом опций для настройки маршрутизации и режима печати

Синтаксис:

```
lp [ -dпринтер] [-опции] имя_файла ...
```

Примеры:

```
$ lp report
request id is dp 112 (1 file)
$ lp -n2 memo1 memo2
request id is dp 113 (2 files)
$ lp dlaser -t"confidential" memo3
request id is laser114 (1 file)
$
```

Пояснения

Команда **lp** позволяет пользователю поставить файлы в очередь на печать. Каждому запросу, инициированному с помощью команды **lp**, присваивается уникальное значение идентификатора задания (называемого идентификатором запроса – request ID).

Команда **lp** ставит в очередь на печать файл или же информацию, прочитанную из стандартного ввода.

В простейшем случае команда **lp** имеет один аргумент – имя файла. В этом случае команда поставит указанный файл в очередь к принтеру, назначенному в системе принтером по умолчанию.

Команда **lp** имеет ряд опций, позволяющих настраивать маршрутизацию и режим печати заданий.

Команда **lp** имеет следующий синтаксис:

```
lp [-dадресат] [-nчисло] [-оопция] [-tзаголовок] [-w] [файл ... ]
```

Ниже приводится описание некоторых опций команды **lp**:

-dадресат *адресат* определяет имя принтера, на котором запрос будет выведен на печать.

-n <i>число</i>	Напечатать указанное <i>число</i> копий по данному запросу (по умолчанию 1).
-o <i>опция</i>	Определяет опции печати, специфические для конкретного принтера, например, тип шрифта, шаг печати, толщину линий, неструктурированные данные (для графических данных), и т.д.
-t <i>заголовок</i>	Напечатать указанный <i>заголовок</i> на титульном листе печатного документа. Титульный лист позволяет идентифицировать владельца печатного документа.
-w	Вывести сообщение на терминал пользователя после того, как файл будет напечатан.

Полный список опций команды можно найти в статье **lp(1)** *Справочника HP-UX*.

В первом примере, приведенном на слайде, показан самый простой способ использования команды **lp**. В этом примере файл **report** посылается для печати на принтер, определенный в системе как принтер по умолчанию. Команда **lp** возвращает идентификатор запроса и количество файлов, поставленных в данном запросе в очередь к принтеру. В данном примере файл **report** поставлен в очередь к принтеру **dp**, и идентификатор данного запроса – **dp-112**.

Во втором примере на печать посылаются два файла – **memo1** и **memo2**, при этом будут напечатаны по две копии каждого файла (**-n2**).

В третьем примере, используя опцию **-d**, пользователь явно указал, в очередь к какому принтеру должен быть поставлен запрос. Печатный документ будет иметь заголовок "confidential" (конфиденциальная информация)

Система Linux



В системе Linux используется BSD-система спулинга принтеров и соответствующие команды. Позднее в этом модуле в таблице будет приведено сравнение команд систем HP-UX и Linux, относящихся к печати.

Для того чтобы напечатать файл, в системе Linux используется команда **lpr**. Например:

```
$ lpr funfile
```

По умолчанию принтер, установленный в системе как принтер по умолчанию, имеет имя **lp**. Администратор системы может установить имя принтера по умолчанию так, как это удобно пользователям, настроив управляющие элементы среды пользователя соответствующим образом.

Если пользователю нужно вывести файл на печать на альтернативном принтере, с помощью опции **-P** должно быть указано имя принтера.

Например:

```
$ lpr -Pnewprinter funfile
```

ПРИМЕЧАНИЕ: Между опцией **-P** и именем принтера не должно быть пробелов.

Более подробную информацию о команде **lpr** можно найти в соответствующей статье справочника.

5–9. СЛАЙД: Команда **lpstat**

Команда **lpstat**

Синтаксис:

lpstat [-t]

- **lpstat** Сообщает о запросах, поставленных в очередь на печать
- **lpstat -t** Сообщает состояние планировщика заданий, имя принтера по умолчанию, состояние принтеров и список всех запросов, стоящих в очереди на печать

Пояснения

Команда **lpstat** сообщает о состоянии различных компонентов системы спулинга.

Команда **lpstat** без опций сообщает о запросах, которые в настоящее время находятся в очереди на печать.

```
$ lpstat
rw-55          john          4025      Jul 6 14:26:33 1994
$
```

Опция **-t** предписывает выводить полную информацию о состоянии всех принтеров, установленных в системе. Результаты работы команды **lpstat -t** позволяют получить следующую информацию:

```
$ lpstat -t
scheduler is running
system default destination: rw
device for rw: /dev/lp2235
rw accepting requests since Jul 1 10:56:20 1994
printer rw now printing rw-55. enabled since Jul 4 14:32:52 1994
rw-55          john          4025      Jul 6 14:26:33 1994 on rw
rw-56          root          966       Jul 6 14:27:58 1994
$
```

`scheduler is running`

Scheduler (планировщик) – это программа, которая пересылает запрос на печать на соответствующий принтер. Если планировщик не работает, ничего не будет печататься.

`system default destination: rw`

Это сообщение означает, что имя принтера, определенного в системе как принтер по умолчанию, есть **rw**. Если используется команда **lp** без опции **-d принтер**, данный запрос будет направлен на принтер с именем **rw**. Следует заметить, принтер, установленный в системе как принтер по умолчанию, может иметь любое имя (например, **lp**).

`device for rw: /dev/lp2235`

Эта информация сообщает планировщику, каким образом принтер подсоединен к компьютеру.

`rw accepting requests`

Это сообщение означает, что система спулинга будет направлять файлы в очередь к принтеру **rw**.

`printer rw now printing rw-55`

В данный момент печатается запрос с идентификатором **rw-55**.

`enabled`

Запросы могут быть напечатаны на принтере **rw**. Если принтер находится в нерабочем состоянии (выводится сообщение **disabled**), пользователь может посылать запросы к этому принтеру, но они будут напечатаны только тогда, когда принтер вновь окажется в рабочем состоянии (снова будет выведено сообщение **enabled**).

Оставшиеся строки в рассмотренном примере представляют запросы на печать. Поля в этих строках содержат: идентификатор запроса, имя пользователя, пославшего запрос, размер запроса и дату, когда выдан запрос на печать.

Система Linux



В системе Linux командой, эквивалентной **lpstat**, является команда **lpc**. Команда **lpc** может быть выполнена только администратором системы (пользователем с именем **root**).

Чтобы увидеть состояние очереди запросов на принтер, установленный как принтер по умолчанию, используется команда **lpq**.

```
[user1]$ lpq
Rank   Owner      Job   Files              Total Size
1st    user1      45    funfile            517 bytes
```

В листинге представлены все задания в очереди к принтеру, установленному как пользовательский принтер по умолчанию (или на системный принтер по умолчанию, если для данного пользователя не был установлен собственный принтер по умолчанию).

Чтобы посмотреть запросы, находящиеся в очереди к другому принтеру, в команде должно быть указано имя этого принтера:

```
[user1]$ lpq matrix1
```

Rank	Owner	Job	Files	Total Size
------	-------	-----	-------	------------

Если в очереди к указанному принтеру нет запросов, выводится только заголовок отчета, что и показано в приведенном выше примере.

5–10. СЛАЙД: Команда **cancel**

Команда **cancel**

Синтаксис:

```
cancel id [ id ... ]  
cancel принтер [ принтер ... ]
```

Примеры:

- Снять задание, поставленное в очередь командой **lp**
\$ cancel dp-115
- Снять текущее задание на указанном принтере
\$ cancel laser

Пояснения

Команда **cancel** используется для удаления задания из очереди к принтеру. Если снимается текущее задание, начинает печататься следующий запрос. Необходимость в снятии запроса может возникнуть, если печатается очень большой файл или если по ошибке на печать был направлен двоичный файл (например, **/usr/bin/cat**). Следует помнить, что команда **lp** обычно печатает текстовые файлы. Попытка вывести что-либо другое только лишь "запутает" принтер и приведет к бесполезной трате бумаги, если только не будет указана соответствующая опция (например, **-oraw** для графических данных).

Чтобы снять запрос, нужно сообщить системе спулинга, какой именно запрос снимается, указав соответствующий аргумент в команде **cancel**. Могут использоваться аргументы двух типов:

- идентификатор запроса (который выводится командами **lp** или **lpstat**)
- имя принтера

Если в команде **cancel** указывается идентификатор запроса, из очереди на печать удаляется соответствующий запрос. Если в команде **cancel** указывается имя принтера, на соответствующем принтере печать текущего задания будет прервана и начнет выполняться (печататься) следующий запрос из очереди.

```
$ lpstat
rw-113          mike      6275   Jul  6 18:46 1994
rw-114          mike      3349   Jul  6 18:48 1994
rw-115          mike      3258   Jul  6 18:49 1994
$ cancel rw-115
request "rw-115" canceled
$ lpstat
rw-113          mike      6275   Jul  6 18:46 1994
rw-114          mike      3349   Jul  6 18:48 1994
$ cancel rw
request "rw-113" canceled
$ lpstat
rw-114          mike      3349   Jul  6 18:48 1994
```

Любой пользователь может выполнять эту команду и снимать любой запрос. Пользователь даже может снять запрос, поставленный в очередь другим пользователем. В этом случае пострадавшему пользователю будет отправлено почтовое сообщение, в котором будет указано имя пользователя, снявшего запрос. Системный администратор может ограничить пользователей в правах, разрешив им удалять только свои запросы.

Система Linux



В системе Linux командой, эквивалентной команде **cancel**, является команда **lprm**. Команда **lprm** позволяет удалять из очереди запрос только владельцу этого запроса. Пользователь не может удалить из очереди на печать запрос, поставленный другим пользователем. Исключением из этого правила является пользователь **root** (*администратор системы*).

Каждый раз, когда нужно удалить задание из очереди на печать, необходимо указывать номер этого задания. В одной команде можно указать несколько номеров заданий.

```
[user1]$ lpq
Rank  Owner      Job  Files                      Total Size
1st   user1       45   funfile                   517 bytes
2nd   user1       46   /etc/passwd               1904 bytes
[user1]$
[user1]$ lprm 45
dfA045ABQzluw dequeued
cfA045ABQzluw dequeued
[user1]$
[user1]$ lpq
Warning: lp is down: printing disabled
Warning: no daemon present
Rank  Owner      Job  Files                      Total Size
1st   user1       46   /etc/passwd               1904 bytes
```

Команды печати систем HP-UX и Linux — сравнение



HP-UX	Linux
lp	lpr
lpstat	lpq
cancel	lprm

Более подробную информацию о командах печати системы Linux следует искать в соответствующих статьях справочника Linux. Например:

```
$ man lpr
...
NAME
    lpr - off line print

SYNOPSIS
    lpr [-Pprinter] [-#num] [-C class] [-J job] [-T title] [-U user]
    [-i [numcols]] [-1234 font] [-wnum] [-cdfghlnmprstv] [name ...
...

```

5–11. СЛАЙД: **cp** – копирование файлов

cp — копирование файлов

Синтаксис:

<code>cp [-i] file new_file</code>	Копирование файла
<code>cp [-i] file [file...] dest_dir</code>	Копирование файлов в каталог
<code>cp -r [-i] dir dir...] dest_dir</code>	Копирование каталогов

Примеры:

```
$ ls -F
f1 f2* memo/ note remind
$ cp f1 f1.copy
$ ls -F
f1 f1.copy f2* memo/ note remind
$ cp note remind memo
$ ls -F memo
note remind
```

Пояснения

Команда **cp** используется для создания копии одного или нескольких файлов. Применяя эту команду, нужно принимать во внимание следующее.

- В команде требуется указать, *по крайней мере, два аргумента* – источник и приемник.
- Для любого из аргументов можно использовать как относительное, так и абсолютное маршрутное имя.
- При копировании одного файла в качестве приемника можно указать маршрутное имя и файла, и каталога. Если в качестве приемника указывается имя файла и этот файл не существует, он будет создан. Если файл-приемник существует, его содержимое будет замещено содержимым файла-источника. Если в качестве приемника указывается имя каталога, файл будет скопирован в указанный каталог под своим именем.

- Опция **-i** (интерактивный режим) выводит на экран предупреждение, если файл-приемник существует, и требует подтвердить перезаписывание существующего файла.

<code>\$ cp f1 f1.copy</code>	Создает в текущем каталоге файл с именем f1.copy
<code>\$ cp f1 memo</code>	Создает в каталоге memo файл с именем f1
<code>\$ cp f1 memo/f1.copy</code>	Создает в каталоге memo файл с именем f1.copy

- Когда копируются несколько файлов, приемником должен быть каталог.

```
$ cp note remind memo
```

- Файл не может быть скопирован сам в себя.
\$ cp f1 f1
cp: f1 and f1 are identical
- Если нужно скопировать каталог(и), должна быть указана опция **-r** (рекурсивно).

ВНИМАНИЕ: По умолчанию **cp** пересоздает существующий файл – без каких-либо предупреждений!

```
$ cp f1 note
$ cat f1
This is a sample file to be copied.
$ cat note
This is a sample file to be copied.
```

5–12. СЛАЙД: **mv** – перемещение или переименование файлов

mv — перемещение или переименование файлов

Синтаксис:

<code>mv [-i] file new_file</code>	Переименование файла
<code>mv [-i] file [file...] dest_dir</code>	Перемещение файла в каталог
<code>mv [-i] dir [dir...] dest_dir</code>	Переименование или перемещение каталогов

Примеры:

<code>\$ ls -F</code>	<code>\$ mv note remind memo</code>
<code>f1 f2* memo/ note remind</code>	<code>\$ ls -F</code>
<code>\$ mv f1 file1</code>	<code>file1 memo/</code>
<code>\$ ls -F</code>	<code>\$ls -F memo</code>
<code>file1 f2* memo/ note remind</code>	<code>file2* note remind</code>
<code>\$ mv f2 memo/file2</code>	<code>\$ mv memo letters</code>
<code>\$ ls -F</code>	<code>\$ ls -F</code>
<code>file1 memo/ note remind</code>	<code>file1 letters/</code>
<code>\$ ls -F memo</code>	
<code>file2*</code>	

Пояснения

Команда **mv** используется для переименования файлов или для перемещения одного или нескольких файлов в другой каталог. Применяя эту команду, нужно принимать во внимание следующее.

- В команде требуется указать, *по крайней мере, два аргумента* – источник и приемник.
- Для любого из аргументов можно использовать как относительное, так и абсолютное маршрутное имя.
- При переименовании одного файла в качестве приемника можно указать маршрутное имя и файла, и каталога. Если в качестве приемника указывается имя файла в текущем каталоге, файл просто переименовывается. Если в качестве приемника указывается имя каталога, исходный файл будет перемещен в указанный каталог. Если файл-приемник не существует, он будет создан.
- Если файл-приемник уже существует, его содержимое будет замещено содержимым файла-источника. Если в качестве приемника указывается имя каталога, файлы будут перемещены в указанный каталог под своими именами.
- Опция **-i** (интерактивный режим) выводит на экран предупреждение, если файл-приемник существует, и требует подтвердить перезаписывание существующего файла.

Модуль 5

Работа с файлами

```
$ mv f1 file1      Переименовать файл f1 в file1 в текущем каталоге
$ mv file1 memo    Переместить файл file1 в каталог memo
$ mv f2 memo/file2 Переместить файл f2 в каталог memo с переименованием
                   его в файл file2
```

- Когда перемещаются несколько файлов, приемником *должен* быть каталог.

```
$ mv note remind memo
```
- Если в качестве источника указан один каталог, он переименовывается и получает имя приемника.

```
$ mv note letter
```

ВНИМАНИЕ: По умолчанию, **mv** приведет к переименованию или перемещению существующих файлов – без каких-либо предупреждений!

```
$ mv file1 note
$ cat file1
cat: cannot open file1
$ cat note
This is a sample file to be copied.
```

5-13. СЛАЙД: `ln` – создание ссылки на файл

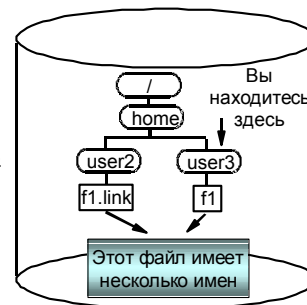
`ln` — создание ссылки на файл

Синтаксис:

`ln file new_file` Создание ссылки на файл
`ln file [file ...] des_dir` Создание ссылок на файлы в каталоге

Примеры:

```
$ ls -l f1
-rw-rw-r-- 1 user3 class 37 Jul 24 11:06 f1
$ ln f1 /home/user2/f1.link
$ ls -l f1
-rw-rw-r-- 2 user3 class 37 Jul 24 11:06 f1
$ ls -l /home/user2
-rw-rw-r-- 2 user3 class 37 Jul 24 11:06 f1.link
$ ls -li f1 /home/user2/f1.link
1789 f1 1789 /home/user2/f1.link
```



Пояснения

Ссылки предоставляют механизм, позволяющий с помощью нескольких имен ссылаться на одни и те же данные, размещенные на диске. Они полезны, когда много пользователей хотят совместно работать с одним и тем же файлом, но в то же время предпочитают осуществлять доступ к этому файлу из своего каталога. Если пользователь **user3** модифицирует файл **f1**, пользователь **user2** увидит эти изменения, когда позднее обратится к файлу **f1.link**.

ВНИМАНИЕ:

Система UNIX не запрещает нескольким пользователям одновременно обращаться к одному и тому же файлу и модифицировать его. Каждый пользователь будет иметь собственное представление относительно того, какие изменения осуществлены в файле, но пользователь, последним сохранивший этот файл, определит окончательную версию файла, которая и будет размещена на диске. Обязанность приложения сообщить пользователю, что данный файл уже открыт для модификации, и, возможно, запретить другим пользователям доступ к файлу, который уже открыт.

Когда несколько файлов связаны друг с другом, количество ссылок на файл, которое выводится при выполнении команды `ls -l`, будет больше единицы. Если какая-то ссылка удаляется, то это приведет только к уменьшению количества ссылок на файл. Содержимое файла сохраняется, пока количество ссылок на файл не уменьшится до нуля, и только тогда дисковое пространство, занятое файлом, освобождается.

Пример

```
$ ls -l f1
-rw-rw-r-- 1 user3 class      37   Jul 24 11:06 f1
$ ln f1 /home/user2/f1.link
$ ls -l f1
-rw-rw-r-- 2 user3 class      37   Jul 24 11:06 f1
$ ls -l /home/user2
-rw-rw-r-- 2 user3 class      37   Jul 24 11:06 f1.link
$ ls -i f1 /home/user2/f1.link
1789 /home/user2/f1.link      1789 f1
```

5–14. СЛАЙД: **rm** – удаление файлов

rm — удаление файлов

Синтаксис:

rm [-if] <i>файл</i> [<i>файл...</i>]	Удаление файлов
rm -r[if] <i>каталог</i> [<i>каталог...</i>]	Удаление каталогов

Примеры:

```
$ ls -F
f1 f2 fruit/memo/
$ rm f1
$ ls -F
f2 fruit/memo/
$ rm -i f2
f2? <user|y|
$ rm fruit
rm: fruit directory
$ rm -r fruit
```

Пояснения

Команда **rm** используется для удаления файлов. После удаления файлов они не могут быть восстановлены. Команда **rm** должна иметь, по крайней мере, один аргумент (имя файла). Допускается задавать в команде много аргументов. Если в команде указаны несколько имен файлов, все указанные файлы будут удалены.

На слайде представлены наиболее часто используемые опции команды.

- f** требует, чтобы указанные в команде файлы были удалены – пользователю не выводятся никакие сообщения, даже если обнаружена какая-то ошибка.
- r** рекурсивное удаление содержимого всех каталогов, указанных в команде.
- i** Режим интерактивного удаления, требующий, чтобы пользователь подтвердил удаление каждого файла. Ответом пользователя должно быть **y**, если нужно удалить файл, или **n**, если не нужно. Нажатие клавиши **Return** аналогично ответу **n**.

ВНИМАНИЕ: Всегда очень осторожно используйте опцию **-r**. При некорректном использовании это приведет к удалению *ВСЕХ* файлов указанных в команде каталогов. Если файл удален, восстановить его можно, если только имеется резервная копия файлов, сделанная до удаления. Если нужно использовать опцию **-r**, ее следует использовать вместе с опцией **-i**.

Например, `rm -ir имя_каталога`

5–15. СЛАЙД: Команды работы с файлами и каталогами: заключение

Команды работы с файлами и каталогами: заклучение

ls -l	Вывести характеристики файла
cat	Объединить и вывести на экран содержимое файлов
more	Форматировать и вывести на экран содержимое файлов
tail	Вывести на экран окончание файла
cp	Копировать файлы или каталоги
mv	Переместить или переименовать файлы или каталоги
ln	Создать ссылку на файл
rm	Удалить файлы или каталоги
lp	Послать запрос на строчный принтер
lpstat	Вывести информацию о состоянии системы спулинга
cancel	Снять запрос из очереди к принтеру

Пояснения

5–16. УПРАЖНЕНИЯ: Работа с файлами и каталогами

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

Управление файлами

1. Находясь в *HOME* каталоге, используйте команду **cat** для вывода на экран содержимого файла **funfile**. Что произошло? Какая альтернативная команда позволяет управлять выводом на экран содержимого файла?
2. Используйте команду **more** для вывода на экран содержимого каталога **tree**. Что произошло? Какую команду нужно использовать для вывода на экран содержимого каталога?
3. Используйте команду **more** для вывода на экран файла **/usr/bin/ls**. Что произошло? Выведите на экран содержимое файла **/usr/bin/ls** с помощью команды **cat**. Что произошло?
4. Вернитесь в *HOME* каталог. Скопируйте файл **names** в файл **names.cp**. Выведите содержимое обоих файлов, чтобы убедиться в их идентичности.

5. Если файл **names** будет изменен, повлияет это на файл **names.cp**? Модифицируйте файл **names**, скопировав в него файл **funfile**. Что произошло с файлами **names** и **names.cp**?
6. Как можно восстановить исходное содержимое файла **names**? Выполните команду для восстановления файла **names**.
7. Создайте еще одну копию файла **names** с именем **names.new**. Измените имя **names.new** на **names.orig**.
8. Как создать два файла (с именами **names.2nd** и **names.3rd**) , которые ссылаются на содержимое файла **names**?
9. Если вы изменили содержимое файла **names**, повлияет ли это на содержимое файлов **names.2nd** и **names.3rd**? Скопируйте файл **funfile** в файл **names** и выведите характеристики всех файлов, имена которых начинаются с **names**, в длинном формате. Изменилось что-нибудь для файла **names.orig**? **names.2nd**? **names.3rd**?

10. Удалите файл **names**. Что случится с файлами **names.2nd** и **names.3rd**?
11. Используйте интерактивную опцию команды **rm** для удаления файлов **names.2nd** и **names.3rd**.
12. Снова скопируйте файл **names.orig** в файл **names**.

Работа с каталогами

1. Создайте в своем *HOME* каталоге каталог **fruit**. Используя только одну команду, переместите перечисленные ниже файлы из *HOME* каталога в каталог **fruit**:

lime
grape
orange

2. Переместите указанные ниже файлы из *HOME* каталога в каталог **fruit**. В каталоге **fruit** перемещенные файлы должны получить другие имена, также указанные ниже:

Источник Приемник

apple	APPLE
peach	Peach

3. Проанализируйте структуру каталога **tree**, находящегося в вашем *HOME* каталоге. Каталог **tree** требует некоторой реорганизации.

Переместите файлы **collie** и **poodle**, чтобы они оказались в каталоге **dog.breeds**.

Переместите файл **probe** в каталог **sports**.

Переместите файл **taurus** в каталог **sedan**.

В каталоге **tree** создайте новый каталог с именем **horses**.

Скопируйте файл **mustang** в созданный каталог **horses**.

Переместите файл **cherry** в каталог **fruit**, созданный в предыдущем упражнении.

ПОДСКАЗКА: Все эти изменения нужно выполнять, находясь в некотором каталоге, но в каком именно каталоге, на ваш взгляд, вы должны находиться?

4. Переместите каталог **fruit** из *HOME* каталога в каталог **tree**.

5. Сделайте каталог **fruit** текущим рабочим каталогом. Переместите файлы **banana** и **lemon** в каталог **fruit**. ПОДСКАЗКА: Помните, что имя "точка-точка" (..) представляет родительский каталог, а имя "точка" (.) представляет текущий каталог.

"Охотник за мусором" (Дополнительно)

1. В *HOME* каталоге в файле **scaveng.README** вы найдете первый ключ к разгадке для охотника за мусором. Охотник будет использовать файлы и каталоги, размещенные в каталоге **scavenger**, который находится в *HOME* каталоге. В каталоге **scavenger** размещены подкаталоги **north**, **south**, **east** и **west**. В каждом из этих подкаталогов, в свою очередь, находятся подкаталоги **1_mile**, **2_mile**, **3_mile**. Ключи к разгадке (подсказки) в каждом из этих подкаталогов позволяют найти зашифрованное секретное

слово. Например, если подсказка выглядит как "иди на восток на три мили", это значит, что нужно выбрать подкаталог **east/3_mile**, в котором нужно прочитать содержимое файла **README**. В этом файле находится следующая подсказка. Движение по системе подсказок продолжается до тех пор, пока не будет найдено зашифрованное секретное слово. Удачи!

Печать файлов – HP-UX

1. Выведите текущее состояние принтеров в системе спулинга lp spooler и найдите имя принтера по умолчанию.
2. Отправьте файл **funfile** на строчный принтер. Запишите идентификатор запроса, который будет выведен на терминал.
3. Проверьте, что ваш запрос поставлен в очередь к принтеру.
4. Как можно определить, какие файлы других пользователей печатаются? Попробуйте узнать это.

5. Используйте команду **cancel**, чтобы удалить свой запрос из очереди к принтеру. Убедитесь в том, что запрос удален из очереди.

Печать файлов – Linux



1. Выведите текущее состояние очереди к принтеру в системе спулинга принтера.
2. Отправьте файл **funfile** на строчный принтер.
3. Убедитесь в том, что ваш запрос поставлен в очередь к принтеру.
4. Как можно определить, какие файлы других пользователей печатаются? Попробуйте сделать это.
5. Используйте команду **lprm**, чтобы удалить свой запрос из очереди к принтеру. Убедитесь в том, что запрос удален из очереди.

Модуль 6 — Доступ к файлам

Цели

Изучив этот модуль, вы сможете:

- Описывать и изменять атрибуты владельца и группы для файла
- Описывать и изменять права доступа к файлу
- Описывать и устанавливать права доступа, используемые по умолчанию при создании новых файлов
- Описывать, как изменить идентификацию пользователя и группы

6–1. СЛАЙД: Доступ к файлам

Доступ к файлам

Доступ к файлам зависит от идентификатора пользователя и от прав доступа, ассоциированных с файлом. В этом модуле будет показано, как права доступа определяют доступ к файлам для чтения, записи и исполнения

ls (ll , ls -l)	Определить права доступа, установленные для файла
chmod	Изменить права доступа к файлу
umask	Изменить устанавливаемые по умолчанию права доступа
chown	Изменить владельца файла
chgrp	Изменить группу файла
su	Переключить идентификатор пользователя
newgrp	Переключить идентификатор группы

Пояснения

Для каждого файла системы определен пользователь, владеющий этим файлом. Владелец файла обладает абсолютным правом управлять тем, кто из пользователей имеет право доступа к его файлу. Владелец файла имеет возможность разрешать или запрещать другим пользователям доступ к своему файлу.

6-2. СЛАЙД: Кто имеет доступ к файлу?

Кто имеет доступ к файлу?

- Система UNIX располагает тройственной структурой, определяющей, кто имеет право доступа к любому файлу или каталогу:

user владелец файла
group группа пользователей, имеющая право доступа к файлу
other все остальные пользователи

- Команда **ls -l** выводит на экран входные имена пользователя-владельца файла и группы

```
$ ls -l
-rw-r--r-- 1 user3 class 37 Jul 24 11:06 f1
-rwxr-xr-x 1 user3 class 37 Jul 24 11:08 f2
drwxr-xr-x 2 user3 class 1024 Jul 24 12:03 memo
          |      |
          |      |
        владелец группа
```

Пояснения

Система UNIX предоставляет тройственную структуру, определяющую права доступа к файлу:

user определяет владельца файла

group определяет группу пользователей, которая может осуществлять доступ к файлу

other определяет всех других пользователей системы

Для каждого файла в системе определен пользователь, владеющий этим файлом. Владелец файла осуществляет полный контроль над тем, кто имеет то или иное право доступа к его файлу. Владелец файла может разрешить или запретить другим пользователям системы доступ к своему файлу. Владелец решает, какая группа пользователей может иметь права доступа к его файлам. Владелец файла также решает, какие права доступа к его файлам дать остальным пользователям системы. Но если права владения файлом переданы кому-то другому, первоначальный владелец файла уже не сможет осуществлять какой-либо контроль над этим файлом.

Так как у файла есть владелец и ассоциированная с файлом группа пользователей, с помощью команды **id** пользователь может вывести на экран свои идентификаторы пользователя и группы и определить, какие права доступа к тому или иному файлу, хранящемуся в системе, он имеет.

Всеми файлами, представленными на слайде, владеет пользователь **user3**, и к ним могут обращаться пользователи из группы **class**. Кроме того, пользователь **user3** разрешает всем другим пользователям системы обращаться к этим файлам.

6–3. СЛАЙД: Типы доступа

Типы доступа

Для каждого файла и каталога определены три типа доступа:

Read (чтение)

файлы:	содержимое можно просмотреть
каталоги:	содержимое можно просмотреть

Write (запись)

файлы:	содержимое можно изменить
каталоги:	содержимое можно изменить

Execute (исполнение)

файлы:	могут быть использованы как команды
каталоги:	могут стать текущим рабочим каталогом

Пояснения

Для каждого файла или каталога определены три типа доступа: **read** (чтение), **write** (запись), и **execute** (исполнение).

Различные команды системы UNIX требуют определенных прав доступа к программе или файлу. Например, команда **cat имя_файла** требует право *read* (чтения) указанного в аргументе файла, так как команда **cat** должна иметь возможность прочесть содержимое этого файла, чтобы вывести его на экран. Аналогично, команде **ls** требуется право *read* (чтения) каталога, чтобы вывести его содержимое на экран.

Следует обратить внимание на то, что смысл доступа зависит от того, к чему именно – файлу или каталогу – осуществляется доступ. Например, право *write* (запись) доступа к файлу подразумевает, что содержимое файла может быть изменено. Отмена права доступа *write* запрещает пользователям изменить содержимое файла. Отсутствие этого права доступа не предохраняет файл от удаления. Право доступа *write* для каталога управляет тем, можно или нельзя изменить содержимое каталога. Если для каталога не установлено право доступа *write*, его содержимое не может быть изменено, и поэтому нельзя удалять или переименовывать файлы из этого каталога или добавлять новые файлы в этот каталог.

ПРИМЕЧАНИЕ: Для того чтобы выполнить файл как программу, этот файл должен одновременно иметь два права доступа: *read* (чтение) и *execute* (исполнение).

6-4. СЛАЙД: Права доступа

Права доступа

Права доступа к файлу выводятся на экран командой `ls -l`

```
$ ls -l
```

-	rw-	r--	r--	1	user3	class	37	Jul	24	11:06	f1
-	rwX	r-x	r-x	1	user3	class	37	Jul	24	11:08	f2
d	rwX	r-x	r-x	2	user3	class	1024	Jul	24	12:03	memo

права владельца (user) →
 права группы (group) →
 права остальных пользователей (other) →

владелец файла →
 группа файла →

Пояснения

Возможности доступа к файлу для пользователя определяются его идентификатором *пользователя*, идентификатором *группы* и правами доступа, ассоциированными с файлом. Права доступа к файлу определяются в **режиме (mode)** файла. Режим файла представляет собой поле длиной девять символов, в котором определены права доступа к файлу для владельца файла, группы, к которой относится файл, и для всех остальных пользователей системы.

Примеры

По отношению к файлам, представленным на слайде, права доступа определяются следующим образом:

Имя файла	Кто связан с файлом	Вид доступа	Разрешенные действия
f1	user3 (владелец)	чтение, запись	Просмотр и изменение содержимого файла
	пользователи-члены группы class	чтение	Просмотр содержимого файла
	остальные пользователи	чтение	Просмотр содержимого файла
f2	user3 (владелец)	чтение, запись, исполнение	Просмотр и изменение содержимого файла, выполнение в качестве программы
	пользователи-члены группы class	чтение, исполнение	Просмотр содержимого файла, выполнение в качестве программы
	остальные пользователи	чтение, исполнение	Просмотр содержимого файла, выполнение в качестве программы
memo	user3 (владелец)	чтение, запись, исполнение	Просмотр и изменение содержимого каталога memo , переход в каталог memo
	пользователи-члены группы class	чтение, исполнение	Просмотр содержимого каталога memo , переход в каталог memo
	остальные пользователи	чтение, исполнение	Просмотр содержимого каталога memo , переход в каталог memo

6–5. СЛАЙД: **chmod** – изменение прав доступа к файлу

chmod – изменение прав доступа к файлу

Синтаксис:

chmod *список_прав* *файл* ... Изменить права доступа к файлу (-ам)

список_прав [*кто* [*операция*] *право_доступа*] [, ...]

кто владделец (**u**ser), группа (**g**roup), остальные (**o**ther)
или все (**a**ll)

операция + (добавить), - (отменить), = (установить)

право_доступа чтение (**r**ead), запись (**w**rite), исполнение (**e**xecute)

Пример:

Исходные права: **режим** **владделец** **группа** **остальные**
 rw-r--r-- **rw-** **r--** **r-**

\$ **chmod u+x,g+x,o+x file** или \$ **chmod +x file**

Новые права: **режим** **владделец** **группа** **остальные**
 rwxr-xr-x **rwX** **r-x** **r-x**

Пояснения

Команда **chmod** используется для изменения прав доступа к файлу или каталогу. Только пользователь – владделец файла (или суперпользователь **root** – администратор системы) может изменить права доступа. Следовательно, в системе UNIX управление доступом к файлу является прерогативой его владельца, а не менеджера системы.

Чтобы защитить файл от удаления или изменения, каталог, в котором находится файл, и сам файл *не должны* иметь права записи. Право записи в файл позволяет пользователям изменять (или перезаписывать) содержимое файла, а право записи в каталог позволяет пользователю удалять файл.

Команда **chmod** поддерживает символьный способ задания прав доступа к файлу.

Права доступа, которые надо модифицировать, можно указать следующим образом:

r право чтения
w право записи
x право исполнения

Как именно модифицируются права доступа, можно указать следующим образом:

+	добавить указанное право доступа
-	удалить указанное право доступа
=	установить указанное право доступа

Можно также указать, для какой категории пользователей (кому именно) следует модифицировать права доступа:

u	user (пользователь-владелец файла)
g	group (группа пользователей, ассоциированная с файлом)
o	other (все остальные пользователи системы)
a	all (вообще все пользователи системы)
(пусто)	указанная модификация права доступа выполняется для всех категорий пользователей

ПРИМЕЧАНИЕ: Чтобы удалить все права доступа к файлу, нужно выполнить следующую команду:

chmod = *имя_файла*

Примеры

```
$ ls -l f1
-rw-r--r-- 1 user3 class 37      Jul 24 11:06 f1
$ chmod g=rw,o=f1
$ ls -l f1
-rw-rw---- 1 user3 class 37      Jul 24 11:06 f1
$ ls -l f2
-rw-rw-rw- 1 user3 class 37      Jul 24 11:08 f2
$ chmod u+x,g=rx,o=rw f2
$ ls -l f2
-rwxr-x--- 1 user3 class 37      Jul 24 11:08 f2
```

Выполнив команду **mesg n**, можно запретить другим пользователям вывод сообщений на свой терминал. Каждому терминалу соответствует специальный файл устройства, отвечающий за связь пользователя с компьютером. В рассматриваемом примере файл **/dev/tty0p1** представляет собой такой файл устройства.

```
$ ls -l /dev/tty0p1
crw--w--w- 1 bin    bin    58 0x000003 Feb 15 11:34 /dev/tty0p3
$ mesg n
$ ls -l /dev/tty0p1
crw----- 1 bin    bin    58 0x000003 Feb 15 11:34 /dev/tty0p3
```

Даже если пользователь запретил вывод на свой терминал сообщений от других пользователей, администратор системы может послать сообщение на терминал, и оно будет выведено.

Команда **chmod** использует и числовое (осьмеричное) представление прав доступа, ассоциированных с файлом. Такой способ задания прав доступа считается устаревшим, но, тем не менее, используется и до сих пор.

1. Чтобы изменить права доступа, нужно каждую группу прав представить соответствующим числом. Права доступа определяются для владельца файла, группы и всех остальных. Следует помнить, что каждый тип доступа задается определенным числом:
 - read = 4
 - write = 2
 - execute = 1
2. Разрешается складывать числа, ассоциированные с типами доступа для одной группы прав
3. Вычисленные права доступа для всех трех групп (три числа) собираются вместе, образуя устанавливаемые права доступа к файлу. Полученное число и будет аргументом команды **chmod**.

Например, надо установить права доступа **rw-** для владельца файла, **r--** для группы и **---** для остальных пользователей:

владелец	группа	остальные	
rw-	r--	---	<i>преобразование в числовые значения</i>
4+2+0	4+0+0	0+0+0	
6	4	0	

Команда **chmod** будет выглядеть следующим образом:

```
chmod 640 имя_файла
```

ПРИМЕЧАНИЕ: Чтобы удалить все права доступа к файлу, нужно выполнить следующую команду:

```
chmod 000 имя_файла
```

6–6. СЛАЙД: `umask` – маскирование прав доступа

`umask` — маскирование прав доступа

Синтаксис:

`umask [-S] [права]`

Устанавливаемая пользователем маска прав доступа, используемая при создании файла

Пример:

	владелец	группа	остальные
текущая маска для задания прав доступа:	<code>rw-</code>	<code>rw-</code>	<code>rw-</code>
требуемая маска для задания прав доступа:	<code>rw-</code>	<code>r--</code>	<code>---</code>

`$ umask g=r,o=`

Пояснения

Опция `-S` выводит текущее значение маски прав доступа, используемой при создании файлов, в символическом виде. Опция `-S` и символическая форма задания прав доступа не доступны в Bourne shell и C shell.

Опция `a-rwx` представляет собой сокращенную форму задания следующей маски: `u-rwx,g-rwx,o-rwx`. Обычно для вновь созданных файлов по умолчанию устанавливаются права доступа `rw-rw-rw-`, что означает, что любой пользователь системы может просматривать и модифицировать содержимое файла. Для вновь созданных каталогов по умолчанию устанавливаются права доступа `rxwxrwxrwx`, что означает, что любой пользователь может перемещаться в этот каталог и удалять из этого каталога.

Чтобы защитить свои файлы, создаваемые в течение сеанса работы в системе, следует использовать команду `umask`. Эта команда позволяет изменить используемые по умолчанию права доступа для любых *новых* файлов и каталогов, создаваемых пользователем. Возможно, наиболее важным правом доступа, которое следует отменить, является право записи для группы и остальных пользователей. Установленная маска прав доступа действует до тех пор, пока пользователь не выйдет из системы или пока не будет выполнена новая команда `umask`. Следует отметить, что команда `umask` не оказывает никакого влияния на уже существующие файлы.

6–7. СЛАЙД: **touch** – обновление временных меток файла

touch — обновление временных меток файла

Синтаксис:

touch [-амс] *файл...* Обновляет время последнего доступа и/или модификации файла

Примеры:

```
$ ll
-rw-r--r-- 1 karenk users 25936 Aug 24 09:53 firstfile
-rw-r--r-- 1 karenk users 10245 Aug 24 09:53 secondfile
$ touch newfile
$ ll
-rw-r--r-- 1 karenk users 25936 Aug 24 09:53 firstfile
-rw-r--r-- 1 karenk users 0 Aug 25 10:02 newfile
-rw-r--r-- 1 karenk users 10245 Aug 24 09:53 secondfile
$ touch secondfile
$ ll
-rw-r--r-- 1 karenk users 25936 Aug 24 09:53 firstfile
-rw-r--r-- 1 karenk users 0 Aug 25 10:02 newfile
-rw-r--r-- 1 karenk users 10245 Aug 25 10:05 secondfile
$
```

Пояснения

Команда **touch** позволяет создавать новый, пустой файл. Если указанный в команде файл уже существует, команда **touch** обновит временные метки файла. Эта команда не изменяет содержимое файла.

Команда **touch** имеет следующие опции:

- a *время*** устанавливает время последнего доступа к файлу, равное указанному значению
- m *время*** устанавливает время последней модификации файла, равное указанному значению
- t *время*** использует вместо текущего времени указанное значение
- c** если указанный в команде файл не существует, не создавать его

Время задается в виде: [[**ГГ**] **ГГ**] **ММ****ДД****ЧЧ****ММ** [**.СС**] (см. статью **touch (1)** *Справочника*).

Примеры

```
$ touch test_file1
$ ls -l test_file1
-rw-rw-rw- 1 user3 class 0 Jul 24 11:08 test_file1
$ umask a-rwx,u=rw,g=r (or umask 137)
$ umask -S (or umask)
u=rw,g=r,o= (or 137)
$ touch test_file2
$ ls -l test_file2
-rw-r----- 1 user3 class 0 Jul 24 11:10 test_file1
```

6–8. СЛАЙД: **chown** – изменение прав собственности файла

chown — изменение прав собственности файла

Синтаксис:

chown *владелец* [:*группа*] *файл* ... Изменяет для файла (ов)
идентификатор владельца
и, возможно, группы

Пример:

```
$ id
uid=303 (user3), gid=300 (class)
$ cp f1 /tmp/user2/f1
$ ls -l /tmp/user2/f1
-rw-r----- 1 user3 class 3967 Jan 24 13:13 f1
$ chown user2 /tmp/user2/f1
$ ls -l /tmp/user2/f1
-rw-r----- 1 user2 class 3967 Jan 24 13:13 f1
```

Только владелец файла (или администратор системы) могут изменить права владения файлом

Пояснения

Только владелец файла может управлять атрибутами файла и доступом к нему. Чтобы, в случае необходимости, передать права владения файлом другому пользователю системы, нужно использовать команду **chown**. Например, пользователь **user3** сделал для пользователя **user2** копию своего файла **f1**. Пользователь **user2** должен иметь право полного контроля над своей собственной копией, поэтому **user3** передает право владения файлом **/tmp/user2/f1** пользователю **user2**. Команда **chown** позволяет также изменить идентификатор группы одного или нескольких файлов на значение *группа*. Владелец (группа) файла может быть задан или числовым идентификатором пользователя (группы), или входным именем. Соответствующие значения можно найти в файле **/etc/passwd** (**/etc/group**).

ПРИМЕЧАНИЕ: После изменения права собственности файла только *новый владелец* файла или суперпользователь **root** могут изменять права собственности файла и права доступа к нему.

Владелец файла представляется идентификатором пользователя, известным системе. В файле **/etc/passwd** указаны идентификаторы всех пользователей системы.

Пример

В примере, приведенном на слайде, после того как пользователь **user3** передал права собственности на файл **/tmp/user2/f1** пользователю **user2**, **user3**, тем не менее, имеет право чтения этого файла, так как для этого файла установлено право чтения для всех пользователей, являющихся членами группы **class**.

Система Linux



В среде Linux политика безопасности осуществляется средствами, отличными от тех, которые используются в HP-UX.

По умолчанию, в среде Linux только администратор системы **root** может выполнять команду **chown**.

6–9. СЛАЙД: Команда **chgrp**

Команда **chgrp**

Синтаксис:

chgrp *новая_группа* *файл* ... Изменяет группу, ассоциированную с файлом. Только владелец файла (или **root**) может изменить группу файла

Пример:

```
$ id
uid=303 (user3), gid=300 (class)
$ ls -l f3
-rw-r----- 1 user3 class 3967 Jan 24 13:13 f3
$ chgrp class2 f3
$ ls -l f3
-rw-r----- 1 user3 class2 3967 Jan 24 13:13 f3
$ chown user2 f3
$ ls -l f3
-rw-r----- 1 user2 class2 3967 Jan 24 13:13 f3
$
```

Пояснения

Поле *group* в выводе команды **ls -l** указывает, какая группа пользователей ассоциирована с этим файлом. Эту группу можно изменить с помощью команды **chgrp**.

Значение **new_group** представляет идентификатор группы, известный системе. В файле **/etc/group** указаны все идентификаторы групп для всех пользователей системы.

Команда **chgrp** не будет выполнена, если указанная в ней новая группа не существует. Составом и наличием групп, а также распределением пользователей по группам управляет администратор системы.

ПРИМЕЧАНИЕ: Только *владелец* файла (или **root**) может изменить идентификатор группы, ассоциированной с файлом.

Пример

В примере, приведенном на слайде, пользователь **user3** изменил группу, ассоциированную с файлом **f1**, на **class2**. Это изменение не повлияло на права доступа к файлу для пользователя **user3**, так как он остался владельцем группы. Но после того как пользователь **user3** передал права владения файлом пользователю **user2**, **user3** не имеет никаких прав доступа к файлу, так как **user3** принадлежит группе пользователей **class**.

Система Linux



В среде Linux команда **chgrp** может быть выполнена только пользователем, который в данный момент принадлежит группе, ассоциированной с файлом. Поэтому эта команда будет выполняться по-другому, и не будет соответствовать тому примеру, который показан для системы HP-UX.



Одна из характеристик политики безопасности в системе Linux заключается в том, что все пользователи принадлежат к определенным рабочим группам, ассоциированным с самими пользователями. Например, если создается пользователь с именем **alun**, этот пользователь автоматически зачисляется в определяемую пользователем рабочую группу с тем же именем **alun**. Администратор системы может включить данного пользователя и в другие рабочие группы, чтобы разрешить данному пользователю доступ к необходимым файлам и каталогам.

6–10. СЛАЙД: **su** – переключение идентификатора пользователя

su — переключение идентификатора ПОЛЬЗОВАТЕЛЯ

Синтаксис:

su [*имя_пользователя*] Изменяет ранее назначенные идентификаторы
пользователя и группы

Пример:

```
$ ls -l /usr/local/bin/class_setup
-rwxr-x--- 1 class_admin teacher 3967 Jan 24 13:13 class_setup
$ id
uid=303 (user3), gid=300 (class)
$ su class_admin
Password:
$ id
uid=400 (class_admin), gid=300 (class)
$ /usr/local/bin/class_setup
$  + 
log out of su session
$
```

Пояснения

Команда **su** позволяет пользователю в интерактивном режиме изменить свои идентификаторы пользователя (user ID) и группы (group ID). Название команды **su** является аббревиатурой для *switch user* или *set user ID*. Эта команда позволяет пользователю, не выходя из системы, начать сеанс работы в системе от имени нового пользователя и гарантирует доступ ко всем файлам, владельцем которых является этот новый пользователь. Поэтому в целях безопасности команда **su**, прежде чем действительно переключить статус пользователя, потребует от него ввод пароля.

Команда **su** без аргументов переключает статус пользователя на **root** (администратор системы). Пользователь **root** известен часто как *суперпользователь* (*superuser*), так как этому входному имени разрешен доступ ко всему в системе. По этой причине часто название команды **su** воспринимается как аббревиатура для *superuser*. Безусловно, в данном случае потребуется ввод пароля пользователя **root**.

ПРИМЕЧАНИЕ: Для того чтобы вернуться к исходному идентификатору пользователя, не следует использовать снова команду **su**. Чтобы завершить сеанс работы в системе, начатый командой **su**, нужно выполнить команду **exit**.

Пример

Рассмотрим пример, приведенный на слайде. Пользователь **user3** не имеет какого-либо права доступа к программе **/usr/local/bin/class_setup**, так как **user3** не является членом группы **teacher**. Тем не менее, **user3** сможет получить права доступа к этой программе, выполнив команду **su class_admin**. Работая от имени пользователя **class_admin**, **user3** может запустить на исполнение программу **class_setup**. Когда выполнение программы закончится, **user3** восстанавливает свой исходный статус пользователя, завершая сеанс работы, начатый командой **su**.

```
su - username
```

В системе имеются специальные конфигурационные файлы, которые настраивают для пользователя его сеанс работы в системе. Когда выполняется команда **su имя_пользователя**, характеристики сеанса работы в системе остаются теми, которые соответствуют исходной идентификации пользователя. Если нужно, чтобы характеристики сеанса работы соответствовали тем, которые устанавливаются для нового пользователя, в команде **su** нужно использовать опцию (-): **su - имя_пользователя**.

6–11. СЛАЙД: Команда **newgrp**

Команда **newgrp**

Синтаксис:

newgrp [*имя_группы*] Изменяет идентификатор группы

Пример:

```
$ ls -l /usr/local/bin/class_setup
-rwxr-x--- 1 class_admin teacher 3967 Jan 24 13:13 class_setup
$ id
uid=303 (user3) gid=300 (class)
$ newgrp teacher
$ id
uid=303 (user3) gid=33 (teacher)
$ /usr/local/bin/class_setup
$ newgrp
Возврат к статусу входной группы
$ newgrp other
Sorry
$
```

Пояснения

Команда **newgrp** аналогична команде **su**. Команда **newgrp** позволяет изменить для пользователя идентификатор группы.

Администратор системы определяет, как именно пользователь может изменять свой идентификатор группы (т.е. определяет группы, в которые включен пользователь). Просмотрев файл **/etc/group**, пользователь может определить, членом каких групп он является (соответственно, во время сеанса работы в системе он может изменять свой идентификатор группы на эти значения). Если пользователь не включен в состав какой-то определенной группы, то при попытке изменить идентификатор группы на это значение он получит сообщение **Sorry**.

Так как команда **newgrp** не начинается новый сеанс работы в системе, для того чтобы вернуться к исходной группе, следует снова использовать команду **newgrp**.

Пример

Рассмотрим пример, приведенный на слайде. Пользователь **user3** не имеет права доступа к программе **/usr/local/bin/class_setup**, так как **user3** изначально является членом группы **class**. Чтобы продолжить работу в системе членом группы **teacher**, **user3** может выполнить команду **newgrp teacher**. Это возможно, так как администратор системы предоставил пользователю **user3** право перехода в эту группу. Теперь **user3** может выполнить программу, так как это разрешено всем пользователям – членам группы

teacher, но **user3** не может изменить содержимое этой программы. Только пользователь **class_admin** может изменять программу **class_setup**. Чтобы вернуться к исходной группе, пользователь **user3** должен выполнить команду **newgrp** без аргументов.

Пример файла **/etc/group**

```
teacher::33:class_admin,user3
```

```
class::300:user1,user2,user3,user4,user5,user6,class_admin
```

6–12. СЛАЙД: Списки управления доступом ACL

Списки управления доступом ACL

Синтаксис:

lsacl *файл* Выводит ACL для указанного файла
chacl *ACL файл* Изменяет ACL для указанного файла

Примеры:

```
$ lsacl funfile
(user3.%,rw-)(%.class,r--)(%.%,r--) funfile
$ chacl "user2.class=rw,%.%-r" funfile
$ lsacl funfile
(user2.class,rw-)(user3.%,rw-)(%.class,r--)(%.%,---) funfile
$ chacl -d "user2.class" funfile
$ ll funfile
rw-r----- 1 user3 class 3081 May 28 16:12 funfile
$ lsacl funfile
(user3.%,rw-)(%.class,r--)(%.%,---) funfile
```

Примечание: ACL не поддерживаются некоторыми версиями JFS
файловых систем

Пояснения

Списки управления доступом (Access Control Lists – ACL) представляют собой принудительный механизм дискреционного управления доступом (discretionary access control – DAC). Этот механизм позволяет более избирательно назначать права доступа к файлу по сравнению с традиционными механизмами, принятыми в системе UNIX. ACL представляет собой список комбинаций идентификаторов пользователя (user ID) и группы (group ID) вместе с правами доступа, установленными для данной комбинации пользователь/группа: (**пользователь . группа , права_доступа**) .

Уровни управления доступом

Существуют четыре основных уровня задания спецификаций, которые могут определять права доступа пользователя к файлу:

(u.g,rwx)	конкретный пользователь, конкретная группа
(u.%,rwx)	конкретный пользователь, любая группа
(%.g,rwx)	любой пользователь, конкретная группа
(%.%,rwx)	любой пользователь, любая группа

Для каждого файла можно задать до 13 различных наборов прав доступа. Если есть несколько похожих элементов, определяющих права доступа к файлу, более точно заданный доступ является более приоритетным по сравнению с остальными менее точно заданными значениями.

Пример

```
$ lsacl funfile
(user2.class,rw-) (user3.%,rw-) (user2.%,r--) (%.class,r--) (%.%,---) funfile
```

Пользователи будут иметь следующие возможности доступа к файлу **funfile**:

Пользователь	Группа	Доступ к файлу
user3	любая	чтение, запись
user2	class	чтение, запись
user2	любая группа, кроме class	чтение
любой пользователь, кроме user2 и user3	class	чтение
любой пользователь, кроме user2 и user3	любая группа, кроме class	нет доступа

Изменение списка прав доступа

Для модификации или удаления существующего списка прав доступа используется команда **chacl**. Опция **-d** позволяет удалить существующие назначения ACL.

Примеры

```
$ lsacl funfile
(user3.%,rw-) (%.class,r--) (%.%,r--) funfile
```

Приведенная ниже команда добавляет ACL для пользователя **user2**, работающего в группе **class**, предоставляя ему право чтения и записи, а также удаляет право чтения для всех остальных пользователей и групп (%.%):

```
$ chacl "user2.class=rw,%.%-r" funfile
$ lsacl funfile
(user2.class,rw-) (user3.%,rw-) (%.class,r--) (%.%,---) funfile
```

Тот же результат можно получить, выполнив следующую команду:

```
$ chacl "(user2.class,rw) (%.%, -r)" funfile
```

Приведенная ниже команда удаляет ACL для пользователя **user2**, работающего в группе **class**:

```
$ chacl -d "user2.class" funfile
$ lsacl funfile
(user3.%,rw-) (%.class,r--) (%.%,---) funfile
```

ПРИМЕЧАНИЕ: Изменение прав доступа к файлу с помощью команды **chmod** удаляет все ACL для этого файла.

ПРИМЕЧАНИЕ: ACL поддерживаются только файловой системой NFS, которая в HP-UX 11.00 не является стандартной файловой системой, устанавливаемой по умолчанию.

Система Linux



ПРИМЕЧАНИЕ: Никакая файловая система в ОС Linux не поддерживает ACL.

6–13. СЛАЙД: Доступ к файлам: заключение

Доступ к файлам: заключение

Права доступа	Определяют, кто (пользователь, группа, остальные) имеют доступ к файлу, и какой именно (чтение, запись, исполнение)
chmod	Изменение прав доступа к файлу
umask	Определение прав доступа по умолчанию, устанавливаемых при создании новых файлов
chown	Изменение пользователя-владельца файла
chgrp	Изменение группы, ассоциированной с файлом
su	Переключение идентификатора пользователя
newgrp	Переключение идентификатора группы

Пояснения

Что нужно помнить в отношении прав доступа к файлу:

- Для того чтобы получить доступ к файлу, заданному полным маршрутным именем, все каталоги, указанные в маршруте, должны иметь право исполнения.
- Чтобы защитить файл, нужно удалить право записи для этого файла и для каталога, в котором этот файл находится.
- Только владелец файла (или **root**) может изменить права доступа к файлу (**chmod**), права собственности для файла (**chown**) или группу, ассоциированную с файлом (**chgrp**).

6–14. УПРАЖНЕНИЯ: Доступ к файлам

Указания

Упражнения разделены на четыре раздела. Выполните команды, необходимые для решения представленных упражнений, и ответьте на поставленные вопросы. Возможно, будет недостаточно времени для выполнения всех упражнений.

Права доступа к файлам

1. Найдите в своем *HOME* каталоге файл **mod5.1**. Кто имеет права доступа к этому файлу? Какие? Можете ли вы вывести на экран содержимое файла **mod5.1**?
2. Установите для файла **mod5.1** следующие права доступа: **-w-----**. Сможете ли вы теперь вывести на экран содержимое файла **mod5.1**?
3. Установите для файла **mod5.1** следующие права доступа: **rw-----**. Сможете ли вы теперь вывести на экран содержимое файла **mod5.1**? Сможет ли другой пользователь — член вашей группы вывести на экран содержимое файла **mod5.1**?
4. Какие права доступа к файлу **mod5.1** нужно установить, чтобы другой пользователь — член вашей группы *смог* прочитать содержимое файла?

5. Создайте копию файла **mod5.1** с именем **mod5.2**. Удалите для файла **mod5.2** право записи. Сможете ли вы удалить этот файл? Что нужно сделать, чтобы защитить этот файл от удаления?

6. Кто является владельцем файла **root_file**, находящегося в вашем *HOME* каталоге? С какой группой ассоциирован этот файл? Кто имеет право изменить для этого файла владельца или группу? Какими правами доступа к этому файлу обладаете вы?

7. Если вы хотите внести изменения в содержимое файла **root_file**, что вы для этого сделаете?

8. Выполните команду **mesg y**. Теперь выполните команду **tty** и посмотрите, какой специальный файл устройства ассоциирован с вашим терминалом. Каковы права доступа к этому файлу? Кто является владельцем этого файла? Выполните команду **mesg n**. Какие теперь права доступа к этому файлу? Что в действительности делает команда **mesg**?

Права доступа к каталогам

1. В своем *HOME* каталоге создайте новый каталог **mod5.dir**. Скопируйте в каталог **mod5.dir** файл **mod5.1**. Выведите содержимое нового каталога. Какие права доступа установлены для каталога **mod5.dir**? (ПОДСКАЗКА: `ls -ld mod5.dir`)
2. Установите для каталога **mod5.dir** следующие права доступа: **rw-----**. Сможете ли вы перейти в каталог **mod5.dir**? Сможете ли вы вывести содержимое каталога **mod5.dir**? Сможете ли вы получить доступ к файлу **mod5.1**, находящемуся в каталоге **mod5.dir**?
3. Установите для каталога **mod5.dir** следующие права доступа: **-wx-----**. Сможете ли вы вывести содержимое каталога **mod5.dir**? Сможете ли вы вывести содержимое файла **mod5.1**, находящегося в каталоге **mod5.dir**? Сможете ли вы перейти в каталог **mod5.dir**?
4. Сможет ли другой пользователь скопировать файл в ваш *HOME* каталог? Как вывести права доступа, установленные для вашего *HOME* каталога?

5. Скопируйте файл **mod5.1** из своего *HOME* каталога в каталог **/usr/bin**. Возникли ли какие-либо проблемы? Какие права доступа установлены для каталога **/usr/bin**?
6. Сможете ли вы скопировать файл **/usr/bin/date** в свой *HOME* каталог?

Изменение для файла владельца и группы



Пользователям системы Linux не разрешается использовать команду **chown**. Поэтому упражнения со второго по четвертый пользователям системы Linux выполнять не следует.

1. В вашем *HOME* каталоге найдите файл, имя которого совпадает с вашим входным именем. Какие права доступа к этому файлу вы имеете? К какой группе принадлежит ваш коллега? Какие права доступа к этому файлу имеет ваш коллега?
2. Продолжая работать с этим файлом, передайте право владения этим файлом вашему коллеге. Есть ли у вас теперь какие-либо права доступа к этому файлу? Попробуйте сделать копию этого файла. Сможете ли вы вернуть себе право владения этим файлом?
3. Создайте копию файла **mod5.1** с именем **mod5.3**. Удалите для файла **mod5.3** все права доступа. Сможете ли вы передать права владения этим файлом своему коллеге?

4. Создайте копию файла **mod5.1** с именем **mod5.4**. Установите для этого файла следующие права доступа: **rw-r-----**. Измените группу, ассоциированную с этим файлом, на **class2**. Передайте права владения этим файлом пользователю **root**. Сможете ли вы вывести содержимое файла **mod5.4**?
5. Перейдите в группу **class2**. Сможете ли вы теперь вывести содержимое файла **mod5.4**? Восстановите свою исходную группу. (ПОДСКАЗКА: используйте команду **id**, чтобы вывести свои идентификаторы пользователя и группы.)

Права доступа для вновь создаваемых файлов

1. Какие права доступа устанавливаются для вновь создаваемых файлов? ПОДСКАЗКА: создайте новый файл с помощью редактора или команды **touch** или же скопируйте какой-либо существующий файл. Проверьте, какие права доступа установлены для вновь созданного файла. Что можно сказать относительно прав доступа, устанавливаемых при создании нового каталога? Какова текущая маска, используемая для установки прав доступа при создании новых файлов?
2. Как нужно модифицировать назначаемые по умолчанию права доступа при создании новых файлов, чтобы исключить возможность модификации файлов и каталогов

Модуль 6
Доступ к файлам

другими пользователями — членами группы и всеми остальными пользователями системы? Проверьте результаты, создав еще один новый файл и еще один новый каталог.

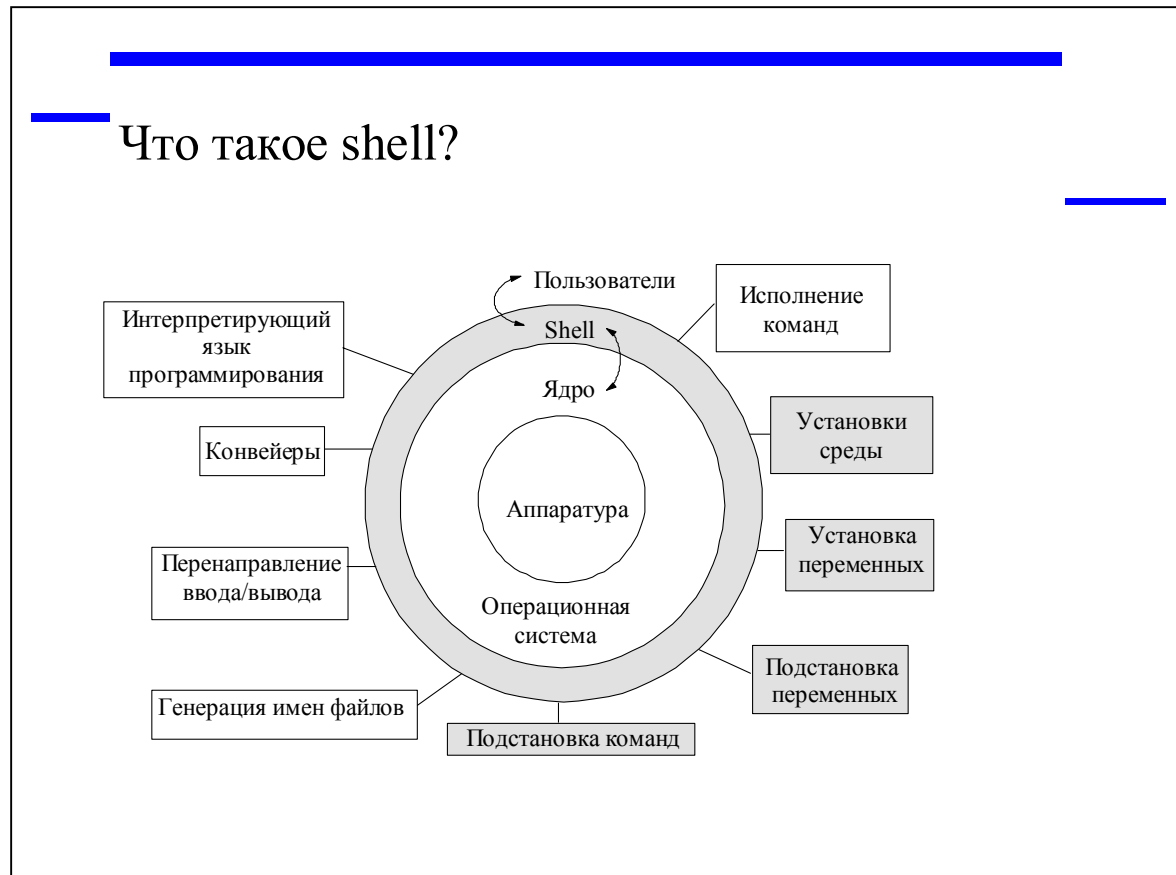
Модуль 7 — Основы shell

Цели

Изучив этот модуль, вы сможете:

- Описывать назначение shell'a
- Описывать, что происходит, когда пользователь входит в систему
- Описывать переменные среды пользователя и их назначение
- Устанавливать и модифицировать переменные shell'a
- Понимать и изменять специфические переменные среды, такие как *PATH* и *TERM*
- Настраивать среду пользователя в соответствии с потребностями конкретного приложения

7-1. СЛАЙД: Что такое shell?



Пояснения

Shell представляет собой интерактивную программу, которая выполняет функции интерпретатора командных строк. Shell отделен от операционной системы. Такое проектное решение предоставляет пользователю дополнительную гибкость с точки зрения выбора интерфейса, наилучшим образом отвечающего его потребностям. Назначение shell'a заключается в том, чтобы предоставить пользователю возможность ввести нужные ему команды, выполнить ряд функций и передать интерпретированную команду в операционную систему (ядро) для исполнения.

В этом модуле рассматриваются интерактивные возможности, предоставляемые POSIX shell'ом. Кроме собственно исполнения команд, POSIX shell выполняет и другие интерактивные функции. Многие из этих функций выполняются *перед* непосредственным исполнением команды.

Функциональные возможности shell'a можно охарактеризовать следующим образом:

- Shell осуществляет поиск указанной команды и выполняет ассоциированную с ней программу.
- Shell подставляет значения своих переменных. Он также выполняет подстановку результатов выполнения команд.
- Shell формирует имена файлов, используя символы генерации имен файлов.

- Shell выполняет перенаправление ввода/вывода и реализует конвейеры.
- Shell обеспечивает интерфейс интерпретирующего языка программирования, включающий в себя возможности проверки условий, реализации ветвления и циклов.

Как только пользователь входит в систему UNIX, shell сначала устанавливает определенные характеристики для сеанса работы с системой, а затем выводит на экран приглашение. POSIX, Bourne и Korn shell'ы в качестве приглашения по умолчанию используют символ **\$**. C shell в качестве приглашения по умолчанию использует символ **%**.

7–2. СЛАЙД: Обычно используемые shell'ы

Обычно используемые shell'ы

<code>/usr/bin/sh</code>	POSIX shell
<code>/usr/bin/ksh</code>	Korn shell
<code>/usr/old/bin/sh</code>	Bourne shell
<code>/usr/bin/csh</code>	C shell
<code>/usr/bin/keysh</code>	Контекстно-зависимый shell с программируемыми клавишами
<code>/usr/bin/rksh</code>	Ограниченный Korn shell
<code>/usr/bin/rsh</code>	Ограниченный Bourne shell

Пояснения

POSIX shell представляет собой POSIX-совместимый командный язык программирования и интерпретатор команд, размещенный в файле `/usr/bin/sh`. Он может выполнять команды, которые вводятся с терминала или читаются из файла. Shell удовлетворяет текущим стандартам POSIX, действовавшим в то время, когда была выпущена данная версия системы HP-UX, и во многом сходен с Korn shell'ом. Он имеет механизм хранения предыстории команд, поддерживает управление заданиями, а также имеет много других полезных возможностей.

Korn shell представляет собой командный язык программирования и интерпретатор команд, размещенный в файле `/usr/bin/ksh`. Он может выполнять команды, читаемые с терминала или из файла. Как и POSIX shell, он имеет механизм хранения предыстории команд, поддерживает управление заданиями и предоставляет много других полезных возможностей. Korn shell разработан Давидом Корном (David Korn) из AT&T Bell Labs.

Bourne shell представляет собой командный язык программирования и интерпретатор команд, размещенный в файле `/usr/old/bin/sh`. Он может выполнять команды, читаемые с терминала или из файла. В этом shell'е отсутствуют многие возможности, предоставляемые POSIX и Korn shell'ами. Автором Bourne shell'а является Стефан Р. Борн (Stephen R. Bourne). Этот shell является первым shell'ом, доступным в версиях системы UNIX компании AT&T.

C shell представляет собой интерпретатор командного языка, который использует буфер для хранения предыстории команд, имеет C-подобный синтаксис и средства управления заданиями. Он был разработан Вильямом Джоом (William Joy) из Калифорнийского университета в Беркли (University of California, Berkeley).

rsh и **rksh** – это ограниченные версии Bourne и Korn shell'ов, соответственно. Ограниченный shell устанавливает входное имя и среду пользователя, возможности которой в большей степени управляемы (ограничены) по сравнению с нормальными версиями shell'ов. Ограниченный shell во многих случаях действует так же, как и обычный shell, но имеет некоторые исключения. Пользователь, использующий ограниченный shell, не может:

- изменять каталог,
- переустанавливать значение переменной среды PATH,
- использовать символ / в маршрутном имени файла,
- выполнять перенаправление вывода.

keysh является расширением стандартного Korn shell'а. Он использует связанные с программируемыми клавишами иерархические меню и контекстно-зависимую помощь, облегчающие пользователю процесс построения командных строк. **keysh** был разработан компаниями HP и AT&T.

Таблица 1. Сравнение характеристик shell'ов

Характеристика	Описание	POSIX	Bourne	Korn	C
Предыстория команд	Позволяет запоминать команды в некотором буфере и затем их модифицировать и повторно выполнять	Да	Нет	Да	Да
Редактирование строк	Позволяет модифицировать текущую или введенные ранее командные строки с помощью некоторого текстового редактора	Да	Нет	Да	Нет
Автоматическое завершение имен файлов	Позволяет автоматически завершать вводимые с клавиатуры имена файлов в командной строке	Да	Нет	Да	Да
Команда alias	Позволяет пользователю переименовывать команды, включать в имя команды необходимые опции или использовать сокращения для длинных командных строк	Да	Нет	Да	Да
Ограниченные shell'ы	Обеспечивает средство защиты, предоставляющее возможность создания управляемой среды пользователя с ограниченными возможностями	Да	Да	Да	Нет
Управление заданиями	Предоставляет средства для трассировки и организации доступа к процессам, исполняющимся в фоновом режиме	Да	Нет	Да	Да

Система Linux

В системе Linux доступны следующие разновидности shell'ов:



/bin/ash	A shell. Аналогичен Bourne shell'у.
/bin/bash	GNU B ourne- A gain shell – используется главным образом администраторами системы
/bin/ksh	Общедоступная версия K orn shell'a
/bin/csh	C shell
/bin/sh	Ссылка на bash shell
/bin/tcsh	Улучшенная версия C shell
/bin/zsh	Z shell

7–3. СЛАЙД: Характеристики POSIX shell

Характеристики POSIX shell

- Shell – интерфейс пользователя с некоторыми дополнительными возможностями:
 - установка псевдонимов команд
 - автоматическое завершение имен файлов
 - механизм предыстории команд
 - повторный вызов и редактирование командной строки
 - управление заданиями
 - расширенные возможности команды **cd**
 - расширенные возможности программирования

Пояснения

POSIX shell – это один из shell'ов, поставляемых вместе с системой UNIX. Этот shell поддерживает многие возможности, которые имеются в Korn shell'е, но отсутствуют в Bourne shell'е. Даже если пользователь не использует все расширенные возможности shell'а, он оценит очень удобный интерфейс пользователя POSIX shell'а. Здесь представлены некоторые возможности POSIX shell'а:

- Механизм предыстории команд
- Повторный вызов и редактирование командной строки
- Управление заданиями
- Автоматическое завершение имен файлов
- Установка псевдонимов команд
- Расширенные возможности команды **cd**
- Расширенные возможности программирования

7–4. СЛАЙД: Установка псевдонимов команд

Установка псевдонимов команд

Синтаксис:

```
alias [имя[=строка]]
```

Примеры:

```
$ alias dir=ls
$ alias mroe=more
$ alias mstat=/home/tricia/projects/micron/status
$ alias laser="lp -dlaser"
$ laser fileX
request id is laser-234 (1 file)
$ alias          Выводит установленные псевдонимы команд
$ alias mroe     Выводит значение псевдонима mroe
mroe=more
```

Пояснения

Псевдоним (alias) – это новое имя команды. Переименование команд – это метод, который дает возможность сокращать длинные командные строки, создавать новые команды или заставлять стандартные команды выполняться по-другому благодаря замене исходной команды новой, более короткой, называемой псевдонимом. В качестве псевдонима можно использовать букву или короткое слово. Например, многие пользователи часто используют команду **ps -ef**. Не будет ли проще вместо этой команды вводить с клавиатуры просто **psf**?

Пользователь может создать нужные ему псевдонимы с помощью команды **alias**:

```
$ alias имя=строка
```

Здесь *имя* – это имя, которое используется в качестве псевдонима, а *строка* – это команда или строка символов, которые ставятся в соответствие псевдониму (определяют его значение). Если в строке встречаются пробелы, всю строку надо заключить в кавычки. Псевдонимы позволяют уменьшить объем ввода, правильно толковать часто допускаемые опечатки или создавать новые команды.

При вводе псевдонимы выглядят так же, как и другие команды. Для пользователя абсолютно безразлично (прозрачно), выполняет он действительно команду UNIX или псевдоним, который ссылается на команду UNIX.

Прежде чем выполнить команду, shell заменит в командной строке имя псевдонима его значением, а затем выполнит полученную в результате командную строку. Определенный в интерактивном режиме псевдоним будет доступен для использования до момента завершения сеанса работы в системе.

Некоторые пользователи находят эту возможность настолько удобной, что создают собственный интерфейс с системой UNIX, позволяющий распознавать команды, которые они обычно используют в других операционных средах (например, **alias dir=ls** или **alias copy='cp -i'**).

Псевдонимы также часто используются для сокращения полных маршрутных имен файлов.

Команда **alias** без аргументов выводит на экран все текущие определения псевдонимов.

Команда **alias ИМЯ** используется для вывода значения конкретного (указанного в команде) псевдонима.

Отмена псевдонима может быть выполнена командой **unalias**. Ее синтаксис:

```
unalias ИМЯ
```

Примеры

В одной командной строке можно использовать несколько псевдонимов, как показано в приведенном ниже примере:

```
$ alias go='cd '  
$ alias there=/home/user3/tree/ford/sports  
$ go there  
$ pwd  
/home/user3/tree/ford/sports
```

Чтобы в одной командной строке использовать несколько псевдонимов, в определении псевдонима последним символом строки должен быть обязательно пробел; в противном случае shell не узнает в следующем слове псевдоним.

7–5. СЛАЙД: Автоматическое завершение имен файлов

Автоматическое завершение имен файлов

```
$ more fra_ [ESC] [ESC]  
$ more frankenstein [Return]
```

.
.
.

```
$ more abc_ [ESC] [ESC]  
$ more abcdef_ [ESC] =
```

- 1) abcdefXlmnop
- 2) abcdefYlmnop

```
$ more abcdef_
```

Нажмите **a**, затем **X** или **Y** и [ESC] [ESC].

Будет автоматически завершен ввод соответствующего имени файла

Пояснения

Автоматическое завершение имени файла удобно применять в тех случаях, когда надо ввести длинное имя файла. Нужно ввести начальные символы имени файла, которые однозначно идентифицируют полное имя этого файла, и нажать [Esc] [Esc]. POSIX shell автоматически введет недостающие символы в имени файла. Если введенная строка встречается в именах нескольких файлов, POSIX shell не сможет определить требуемое имя файла, и пользователь должен как-то помочь shell'у. Терминал обычно издает звуковой сигнал, если обнаруживает такую конфликтную ситуацию при вводе имени файла.

Shell автоматически продолжает ввод имени файла до тех пор, пока он не обнаружит конфликтную ситуацию. В случае обнаружения конфликта пользователь может вывести на экран имеющиеся варианты имен файлов, нажав [Esc] = . После того как POSIX shell выведет доступные варианты, пользователь может с помощью редактора команд ввести в имени файла необходимые символы, однозначно определяющие требуемый файл, и вновь нажать [Esc] [Esc], чтобы shell завершил ввод имени файла.

Автоматическое завершение имени файла может быть использовано в любой позиции маршрутного имени файла. Например, ввод следующих символов:

```
$ cd tr [Esc] [Esc] do [Esc] [Esc] r [Esc] [Esc]
```

приведет к тому, что на экран будет выведена команда:


```
$ cd tree/dog.breeds/retriever
```

Автоматическое завершение имен файлов в системе Linux



Автоматическое завершение имен файлов в Bash shell аналогично тому, что используется в POSIX shell. С другой стороны, Linux Korn shell использует другие средства управления автоматическим завершением имени файла. Ниже приводится сравнительная характеристика возможностей автоматического завершения имен файлов в разных средах.

Описание	HP-UX POSIX	Linux bash	Linux Korn
Необходимо ли включить опцию, устанавливающую, что для редактирования командной строки используется редактор vi?	Нет	Нет	Да \$ set -o vi
Автоматическое завершение имени файла, если существует единственное соответствие	<code>Esc</code> <code>Esc</code>	<code>Esc</code> <code>Esc</code>	<code>Esc</code> <code>\</code>
Вывод всех соответствующих имен файлов, если имеется не однозначное соответствие	<code>Esc</code> <code>=</code>	<code>Esc</code> <code>=</code>	<code>Esc</code> <code>=</code>

7–6. СЛАЙД: Предыстория команд

Предыстория команд

- Shell сохраняет команды, вводимые пользователем, в файле предыстории
- Команда **history** выводит на экран последние 16 команд
- Ранее введенные команды можно вызвать на экран повторно, отредактировать и запустить вновь

Синтаксис:

history [-n | a z] Вывести на экран команды из файла предыстории

Примеры:

```
$ history -2      Вывести две последние команды
15 cd
16 more .profile
$ history 3 5      Вывести команды с номерами с 3 по 5
3 date
4 pwd
5 ls
```

Пояснения

POSIX shell обеспечивает хранение команд, вводимых пользователем, в специальном файле предыстории и позволяет повторно запускать на исполнение команды из этого файла. Файл предыстории поддерживается в течение всего сеанса работы в системе.

Команда **history** выводит на экран из файла предыстории последние 16 введенных пользователем команд. В начале каждой строки выводится ее номер. Этот номер можно указывать, чтобы повторно вывести на экран соответствующую команду из файла предыстории.

На экран можно вывести любое количество команд (не обязательно 16), если использовать следующую команду:

```
history -n
```

Здесь *n* указывает, сколько команд надо вывести на экран.

Можно указать диапазон номеров команд, которые надо вывести на экран:

```
history a z
```

Здесь значения *a* и *z* задают требуемый диапазон номеров команд.

Переменная среды *HISTSIZE* определяет максимальное количество последних введенных команд, которое можно запомнить в файле предыстории (по умолчанию значение *HISTSIZE* позволяет хранить 128 командных строк). Переменная среды *HISTFILE* задает имя текстового файла, в котором запоминаются вводимые команды (если этот файл не существует, при вводе первой команды он создается; по умолчанию значением *HISTFILE* является **.sh_history**).

После вывода на экран команд из файла предыстории, любую из команд можно вывести на экран повторно, отредактировать ее и/или запустить на исполнение.

7–7. СЛАЙД: Повторный запуск команды на исполнение

Повторный запуск команды на исполнение

- Чтобы повторно запустить на исполнение команду с номером **c**, следует выполнить команду **r c**

Пример:

```
$ history 3 5    Вывести на экран команды с номерами с 3 по 5
```

```
3 date
```

```
4 pwd
```

```
5 ls
```

```
$
```

```
$ r 4            Выполнить команду с номером 4
```

```
pwd
```

```
/home/kelley
```

Пояснения

Любую команду из файла предыстории можно выполнить повторно с помощью следующей команды:

```
r c
```

Здесь **c** – номер команды.

Вместо номера команды можно также ввести первый символ ее имени; тогда будет выполнена самая последняя из команд, хранящихся в файле предыстории, имя которой начинается с этого символа. Например:

```
$ history
1 date
2 cat file1
3 ls -l
$ r d
Mon Jul 4 10:03:13 1994
```

7–8. СЛАЙД: Повторный вызов команд

Повторный вызов команд

- Используется механизм команды **history**
- Должна быть установлена переменная среды *EDITOR*
EDITOR=vi
export EDITOR
 - В ответ на приглашение **\$** нажмите **Esc** и используйте обычные команды редактора **vi** для просмотра ранее введенных команд:
 - **k** просмотр буфера команд назад
 - **j** просмотр буфера команд вперед
 - **nG** вывод на экран команды с номером **n**
 - Нажмите **Return** для выполнения команды

Пояснения

Наиболее широко используемым редактором, входящим в состав shell UNIX, является редактор **vi**. Основные возможности этого редактора и будут использованы для иллюстрации принципов редактирования командной строки. Позднее возможности редактора **vi** будут рассмотрены более подробно.

Возможность shell'a запоминать команды в файле предыстории позволяет повторно запускать команды на исполнение без необходимости нового ввода этой команды. Этот механизм позволяет также редактировать введенные ранее командные строки, используя редактор **vi**. Эта возможность позволяет сократить большие объемы ввода. Если пользователь не имеет больших навыков в наборе текста, эта возможность позволяет ему также сэкономить время и усилия.

Для того чтобы использовать команды редактора **vi** для доступа к механизму предыстории POSIX shell, необходимо надлежащим образом установить переменную среды *EDITOR*. Увидеть значение этой переменной среды можно с помощью команды **env**:

```
$ env
. . .
EDITOR=/usr/bin/vi
. . .
```

Если эта переменная не установлена, чтобы установить ее, нужно выполнить следующие команды:

```
$ EDITOR=/usr/bin/vi
$ export EDITOR
```

Это позволяет сообщить POSIX shell'у, что для повторного вызова и редактирования ранее введенных команд будет использоваться редактор **vi**. Если нужна гарантия того, что при каждом входе в систему переменная **EDITOR** установлена надлежащим образом, эти команды следует поместить в файл **.profile**. Если переменная среды **EDITOR** не установлена, ее значением по умолчанию является **/usr/bin/ed**.

Чтобы повторно вызвать в командной строке экрана ранее введенную команду, сначала следует просто нажать **[Esc]**. При этом на экране не произойдет никаких изменений. Нажатие клавиши **[Esc]** всего лишь переводит POSIX shell в режим команд редактора **vi**. После этого становятся доступными многие обычные команды редактора **vi**. Например, нажатие клавиши **[k]** перемещает на одну команду назад по буферу команд. Если продолжать нажимать клавишу **[k]**, можно увидеть, как в командной строке экрана появляются ранее введенные команды, по одной на каждое нажатие клавиши. Нажатие клавиши **[j]** приводит к выводу на экран команд из буфера команд в обратном направлении. Когда в командной строке появится нужная команда, следует нажать клавишу **[Return]**.

Команда **history** позволяет увидеть последние 16 команд, введенные пользователем. Вместе с командной строкой на экран выводится и ее номер. Чтобы выполнить конкретную команду, следует последовательно нажать клавиши **[Esc] n G**, где **n** – номер командной строки, взятый из списка команд, выведенного командой **history**, а **G** – команда редактора **vi**, которая выполняет перемещение на указанную строку файла.

7–9. СЛАЙД: Редактирование командной строки

Редактирование командной строки

- Позволяет модифицировать текущую или введенную ранее команду
- Нажмите `[Esc]` для переключения в *командный режим*
- Вызовите на экран нужную команду:
 - нажимая `[k]`, пока команда не появится на экране, или
 - введя *номер команды*, а затем **G**

Пояснения

Часто возникает необходимость повторно вызвать на экран ранее исполнявшуюся команду и вновь запустить ее на исполнение, но перед запуском в команду необходимо внести некоторые незначительные изменения. Нажав клавишу `[Esc]` и затем `[k]`, можно увидеть на экране последнюю выполнявшуюся команду. Если известен номер команды, которую нужно вызвать на экран, можно ввести этот *номер команды*, а затем нажать клавишу **G**. Например, предположим, что команда **history** вывела на экран следующий список команд:

```
120      env
121      ls
122      cd
123      cd /tmp
124      pwd
125      history
```

Если нажать клавиши `[Esc]` `[k]` и затем ввести **122G**, на экране можно будет увидеть:

```
cd
```

Еще один способ найти в стеке команд нужную команду заключается в следующем: нужно последовательно нажать клавиши `[Esc]` `[k]`, а затем ввести */шаблон*. Например, для того чтобы найти в стеке команд последнюю выполнявшуюся команду **cd**, после нажатия клавиш `[Esc]` `[k]` можно ввести текст **/cd**. Если теперь еще раз ввести символ **/**, на экране

можно будет увидеть предыдущую выполнявшуюся команду **cd** (предшествующую выведенной на экран), и так далее. Если таким образом начата операция поиска по шаблону, ввод символа **n** позволит искать в стеке следующую команду, соответствующую шаблону (выполнявшуюся после той команды, которая была выведена на экран).

Теперь можно нажать клавишу **Return**, чтобы запустить найденную команду на исполнение, или использовать команды редактирования текста, представленные на следующем слайде. Если же выведенную на экран команду выполнять не нужно, следует нажать клавиши **Ctrl** + **c**, отменяющие команду.

7–10. СЛАЙД: Редактирование командной строки (продолжение)

Редактирование командной строки (продолжение)

- Позиционирование курсора
 - для перемещения вправо следует нажать `l` или `пробел`
 - для перемещения влево следует нажать `h` или `забой`
- *Не используйте клавиши со стрелками!*
- Для модификации текста
 - используйте `x` для удаления символа
 - используйте `i` или `a` для вставки или добавления символов
 - нажмите `Esc`, чтобы закончить ввод символов
- Нажмите `Return` для выполнения модифицированной команды

Пояснения

Как часто при вводе длинной командной строки вы вдруг обнаруживали, что допустили опечатку в самом начале этой строки? Такие ситуации встречаются достаточно часто, и все, что можно сделать в такой ситуации – это стереть (с помощью клавиши `забой`) строку вместе с ошибкой и потом вновь вводить полный текст командной строки.

POSIX shell позволяет исправлять ошибки и модифицировать необходимые фрагменты командной строки перед выполнением команды. Это делается также с помощью команд редактирования редактора `vi`.

Чтобы изменить командную строку, следует войти в режим редактирования редактора `vi`, нажав клавишу `Esc`. Изменения можно делать как для командных строк, которые вводятся в данный момент, так и для командных строк, которые вызываются на экран повторно с помощью клавиш `Esc` и `k`.

Находясь в режиме редактирования, для редактирования командной строки следует использовать обычные команды редактора `vi`. Например, команда `x` удаляет символ, `h` и `l` перемещают курсор по строке влево и вправо, `cw` позволяет изменить слово, `dw` удаляет слово, и так далее.

И работа со стеком команд, и редактирование командных строк обеспечиваются командами редактора `vi`. Достоинство такого проектного решения заключается в том, что, освоив

команды редактора **vi**, пользователь получает рабочий инструмент, необходимый для использования стека команд. Ему *не нужно* осваивать *другой* интерфейс и набор команд!

Для редактирования командной строки используются следующие команды редактора **vi**:

h , [Забой] , l , [Пробел] , w , b , \$	перемещение курсора
x , dw , p	удаление и вставка текста
r , R , cw	изменение текста
a , i	переход в <i>режим ввода</i> для добавления нового текста

Для того чтобы с помощью команд редактора **vi** получить доступ к стеку команд, необходимо установить переменную среды **EDITOR=/usr/bin/vi**. (Допускается использовать и другие редакторы – **gmacs** и **emacs**.)

Каждая командная строка рассматривается как мини-сеанс работы редактора **vi**. Начиная ввод каждой командной строки, пользователь находится в *режиме ввода*. Для того чтобы увидеть на экране введенную ранее команду, нужно выполнить команду редактора **vi**, которая перемещает курсор вверх. Прежде чем выполнять какие-либо команды редактора **vi**, необходимо, нажав клавишу **[Esc]**, перейти в *режим команд*.

Теперь можно выполнить команду **[k]** редактора **vi** для перемещения курсора вверх. Если продолжать выполнять команду **[k]**, можно перемещаться вверх через все ранее введенные команды. Когда на экране появится та команда, которую нужно выполнить, следует нажать клавишу **[Return]**, и команда будет выполнена. Эта команда также будет помещена в стек команд.

Главное преимущество POSIX shell'a заключается в том, что он позволяет изменять и ранее введенные команды, и текущую команду. Нет необходимости возвращаться (стирая введенные позже символы) в точку, где необходимы изменения, или начинать ввод команды сначала.

Эта возможность особенно полезна, когда вводятся длинные командные строки и в них имеются опечатки, или надо изменить аргументы команды. Когда рассматриваемые здесь возможности отсутствовали, в таких случаях надо было или вводить всю командную строку заново, или с помощью клавиши **[Backspace]** стирать введенные символы до места коррекции и затем вводить заново нужный текст.

Используя предоставляемые POSIX shell'ом возможности редактирования строк, можно вывести на экран ранее введенную команду и, прежде чем ее выполнить, внести в нее с помощью соответствующих команд редактора **vi** необходимые изменения. Эти изменения могут быть и простыми (например, изменить один символ), и достаточно сложными (например, изменить весь список аргументов командной строки).

Пример

```
$ cp /usr/lib/X11/app-defaults
Usage: cp f1 f2
       cp [-r] f1 ... fn d1
```

В приведенном выше примере предполагалось, что будет выполняться команда **cd**, а не **cp**. POSIX shell позволяет выполнить требуемую команду, не вводя ее заново. Следует просто нажать клавишу **[Esc]** и затем **[k]**, и командная строка будет вновь выведена на экран. Теперь

нужно нажать клавишу **l**, чтобы переместить курсор на место буквы **p** в имени команды **cp** и с помощью команды **r** заменить букву **d** на **p**. Теперь команда будет выглядеть так:

```
$ cd /usr/lib/X11/app-defaults
```

Теперь можно нажать клавишу **Return**, и команда **cd** будет выполнена.

Если при редактировании строки возникли какие-либо проблемы и надо начать редактирование сначала, для отмены редактирования нужно нажать клавишу **Break**. В ответ на экран будет выведено обычное приглашение shell'a, так что все можно будет начать сначала.

Не используйте при редактировании командных строк в POSIX shell'e клавиши со стрелками. Кроме клавиш **h** и **l**, для перемещения курсора можно использовать клавиши **Забой** и **Пробел**.

Наиболее часто встречаемой опечаткой является ситуация, когда при вводе две буквы меняются местами. Предположим, была введена следующая командная строка, в которой в слове **ford** буквы **r** и **o** поменялись местами:

```
$ cd $HOME/tree/car.models/frod/sports
cd: directory not found
```

Чтобы исправить командную строку и повторно запустить команду, нужно выполнить следующие шаги:

Esc

k Клавиша нажимается столько раз, сколько нужно, чтобы на экране появилась данная команда

w Команда вводится несколько раз, пока курсор не окажется под буквой **f** в слове **frod**.

l После ввода данной команды курсор окажется под буквой **r** в слове **frod**.

x **p** Удалить букву **r** и вставить ее после буквы **o**

Return Выполнить команду

7–11. СЛАЙД: Среда пользователя

Среда пользователя

- Среда пользователя описывает сеанс работы, в котором исполняются программы

Синтаксис:

env

Пример:

```
$ env
HOME=/home/gerry
PWD=/home/gerry/develop/basics
EDITOR=vi
TERM=70092
...
PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin:\
/home/gerry/bin
```

Пояснения

Среда пользователя передает программам, которые исполняет пользователь, многие характеристики сеанса работы в системе. Иными словами, среда описывает для системы сеанс работы пользователя. В среде содержится следующая информация:

- Маршрутное имя входного каталога пользователя
- Где размещать электронную почту
- Зона времени, в которой работает пользователь
- Входное имя пользователя, под которым он начал сеанс работы в системе
- Где shell будет искать команды при их запуске
- Тип и характеристики терминала
- Другая информация, которая может потребоваться прикладной программе

Например, команды **vi** и **more** должны знать тип терминала, на котором работает пользователь, чтобы корректно форматировать выводимую ими информацию.

Аналогом среды пользователя может служить атмосфера учреждения, в котором работает сотрудник. Такие характеристики учреждения, как освещенность, уровень шума,

температура являются общими для всех сотрудников. Специфические для конкретного сотрудника характеристики учреждения создают его собственную атмосферу. Такие характеристики могут включать в себя задачи, над которыми он работает, освещенность его рабочего места, а также его отношение к другим сотрудникам. Рабочая атмосфера сотрудника уникальна для него, так же как и среда пользователя уникальна для пользователя.

Многие приложения требуют настройки среды тем или иным способом. Это можно сделать, соответствующим образом модифицировав файл **.profile**.

Работая в системе, можно проверить установки среды, выполнив команду **env**. Эта команда выводит значения всех установленных характеристик среды.

В выводе команды **env** слова, указанные слева от знака **=**, определяют имена различных переменных среды, установленных в ней. Весь текст строки, расположенный справа от знака **=**, определяет значение, ассоциированное с данной переменной среды. Более подробную информацию о данной команде можно найти в статье **env (1)**.

Конкретные значения тех или иных переменных среды устанавливаются в соответствии с назначением этих переменных. Ниже приведены некоторые переменные среды и их назначение:

<i>TERM, COLUMNS, LINES</i>	Описывают используемый терминал
<i>HOME</i>	Определяет маршрутное имя входного каталога пользователя
<i>PATH</i>	Перечисляет каталоги, в которых можно найти исполняемую команду
<i>LOGNAME</i>	Определяет имя пользователя, под которым он вошел в систему
<i>ENV, HISTFILE</i>	Специальные переменные POSIX shell'a
<i>DISPLAY</i>	Специальная переменная X Window

Одни из этих переменных (например, *LOGNAME*) устанавливаются самой системой, в то время как другие (например, *ENV*) могут быть установлены администратором системы или пользователем в файлах **/etc/profile** и **.profile**.

7–12. СЛАЙД: Установка переменных среды

Установка переменных среды

- Переменная shell'a определяется именем и имеет значение
- Значение, ассоциированное с именем, может быть изменено
- Некоторые переменные определяются во время входа в систему
- Пользователь может определить новые переменные shell'a

Синтаксис:

имя=значение

Пример:

\$ PATH=/usr/bin/X11:/usr/bin

Пояснения

Переменная shell'a аналогична переменной в алгебре. Она задается именем, которое представляет некоторое значение. Присваивание переменной некоторого значения позволяет связать это значение с именем переменной. Это значение в дальнейшем можно получить, указав имя переменной. Если значение изменяется, тогда при указании имени той же переменной будет получено новое значение. Присваивание переменной некоторого значения имеет следующий синтаксис: **имя=значение**.

Такое предложение может быть введено и с терминала после приглашения shell'a, и в shell программе. *Следует обратить внимание на то, что перед знаком присваивания (=) и после него нет пробелов.* Это гарантирует, что shell не будет интерпретировать данное предложение как вызов команды.

Важно понимать различие между **именем** переменной shell'a и **значением** переменной shell'a. Значение переменной устанавливается при выполнении предложения присваивания, например:

```
TERM=70092
```

Это предложение предписывает shell'у запомнить имя *TERM*; когда же будет затребовано значение переменной *TERM*, ответом будет 70092.

Ограничения на задание имени переменной

Имя переменной должно начинаться с буквы (a–z или A–Z), за которой могут следовать буквы, цифры и знак подчеркивания. На количество символов, использованных в имени переменной, ограничений нет.

7–13. СДАЙД: Две важные переменные

Две важные переменные

- Переменная *PATH*
 - определяет список каталогов, в которых shell ищет команды, запускаемые на исполнение
- Переменная *TERM*
 - описывает тип терминала и размеры экрана для исполняющейся программы

```
$ env
...
PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin
$ TERM=70092
$
$ tset
Erase is Backspace
Kill is Ctrl-U
$
```

Пояснения

PATH

Переменная *PATH* определяет список каталогов, в которых shell ищет команды. Эта переменная позволяет пользователю вводить только имя команды вместо ее полного маршрутного имени (например, **vi** вместо **/usr/bin/vi**). Ниже приведен пример задания значения переменной *PATH* по умолчанию:

```
PATH=/usr/bin:/usr/contrib/bin:/usr/local/bin
```

Это означает, что когда пользователь вводит команду, shell будет искать эту команду в каталоге **/usr/bin**, затем (если не найдет) в каталоге **/usr/contrib/bin** и так до тех пор, пока он или найдет требуемую команду, или просмотрит все указанные в переменной каталоги. Если команда не будет найдена ни в одном из каталогов, заданных переменной *PATH*, на экран будет выведено сообщение об ошибке: **имя_команды: not found**

TERM

TERM – это переменная среды, которая описывает характеристики терминала пользователя. Многие команды для своей корректной работы должны знать тип используемого терминала. Например, команда **ls** должна знать ширину экрана (т.е. количество колонок – позиций в строке экрана), команда **more** должна знать количество строк на экране, а команда **vi** для корректной работы должна знать не только количественные характеристики экрана

(количество строк и колонок), но также и многие другие свойства терминала, зависящие от его типа. Тип терминала устанавливается, например, с помощью задания номера модели терминала (например, 2392, 70092, и другие).

Используемый по умолчанию способ установки переменной терминала выводит пользователю приглашение установить соответствующий тип терминала следующим образом:

```
TERM= (hp)
```

В ответ на это приглашение пользователь может или нажать клавишу `[Return]`, чтобы установить тип терминала равным **hp**, или же ввести соответствующий тип терминала, на котором он работает. Тип терминала **hp** определяет стандартный терминал Hewlett-Packard со стандартными размерами (24 строки и 80 колонок).

Администратор системы может так настроить систему, что она никогда не будет запрашивать у пользователя тип терминала. В этом случае следует проверить установку переменной *TERM* с помощью команды **env**. Если используется рабочая станция с единственным дисплеем, переменная *TERM*, скорее всего, будет установлена корректно, и ее не надо будет изменять.

Если терминал при выполнении таких команд, как **more** и **vi**, ведет себя странно, следует проверить установку переменной *TERM*. Если она установлена корректно, следует выполнить команду **tset**, которая переустановит характеристики терминала на основании типа терминала, заданного переменной *TERM*.

7–14. ТЕКСТОВАЯ СТРАНИЦА: Обычно используемые значения переменных

Обычно используемые значения переменных

Значения переменных, имена которых выделены **ПОЛУЖИРНЫМ (BOLD)** шрифтом, *можно* настраивать.

EDITOR=/usr/bin/vi	Устанавливает редактор vi для редактирования командных строк
ENV=\$HOME/.shrc	Устанавливает выполнение файла \$HOME/.shrc при запуске shell'a
FCEDIT=/usr/bin/vi	Начинает сеанс редактирования vi для редактирования ранее введенных команд
HOME=/home/user3	Определяет входной каталог пользователя
~ (тильда)	Для POSIX shell'a эквивалентна входному (<i>HOME</i>) каталогу
HISTFILE=\$HOME/.sh_history	Определяет файл, в котором запоминаются все команды, вводимые в интерактивном режиме
LOGNAME=user3	Соответствует входному имени пользователя
MAIL=/var/mail/user3	Определяет почтовый ящик пользователя
OLDPWD=/tmp	Определяет имя предыдущего текущего каталога
PATH=/usr/bin:\$HOME/bin	Определяет список каталогов, в которых выполняется поиск команд
PS1='\$ '	Определяет первичное приглашение shell'a. Обратите внимание: в приводимых ниже примерах должны быть использованы одиночные кавычки ('), а не двойные (").
PS1=' [!] \$ '	В приглашении выводится текущий номер командной строки
PS1='\$PWD \$ '	В приглашении выводится имя текущего каталога
PS1=' [!]\$PWD \$ '	В приглашении выводятся текущий номер командной строки и имя текущего каталога

PWD=/home/user3/tree	Определяет текущий рабочий каталог
SHELL=/usr/bin/sh	Определяет используемый интерпретатор команд
TERM=2392a	Определяет тип терминала. Для этого используется команда: eval 'tset -s -Q -h' Во время входа в систему эта команда читает файл /etc/ttytype, чтобы установить соответствие между портом терминала пользователя и типом терминала. Это полезно в том случае, когда к системе подключено несколько терминалов разных моделей.
TMOUT=300	Устанавливает время тайм-аута: если в течение указанного количества секунд с терминала не будет введена ни одна команда (т.е. не будет нажата клавиша Return), shell завершится и завершит сеанс работы пользователя
TZ=EST5EDT	Определяет часовой пояс, который использует система для вывода времени

Переменная *TERM*

Для того чтобы система UNIX знала характеристики терминала, должна быть корректно установлена переменная *TERM*. Многие команды должны знать тип используемого ими терминала, чтобы корректно выводить на экран результаты своей работы. Например, команды **more** и **vi** для корректного управления экраном должны знать его размеры (количество строк и колонок).

Переменная *TERM* может быть определена явно, с помощью присваивания переменной необходимого значения, или неявно, при выполнении команды **tset**. В этом случае значение переменной устанавливается в зависимости от того, какому значению в файле **/etc/ttytype** соответствует терминал, на котором работает пользователь.

В приведенной ниже таблице представлены некоторые типы терминалов и соответствующие им значения переменной *TERM*. Если в приведенной таблице отсутствует тип используемого терминала, нужно обратиться к соответствующему подкаталогу в каталоге **/usr/lib/terminfo**.

Модель терминала	Значение TERM
HP 2392a	2392a
HP 70092	70092
HP 70094	70094
vt 100	vt100
Wyse 50	wy50
Графический терминал среднего разрешения (512 x 600 пикселей)	300l или hp300l
Графический терминал высокого разрешения (1024 x 768 пикселей)	300h или hp300h
Дисплейная станция HP 98550 (1280 x 1024 пикселей)	98550, hp98550, 98550a или hp98550a
HP 98720 или HP 98721 SRX (1280 x 1024 пикселей)	98720, hp98720, 98720a, hp98720a, 98721, hp98721, 98721a или hp98721a
HP 98730 или HP 98731 Turbo SRX (1280 x 1025 пикселей)	98730, hp98730, 98730a, hp98730a, 98731, hp98731, 98731a или hp98731a

7-15. СЛАЙД: Что происходит при входе в систему?



Пояснения

Когда пользователь собирается начать сеанс работы в системе, прежде всего он видит на экране приглашение **login:**. Когда он введет с клавиатуры свое входное имя, система прочтает его и выведет приглашение **password:** на ввод пароля. После того как пользователь введет свой пароль, система проверит, есть ли соответствующие имя и пароль пользователя в системном файле паролей (`/etc/passwd`). Если имя и пароль пользователя введены правильно, система перемещает пользователя в его входной каталог и запускает для него shell. Все это пользователь видит каждый раз, когда он входит в систему. Важно понять, что же происходит на самом деле, когда стартует shell.

1. **getty**

- Выводит содержимое файла `/etc/issue`
- Выдает на экран приглашение **login:**
- Запускает процесс **login**

2. **login**

- Проверяет правильность ввода имени и пароля пользователя
- Перемещает пользователя в его входной каталог
- Запускает пользовательский shell

3. shell

- a. Выполняет файл **/etc/profile** для POSIX, Bourne и Korn shell'ов (или файл **/etc/csh.login** для C shell'a)
- b. Выполняет файл **.profile** (или **.login**), находящийся во входном каталоге пользователя
- c. Выполняет файл **.kshrc**, находящийся во входном каталоге пользователя (POSIX и Korn shell'ы), если такой файл создан пользователем и если в файле **.profile** его имя (**.kshrc**) присвоено переменной среды **ENV**
- d. Выводит на экран первичное приглашение shell'a

Когда shell начинает свою работу, прежде всего он читает команды из системного командного файла **/etc/profile**. Каждый раз, когда любой пользователь начинает сеанс работы в системе и для него стартует shell, shell будет читать этот файл. Во входном каталоге пользователя может находиться файл с именем **.profile**. После того как shell прочтет файл **/etc/profile**, он начнет читать собственный файл пользователя **.profile**. Эти две shell программы используются для настройки среды пользователя.

Файл **/etc/profile** устанавливает общую среду, используемую всеми пользователями системы, а файл **.profile** настраивает эту среду в соответствии с потребностями конкретного пользователя. Так как каждый пользователь использует файл **/etc/profile**, за его создание и наполнение отвечает администратор системы. Если пользователь хочет настроить собственную среду в соответствии со своими потребностями, он может создать свой собственный файл **.profile**.

После того как эти два файла будут выполнены, shell выводит на экран свое первичное приглашение.

Замечание относительно CDE

Если пользователь входит в систему с помощью CDE, обычно CDE не использует стартовые процедуры shell'a **/etc/profile** и **\$HOME/.profile** или **\$HOME/.login**. Тем не менее, пользователь может заставить CDE выполнить стартовую процедуру **\$HOME/.profile** (для пользователей, использующих **sh** или **ksh**) или **\$HOME/.login** (для пользователей **csh**), если включит в файл **.dtprofile** следующую установку переменной среды:

```
export DTSOURCEPROFILE=true
```

В противном случае при входе в систему будет выполняться только файл **.dtprofile**. Файл **.dtprofile** содержит закомментированные строки установки переменных, необходимых для настройки среды CDE.

Замечание относительно входа в систему Linux



Управление входом в систему Linux похоже на управление входом в HP-UX, хотя и может незначительно отличаться от него. Файлы, управляющие входом в систему, зависят от входного shell'a пользователя и графической оболочки, в которой осуществляется вход в систему по умолчанию. Поскольку здесь возможно большое количество вариантов, процедура входа в систему в данном примечании не может быть рассмотрена со всеми подробностями. Однако там, где это возможно, включается информация, специфическая для системы Linux.

7–16. СЛАЙД: Стартовые файлы shell'a

Стартовые файлы shell'a

Если используется shell ...	Локальной входной процедурой является ...
Korn (/usr/bin/ksh)	.profile .kshrc
Bourne (/usr/old/bin/sh)	.profile
POSIX (/usr/bin/sh)	.profile .shrc
Ограниченный (/usr/bin/rsh, /usr/bin/rksh)	.profile
C (/usr/bin/csh)	.login .cshrc

Пояснения

Для настройки сеанса работы пользователя необходимо установить значения некоторых переменных среды (например, *PATH*, *EDITOR*). Как было показано ранее, если эти переменные определяются интерактивно, они должны переопределяться *каждый* раз при входе в систему. Для того чтобы облегчить настройку сеанса работы системы, используются файлы **.profile**, **.kshrc**, **.shrc** и **.cshrc**. Эти файлы представляют собой обычные shell процедуры, содержащие определение переменных среды, определение псевдонимов (aliases) и вызовы программ, которые должны выполняться при входе в систему.

.profile

Если пользователь хочет изменить те или иные переменные среды, установленные и предоставленные ему администратором системы, он может создать или модифицировать собственный файл **.profile**. Обычно этот файл содержит команды определения или настройки переменных среды, установки терминала пользователя и вызовы команд, например, **date**, которые выполняются во время входа пользователя в систему. Из файла **.profile** можно также инициировать приложение пользователя, если в этот файл включить предложение **exec имя_приложения**. В этом случае пользователь не увидит на экране первичное приглашение shell'a, а когда приложение завершит свою работу, пользователь автоматически выйдет из системы.

/etc/profile

Файл **/etc/profile** представляет собой общесистемный стартовый файл, который выполняется для *всех* пользователей, которые работают в Bourne, Korn или POSIX shell'ах. Этот файл настраивается администратором системы таким образом, чтобы предоставить всем пользователям непротиворечивую среду, необходимую для запуска их приложений. Рядовые пользователи обычно не имеют права записи в этот файл, следовательно, им не разрешено модифицировать его содержимое. Пользователи могут настроить свою собственную среду, используя собственную копию файла **.profile** или файлы **.kshrc** или **.shrc**.

.kshrc и .shrc

Korn и POSIX shell'ы имеют не обязательные конфигурационные файлы обычно с именами **.kshrc** и **.shrc**, соответственно. Эти конфигурационные файлы используются для настройки собственной среды пользователя и во многом похожи на файл **.profile**. Однако в отличие от файла **.profile**, файлы **.kshrc** и **.shrc** читаются и выполняются каждый раз, когда стартует новый shell, а не только во время входа пользователя в систему. Это позволяет пользователю устанавливать свои собственные псевдонимы и даже первичное приглашение shell'а каждый раз, когда он запускает новый shell. В среде, аналогичной X 11 Windows, пользователь может иметь несколько различных shell'ов, работающих одновременно. Пользователь может использовать файлы **.kshrc** или **.shrc** таким образом, что все эти разные shell'ы будут выглядеть одинаково.

Имена этих файлов – **.kshrc** и **.shrc** – не являются предопределенными. Когда пользователь запускает shell, этот shell ищет файл, на который ссылается переменная среды **ENV**. Часто этот файл называют **.kshrc** или **.shrc**, но может быть использовано и любое другое имя по желанию пользователя.

Для того чтобы использовать файлы **.kshrc** или **.shrc**, пользователь должен в своем файле **.profile** (или в **.dtprofile**, если используется CDE) определить новую переменную среды – переменную **ENV**. Для этого в файл **.profile** нужно добавить следующие строки:

```
ENV=~/.kshrc
export ENV
или
ENV=~/.shrc
export ENV
```

Тем самым пользователь сообщает shell'у, что он хочет использовать файл **.kshrc** (или **.shrc**), находящийся в его входном каталоге. После этого можно включить в файл **.kshrc** (или **.shrc**) все псевдонимы команд, определенные и используемые пользователем.

Если пользователю приходится работать и в среде Bourne shell, и в среде POSIX shell, он может сохранить все определения специфических для POSIX shell переменных в этом файле, так как он *никогда* не читается Bourne shell'ом.

.cshrc и .login

Если пользователь входит в систему, используя в качестве входного shell'а C shell, этот shell ищет во входном каталоге файл с именем **.login**. Если этот файл будет найден, тогда прежде чем на экран будет выведено первичное приглашение shell'а, будут выполняться содержащиеся в нем команды. Все происходит точно так же, как и для файла **.profile** в

POSIX, Bourne и Korn shell'ах. В случае соответствующего указания также будут выполняться команды из файла **.cshrc**, прежде чем на экран будет выведено первичное приглашение C shell'а.

Система Linux

.bash_profile



При входе в систему Linux вместо файла **.profile** читается файл **.bash_profile**. Файл **.profile** тоже будет читаться, если в файл **.bash_profile** будут включены соответствующие команды.

7–17. СЛАЙД: Встроенные команды shell и команды UNIX

Встроенные команды shell и команды UNIX

Встроенные команды shell являются частью shell'a

Примеры:

cd
pwd
echo

Команды UNIX находятся в каталоге **/usr/bin**

Примеры:

more
file
ls

Некоторые встроенные команды доступны также и как отдельные команды. Система находит место размещения команды UNIX, используя переменную среды *PATH*

Пояснения

Некоторые команды, вводимые пользователем с клавиатуры, представляют собой файлы, размещенные в каталогах, таких как, например, **/usr/bin**. Такие команды являются командами UNIX. С другой стороны, многие команды, такие как, например, **cd**, **pwd** и **echo**, в действительности размещены непосредственно в самом shell'e. Такие команды не размещаются в файловой системе UNIX; эти команды можно трактовать как подпрограммы, включенные в программу shell. Такие команды являются встроенными командами shell'a.

Так как команды UNIX могут находиться в разных каталогах, shell должен знать, где искать эти команды. Переменная среды *PATH* определяет каталоги, в которых следует искать команды UNIX, а также порядок поиска.

Команды UNIX могут иметь такие же имена, как и встроенные команды shell'a. В этом случае, для того чтобы использовать команды UNIX, а не встроенные с таким же именем, пользователь должен задать такую команду с помощью *маршрутного* имени.

7–18. СЛАЙД: Поиск команд – команда **whereis**

Поиск команд — команда **whereis**

Синтаксис:

\$ whereis [-b|-m|-s] команда Просматривает каталоги
для поиска указанной команды

Примеры:

```
$ whereis if
if :
$
$ whereis ls
ls : /sbin/ls /usr/bin/ls /usr/share/man/man1.Z/ls.1
$
$ whereis cd
cd : /usr/bin/cd /usr/share/man/man1.Z/cd.1
$
$ whereis holdyourhorses
holdyourhorses :
$
```

Пояснения

UNIX размещает свои команды в четырех основных каталогах: **/sbin**, **/usr/bin**, **/usr/local/bin** и **/usr/contrib/bin**. Команда **whereis** просматривает эти, а также и некоторые другие каталоги, чтобы определить, где именно размещена заданная команда. Многие пользователи в своих входных каталогах создают собственные **bin** каталоги. **whereis** не просматривает эти каталоги. В некоторых случаях пользователь может потерять следы какой-либо команды и ее статьи справочника. В этом случае UNIX с помощью команды **whereis** предоставит путь к месту размещения команды и ее статей справочника.

Команда **whereis** требует задания единственного аргумента – имени команды. Результатом **whereis** является абсолютное маршрутное имя исполняемого файла и статей справочника, ассоциированных с указанной командой.

Команда **whereis** просматривает следующие каталоги:

<code>/usr/src/*</code>	<code>/usr/sbin</code>	<code>/sbin</code>
<code>/usr/bin</code>	<code>/usr/lbin</code>	<code>/usr/ccs/bin</code>
<code>/usr/share/man/*</code>	<code>/usr/local/man/*</code>	<code>/usr/local/bin</code>
<code>/usr/local/games</code>	<code>/usr/local/include</code>	<code>/usr/local/lib</code>
<code>/usr/contrib/man/*</code>	<code>/usr/contrib/bin</code>	<code>/usr/contrib/games</code>
<code>/usr/contrib/include</code>	<code>/usr/contrib/lib</code>	<code>/usr/share/man/\$LANG/*</code>
<code>/usr/local/man/\$LANG/*</code>	<code>/usr/contrib/man/\$LANG/*</code>	

Если необходимо изменить список каталогов, просматриваемых командой **whereis**, следует указать опции **-b**, **-m** или **-s**; эти опции устанавливают поиск только исполняемых файлов, статей справочника или исходных текстов, соответственно.

7-19. ТЕКСТОВАЯ СТРАНИЦА: Пример файла .profile

Пример файла .profile

```
# Set up the command search paths:
# (задание каталогов, в которых осуществляется поиск команд)
PATH=./bin:/usr/bin ; export PATH

# Define the prompt:
# (установка первичного приглашения)
PS1="$ " ; export PS1

# Set up the terminal:
# The -h option in the following tset command tells the shell to
# find the appropriate terminal type to assign to TERM from the
# file /etc/ttytype
# (установка терминала: опция -h команды tset предписывает shell'у искать тип терминала,
# в соответствии с которым будет установлена переменная TERM, в файле /etc/ttytype)
eval `tset -s -Q -h`

# You could also hardcode your terminal type with:
# (можно напрямую установить тип терминала с помощью следующей команды:)
#TERM=2392a

# Map control characters
# The intr "^C" maps Ctrl-c instead of DEL for program interrupt
# (Установка управляющих символов для терминала:
# запись intr "^C" устанавливает для прерывания программы вместо клавиши DEL комбинацию
# клавиш Ctrl-c)
stty erase "^H" kill "^U" intr "^C" eof "^D" susp "^S"
stty brkint hupcl ixon ixoff

# Uncomment the following line if you want to change default permissions
# (следует снять знак комментария у следующей строки, если нужно изменить устанавливаемые
# по умолчанию права доступа)
#umask 022

# Set up POSIX shell variables
# (установка переменных POSIX shell)

# Inform the POSIX shell to reference the $HOME/.kshrc file
# Aliases are usually defined here
# (установка ссылки на файл $HOME/.kshrc;
# определения псевдонимов команд размещаются обычно в этом файле)
ENV=$HOME/.kshrc
export ENV

# The following variables are used to set up the command stack
# and the history feature
# (определенные ниже переменные используются для установки свойств стека команд
# и файла предыстории)
EDITOR=/usr/bin/vi; export EDITOR
HISTSIZE=50; export HISTSIZE
HISTFILE=$HOME/.sh_history; export HISTFILE
FCEDIT=/usr/bin/vi; export FCEDIT

# Run the script .logout to clean out the history file
```

Модуль 7

Основы shell

```
# created by the POSIX shell command stack
# (по окончании сеанса работы выполнить процедуру .logout для очистки файла предыстории,
# созданного POSIX shell'ом)
trap "$HOME/.logout" 0

# The following lines can be updated for your application and uncommented
# if you want your application to start automatically when logging in
# (если нужно, чтобы при входе в систему автоматически запускалось собственное приложение
# пользователя, нужно соответствующим образом отредактировать приведенную ниже строку
# и удалить символ комментария)
#exec /usr/bin/имя_собственного_приложения
```

7–20. ТЕКСТОВАЯ СТРАНИЦА: Примеры файлов `.kshrc` и `.logout`

Пример файла `.kshrc`

```
# Customize the prompt:
# (настройка первичного приглашения)
# The ! will display the command number in the prompt
# (символ ! предписывает выводить в приглашении номер командной строки)
#PS1='[!] $ '

# The $PWD will display the present working directory in the prompt
# ($PWD предписывает выводить в приглашении имя текущего рабочего каталога)
#PS1='[!] $PWD $ '

# The hostname will display the system name in the prompt
# (hostname предписывает выводить в приглашении имя хост-машины)
#PS1="[`hostname`] $ "

# Define some aliases
# (определение некоторых псевдонимов команд)
alias ls="ls -aCF"
alias history="fc -l"
alias h="fc -l"
alias r="fc -e - "
alias mroe=more

# Set up the shell environment
# (установка среды shell'a)
set -o markdirs      # All directory names resulting from filename
                     # generation will have a trailing / appended
                     # все имена каталогов, полученные в результате генерации имен файлов,
                     # будут заканчиваться символом /
set -o monitor       # Jobs will send messages to screen when complete
                     # (при завершении задания на экран будет посылаться соответствующее
                     # сообщение)
set -u               # Treat unset parameters as an error when substituting
                     # (при подстановке значения переменных рассматривать не установленные
                     # параметры как ошибку)
```

Пример файла `.logout`

Когда пользователь вводит с клавиатуры команду, эта команда записывается в назначенный файл предыстории (**\$HISTFILE**). В POSIX shell'e отсутствует механизм автоматической очистки этого файла. Поэтому у пользователя может возникнуть потребность выполнять что-то подобное приведенному ниже. В приведенном примере текущий файл предыстории перемещается в *старый* файл предыстории. При следующем входе в систему будет создан новый файл предыстории. В результате файл предыстории не будет непредсказуемо увеличиваться в размерах. Для того чтобы POSIX shell использовал (выполнил) этот файл, в файле **.profile** должна быть включена соответствующая установка прерывания (команда **trap**).

```
# Change to login directory
# (перейти во входной каталог)
```

Модуль 7

Основы shell

```
cd
```

```
# Save the current history file  
# (сохранить текущий файл предыстории)  
mv $HOME/.sh_history $HOME/.sh_hist.old
```

```
# Send messages to the user  
# (послать сообщение пользователю)  
clear  
echo `whoami` logged out at `date`  
echo
```


7–21. УПРАЖНЕНИЯ: Основы shell

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

Система Linux



В приводимых ниже текстах упражнений для завершения ввода имен файлов используются последовательности клавиш, принятые в версии системы HP-UX. Выполняя эти упражнения в системе Linux, пожалуйста, обратитесь к приведенной ранее в этом модуле справочной таблице, в которой указаны последовательности клавиш, используемые для завершения ввода имени файла в вашем shell'е.

1. Создайте псевдоним команды с именем **h**, который позволит выполнить команду **history**.
2. Проверьте команды, включенные в файл **.shrc** (или **.kshrc**) в вашем входном каталоге. Добавьте к списку команд псевдоним **h**.
3. В командной строке определите псевдоним команды с именем **go**, который позволит перейти в каталог **tree** и выполнить в нем команду **ls -F**. Теперь выполните из командной строки команду **go**. Что произошло? Выполните команду **pwd**, чтобы увидеть, где вы находитесь. Вернитесь в свой входной каталог. (ПОДСКАЗКА: в одной командной строке можно ввести несколько команд, если разделять их точкой с запятой)
4. Завершите сеанс работы, а затем вновь войдите в систему и проверьте свои псевдонимы. Почему нужно было выходить из системы?

5. Убедитесь в том, что вы находитесь в вашем *HOME* каталоге. Что произойдет, если вы введете с клавиатуры текст **more f** EscEsc? Каким образом можно увидеть в командной строке имя файла **funfile**?

6. Находясь в *HOME* каталоге, скопируйте файл **frankenstein** в каталог **tree/car.models/ford/sports**. Используйте завершение ввода имени файла **frankenstein**, а также любого имени в маршрутном имени каталога.

7. Введите в командной строке следующие неправильные команды и *не нажимайте* Return:

```
cd /user/spol/ko/interface
```

Используя редактирование командной строки, исправьте текст так, чтобы можно было прочитать следующее:

```
cd /usr/spool/lp/interface
```

(*Не вводите заново правильный текст*).

8. Выполните команду **ls -F**.

Вызовите повторно эту команду на экран и замените **ls -F** на **ls -l**, используя необходимые команды редактора **vi**. Выполните полученную команду.

9. Используя стек команд, повторно вызовите на экран последнюю команду копирования и замените в ней имя **frankenstein** на **funfile**.

10. Повторно вызовите на экран последнюю команду копирования и модифицируйте ее таким образом, чтобы можно было вывести на экран содержимое каталога **sports**.

11. Вызовите повторно на экран последнюю выполнявшуюся команду и модифицируйте ее таким образом, чтобы можно было перейти в каталог **sedan** (ПОДСКАЗКА: маршрутное имя этого каталога – **tree/car.models/ford/sedan**). Используйте команду **pwd**, чтобы убедиться в изменении рабочего каталога.

12. Вернитесь в свой *HOME* каталог и с помощью команды **history** или созданного псевдонима **h** выведите на экран содержимое стек команд; затем с помощью команды **r** выполните повторно команду перехода в каталог **sedan**. Также с помощью команды **r** выведите на экран имя текущего рабочего каталога.

Модуль 8 —Дополнительные возможности shell

Цели:

Изучив этот модуль, вы сможете:

- Использовать механизм подстановок shell'a, включающий в себя подстановки переменных, команд и тильда подстановки
- Устанавливать и изменять значения переменных shell'a
- Экспортировать локальные переменные в среду
- Обеспечивать доступность переменных из порожденных процессов
- Объяснять, как создаются процессы

8–1. СЛАЙД: Shell: подстановки

Shell: подстановки

Shell допускает три типа подстановок:

- Подстановка переменных
- Подстановка команд
- Тильда подстановка

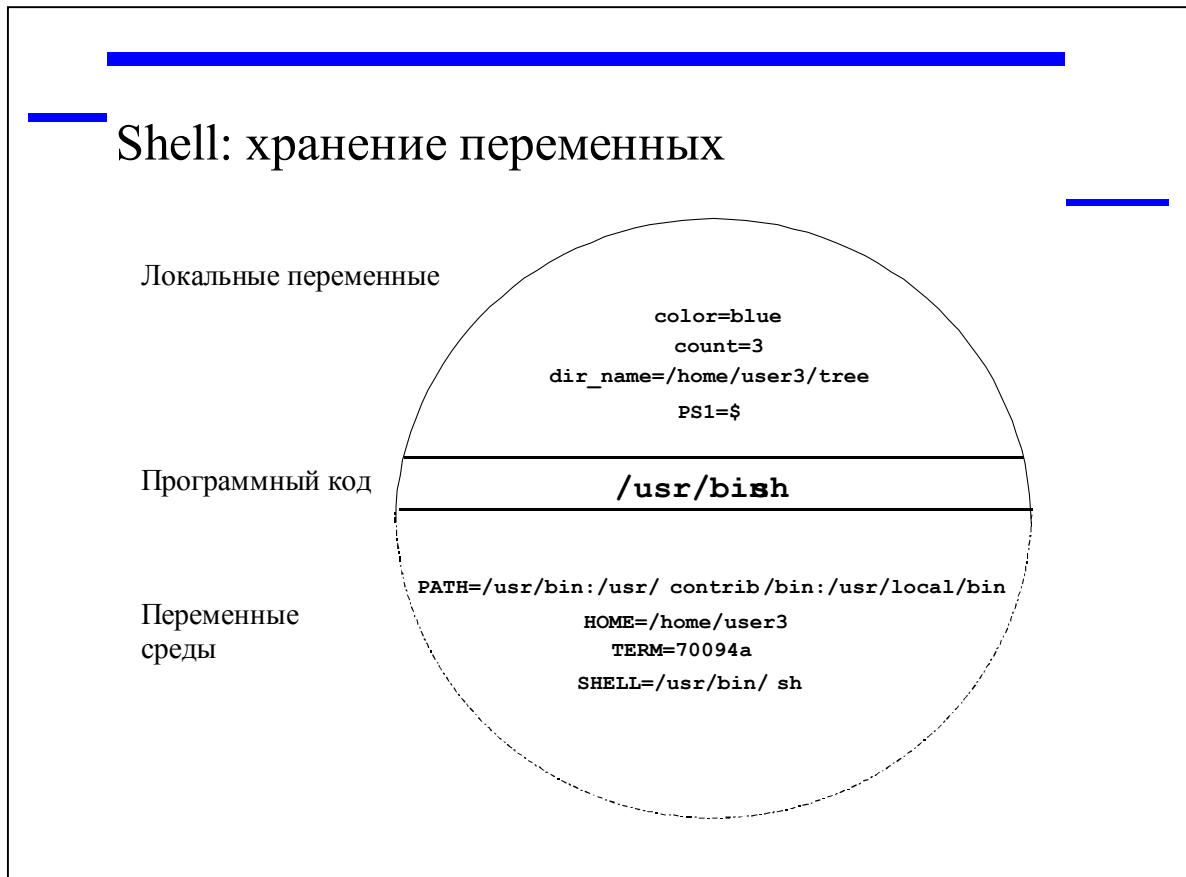
Пояснения

В shell'е существуют три типа подстановок:

- Подстановки переменных
- Подстановки команд
- Тильда подстановки

Подстановки используются для увеличения скорости ввода и исполнения командных строк.

8–2. СЛАЙД: Shell: хранение переменных



Пояснения

В shell встроены две области памяти, используемые его переменными: **область локальных данных** и **среда**. В области локальных данных память выделяется, когда определяется *новая* переменная. Переменные, размещенные в этой области, доступны только текущему shell'у и поэтому их часто называют *локальными переменными*. Никакой процесс-потомок не имеет доступа к этим локальным переменным. Однако процесс-потомок может иметь доступ к переменным, перемещенным (экспортированным) в среду.

Существует несколько специальных переменных shell'a, которые устанавливаются для пользователя, когда он входит в систему. Многие из них хранятся в среде, некоторые, как, например, PS1 и PS2, размещаются обычно в области локальных данных. Пользователь может изменять значения этих переменных, чтобы настроить характеристики своего сеанса связи.

Для того чтобы вывести на экран значения *всех* переменных, которые в текущий момент находятся в среде, можно использовать команду **env**, например:

```
$ env
MANPATH=/usr/share/man:/usr/contrib/man:/usr/local/man
PATH=/usr/bin:/usr/ccs/bin:/usr/contrib/bin:/usr/local/bin
LOGNAME=user3
ERASE=^H
SHELL=/usr/bin/sh
HOME=/home/user3
TERM=hpterm
PWD=/home/user3
TZ=PST8PDT
EDITOR=/usr/bin/vi
```


8–3. СЛАЙД: Установка значений переменных shell'a

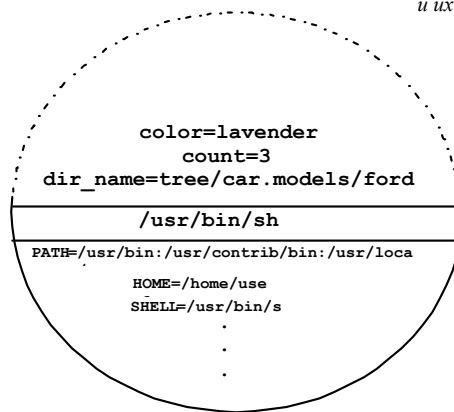
Установка значений переменных shell'a

Синтаксис: *имя=значение*

Примеры

```
$ color=lavender
$ count=3
$ dir_name=tree/car.models/ford
$ PS1=hi_there$
hi_there$set
```

Установка локальной переменной
Установка локальной переменной
Установка локальной переменной
Изменение переменной среды
Вывод на экран всех переменных
и их значений



Пояснения

Когда пользователь создает новую переменную, например, *color*, эта переменная будет размещена в области локальных данных. Когда присваивается новое значение существующей переменной среды, например, *PATH*, старое значение этой переменной, размещенное в среде, будет заменено новым.

8–4. СЛАЙД: Подстановка переменных

Подстановка переменных

Синтаксис:

\$имя Предписывает shell'у выполнить подстановку переменной

Примеры:

```
$ echo $PATH
/usr/bin:/usr/contrib/bin:/usr/local/bin
$ PATH=$PATH:$HOME:.
$ echo $PATH
/usr/bin:/usr/contrib/bin:/usr/local/bin:/home/user3:.
$ echo $HOME
/home/user3
$ file_name=$HOME/file1
$ more $file_name
<содержимое файла /home/user3/file1>
```

Пояснения

Каждая определенная переменная имеет ассоциированное с ней значение. Если имени переменной непосредственно предшествует знак доллара (\$), shell замещает эту конструкцию значением переменной. Такая операция называется **подстановкой переменной**. Это одно из действий, которые shell выполняет *перед* запуском команды, введенной из командной строки. Только после того, как shell выполнит в командной строке все подстановки переменных, он будет выполнять саму команду. Поэтому значениями переменных могут быть непосредственно команда, аргументы команды или вся командная строка, что обеспечивает удобный механизм для переименования часто используемых длинных маршрутных имен или длинных командных строк.

Примеры

Данный слайд представляет некоторые способы использования переменных shell'a. Следует обратить внимание на то, что подстановка переменной может появиться в любом месте командной строки, и что в одной командной строке могут появляться ссылки на несколько переменных. Как показано на слайде, для обновления значения переменной может быть использовано даже существующее значение этой переменной.

```
$ echo $PATH
/usr/bin:/usr/contrib/bin:/usr/local/bin
$ PATH=$PATH:$HOME:.
$ echo $PATH
/usr/bin:/usr/contrib/bin:/usr/local/bin:/home/user3:.
```

```
$ echo $HOME
/home/user3
$ file_name=$HOME/file1
$ more $file_name
<contents of /home/user3/file1>
```

*file_name=/home/user3/file1
more /home/user3/file1*

ПРИМЕЧАНИЕ: Чтобы вывести на экран текущее значение переменной, удобно использовать команду **echo \$имя**.

Использование фигурных скобок ({ })

Предположим, что есть две переменные – с именами *file* и *file1*. Им можно присвоить какие-либо значения, например:

```
$ file=this
$ file1=that
$ echo $fileand$file1
sh: fileand: parameter not set
$ echo ${file}and$file1
thisandthat
```

ищет переменные с именами fileand и file1
ищет переменные с именами file и file1

Для того чтобы отделить имя переменной от окружающего текста, используются фигурные скобки.

8–5. СЛАЙД: Подстановка переменных (продолжение)

Подстановка переменных (продолжение)

```
$ dir_name=tree/car.models/ford
$ echo $dir_name
tree/car.models/ford
$ ls -F $dir_name
sedan/  sports/
$ my_ls="ls -aFC"
$ $my_ls
./          file.1          tree/
../         file.2
$ $my_ls $dir_name
./          ../          sedan/  sports/
$ cd /tmp
$ dir_name=/home/user2/tree/dog.breeds/retriever
$ $my_ls $dir_name
./          ../          golden  labrador  mixed
```

Пояснения

Использование в качестве значения переменной *абсолютного маршрутного имени*, ссылающегося на файл или каталог, позволяет из любого места в иерархии файловой системы получить доступ к требуемому файлу или каталогу.

Рассмотрим пример, приведенный на слайде:

```
$ dir_name=tree/car.models/ford
$ echo $dir_name           echo tree/car.models/ford
tree/car.models/ford
$ ls -F $dir_name          ls -F tree/car.models/ford
sedan/ sports/
$ my_ls="ls -aFC"          используются кавычки, чтобы shell игнорировал
                           пробелы
$ $my_ls                   ls -aFC
./  file.1  tree/
../  file.2
$my_ls $dir_name          ls -aFC tree/car.models/ford
./  ../  sedan/  sports/
$ cd /tmp
$ dir_name=/home/user2/tree/dog.breeds/retriever
$ $my_ls $dir_name         ls -aFC /home/user2/tree/dog.breeds/retriever
./  ../  golden  labrador  mixed
```

8–6. СЛАЙД: Подстановка команд

Подстановка команд

Синтаксис:

`$ (команда)`

Примеры:

```
$ pwd
/home/user2
$ curdir=$(pwd)
$ echo $curdir
/home/user2
$ cd /tmp
$ pwd
/tmp
$ cd $curdir
$ pwd
/home/user2
```

Пояснения

Подстановка команд используется для того, чтобы в командной строке заменить команду результатом ее исполнения. Обычный синтаксис подстановки команд, поддерживаемый POSIX, имеет вид:

`$ (команда)`

Подстановка команд позволяет перехватить результат выполнения команды и использовать его в качестве аргумента другой команды или же присвоить этот результат некоторой переменной. Как и в случае подстановки переменных, подстановка команды выполняется прежде, чем будет выполнена предшествующая ей команда в командной строке. Если результат выполнения команды содержит символы «возврат каретки» или «перевод строки», они будут заменены пробелами.

Подстановка команд обозначается круглыми скобками, в которых записана команда и которым предшествует знак доллара, как и в случае подстановки переменных.

В подстановке команд может быть использована любая корректная процедура shell'a. Shell сканирует строку и выполняет любую команду, которую обнаружит после левой круглой скобки и пока не найдет соответствующую правую круглую скобку.

Альтернативная форма подстановки команд использует символы «обратный апостроф» (или знак тупого ударения), в которые заключена команда: ``команда``.

Эта форма эквивалентна форме **\$ (команда)** и представляет собой единственный способ задания подстановки команд, распознаваемый Bourne shell'ом. Форму **`команда`** следует использовать в тех процедурах, которые могут выполняться в POSIX, Korn и Bourne shell'ах.

Примеры

Подстановка команд очень часто используется для того, чтобы запомнить результаты выполнения команды (присвоив их некоторой переменной) для их последующей обработки. Обычно команда **pwd** выводит свои результаты на экран. Если выполнить следующее присваивание:

```
$ curdir=$(pwd) или $ curdir=`pwd`
```

то результат выполнения команды **pwd** будет присвоен переменной **curdir**.

Рассмотрим следующий пример:

```
$ echo date
date
$ banner date
##### # #####
# # # # #
# # # # #
# # # # #
##### # # #####$
echo "$(date)"          выполняется: echo "Thu Jul 11 16:40:32 EDT 1994"
Thu Jul 11 16:40:32 EDT 1994
$ banner "$(date)"      выполняется: banner "Thu Jul 11 16:40:32 EDT 1994"

##### # # # # # # # # # # # #
# # # # # # # # # # # #
# ##### # # # # # # # # # #
# # # # # # # # # # # #
# # # ##### # # # # # # # #
# # # # # # # # # # # #
```

Обычно команда **date** выводит результаты своего выполнения на экран. Когда выполняется команда **banner date**, строка **date** является аргументом команды **banner** и выводится на экран. Во втором примере, когда **date** используется в подстановке команд, shell сначала выполнит команду **date** и подставит вместо аргумента **date** результаты выполнения этой команды. Поэтому на экран будут выведены десять первых символов аргумента команды **banner "Thu Jul 11 16:40:32 EDT 1994"**.

Система Linux



ПРИМЕЧАНИЕ: Версия команды **banner** в системе Linux не выводит результаты своего выполнения на экран, как это показано в приведенном выше примере. Поэтому пользователям системы Linux не рекомендуется использовать команду **banner**.

8–7. СЛАЙД: Тильда подстановка

Тильда подстановка

```
$ echo $HOME
/home/user3
$ echo ~
/home/user3

$ cd tree
$ echo $PWD
/home/user3/tree
$ ls ~+/dog.breeds
collie poodle

$ echo $OLDPWD
/home/user3/mail
$ ls ~-
/home/user3/mail/from.mike /home/user3/mail/from.jim

$ echo ~tricia/file1
/home/tricia/file1
```

Пояснения

Если слово начинается символом «тильда» (~), для этого слова выполняется тильда подстановка. Следует обратить внимание на то, что тильда подстановка выполняется только тогда, когда тильда находится в начале слова. Поэтому в строке `/~home/user3` тильда подстановка выполняться не будет. Тильда подстановка выполняется в соответствии со следующими правилами:

- Отдельный знак тильда или знак тильда, помещенный перед символом `/`, заменяется значением переменной *HOME*.
- Знак тильда, за которым следует символ `+`, заменяется значением переменной *PWD*. Значение переменной *PWD* устанавливается командой `cd` и соответствует имени текущего рабочего каталога.
- Знак тильда, за которым следует символ `-`, заменяется значением переменной *OLDPWD*. Значение переменной *OLDPWD* устанавливается командой `cd` и соответствует имени предыдущего рабочего каталога.
- Если за знаком тильда следует некоторое слово, а затем символ `/`, shell проверяет, не соответствует ли это слово какому-либо имени пользователя системы. Если такое соответствие будет обнаружено, последовательность `~слово` будет заменена маршрутным именем входного каталога данного пользователя.

Модуль 8

Дополнительные возможности shell

Тильда может быть использована в псевдонимах команд:

```
$ pwd
/home/user3
$ alias cdn='cd ~/bin'
$ cdn
$ pwd
/home/user3/bin
```

Система Linux



ПРИМЕЧАНИЕ: В системе Linux установленные по умолчанию права доступа могут запрещать пользователям доступ к *HOME* каталогам других пользователей.

8–8. СЛАЙД: Вывод значений переменных

Вывод значений переменных

```
$ echo $HOME
/home/user3

$ env
HOME=/home/user3
PATH=/usr/bin:/usr/sbin:/usr/local/bin
SHELL=/usr/bin/sh

$ set
HOME=/home/user3
PATH=/usr/bin:/usr/sbin:/usr/local/bin
SHELL=/usr/bin/sh
color=lavender
count=3
dir_name=/home/user3/tree

$ unset dir_name
```

Пояснения

Для того чтобы вывести значение *отдельной переменной* (вне зависимости от того, является она локальной или переменной среды), используется подстановка переменных **\$имя**.

Для того чтобы вывести значения *всех переменных среды*, определенных на данный момент, используется команда **env**.

Команда **set** используется для того, чтобы вывести *все* определенные на данный момент переменные, как локальные, так и переменные среды, а также их значения.

Команда **unset** может быть использована для удаления текущего значения указанной переменной. В действительности, значение переменной устанавливается равным NULL.

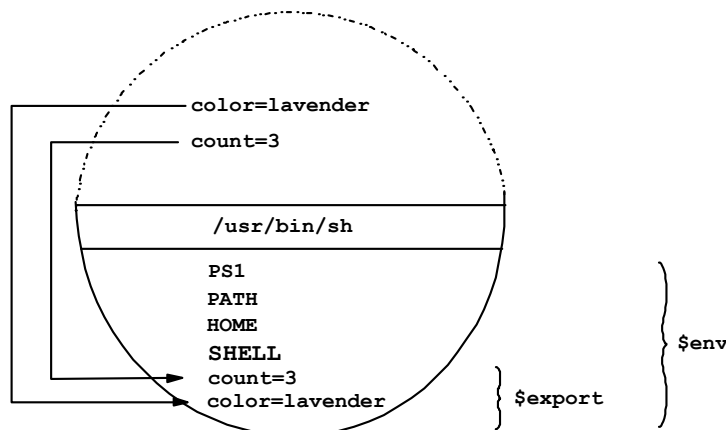
Команды **set** и **unset** являются встроенными командами shell'а. Команда **env** – это команда UNIX **/usr/bin/env**.

8–9. СЛАЙД: Экспортирование локальных переменных в среду

Экспортирование локальных переменных в среду

Синтаксис:

export *переменная* экспортирование *переменной* в среду



Пояснения

На диаграмме, приведенной на слайде, показано, как переменные *color* и *count* с помощью команды **export** перемещаются в среду:

```
$ color=lavender
$ export color
$ export count=3
$ export
export PATH=/usr/bin:/usr/ccs/bin:/usr/contrib/bin:/usr/local/bin
export color=lavender
export count=3
```

Для того чтобы переменная была доступна другим процессам, она должна находиться в среде. Когда переменная определяется, она размещается в области локальных данных, после чего она должна быть перемещена, или **экспортирована**, в среду.

Команда **export** *переменная* перемещает указанную переменную из области локальных данных в среду. Команда **export** *переменная=значение* позволяет одновременно установить (или изменить) значение переменной и переместить ее в среду. Команда **export** без аргументов аналогична команде **env** и выводит имена и значения всех экспортированных переменных. Следует обратить внимание на то, что команда **export** является встроенной командой shell'a.

8–10. СЛАЙД: Передача переменных в приложение



Пояснения

Каждому приложению или команде системы UNIX соответствует программный файл, хранящийся на диске. Многие стандартные команды системы UNIX размещены в каталоге `/usr/bin`. Когда какая-либо команда запускается на исполнение, соответствующий ей программный файл должен быть найден, загружен в оперативную память и затем исполнен. Выполняющаяся программа известна в системе UNIX под названием **процесс**.

Когда пользователь входит в систему UNIX, для него будет загружена программа shell и запущен процесс shell. Когда пользователь в ответ на приглашение shell'a вводит имя приложения (или команды), которое должно быть выполнено, для него создается и выполняется **порожденный** процесс. Создание и выполнение процесса осуществляется по шагам в соответствии со следующими правилами:

1. **fork** – порождает параллельный процесс, в котором дублируется родительский процесс shell, включая программный код, область данных среды и область локальных данных.
2. **exec** – замещает программный код и область локальных данных порожденного процесса кодом и локальными данными затребованного приложения.
3. **exec** – завершает выполнение процесса затребованного приложения.

Когда выполняется порожденный процесс, shell (**родительский процесс**) «спит» – ожидает завершения порожденного процесса. Как только порожденный процесс заканчивает свою работу, он завершается, освобождает память, ассоциированную с этим процессом, и

пробуждает родительский процесс, который опять готов принимать команды. Свидетельством того, что порожденный процесс завершен, является вывод на экран первичного приглашения shell'a.

Локальные переменные и переменные среды

Каждый раз, когда определяется новая переменная, она размещается в области локальных данных, ассоциированной с процессом. Если порожденному процессу требуется доступ к этим переменным, переменные с помощью команды **export** должны быть экспортированы в среду. Как только переменная попадает в среду, она становится доступной *всем* последующим порожденным процессам, так как среда передается каждому порожденному процессу.

В примере, приведенном на слайде, перед выполнением команды **vi** переменная **color** находится в локальной области данных, а переменная **TERM** – в среде. Когда будет введена команда **vi**, shell выполнит шаги **fork** и **exec**; область локальных данных порожденного процесса будет перезаписана его программным кодом, но среда без изменений передается порожденному процессу. Поэтому порожденному процессу **vi** будет *недоступна* переменная **color**, но *доступна* переменная **TERM**. Редактор **vi** должен знать тип терминала, на котором работает пользователь, чтобы правильно форматировать выводимые на экран данные. Он получит эту информацию, прочитав значение переменной **TERM**, доступной ему из его среды.

Таким образом, один из способов передачи данных (порожденному) процессу – через среду.

8–11. СЛАЙД: Наблюдение за процессами

Наблюдение за процессами

```
$ ps -f
  UID  PID  PPID   C   STIME  TTY   TIME COMMAND
  user3 4702    1    1   08:46:40 tttyp4  0:00  -sh
  user3 4895  4702   18   09:55:10 tttyp4  0:00  ps -f

$ ksh
$ ps -f
  UID  PID  PPID   C   STIME  TTY   TIME COMMAND
  user3 4702    1    0   08:46:40 tttyp4  0:00  -sh
  user3 4896  4702    1   09:57:20 tttyp4  0:00  ksh
  user3 4898  4896   18   09:57:26 tttyp4  0:00  ps -f

$ exec ps -f
  UID  PID  PPID   C   STIME  TTY   TIME COMMAND
  user3 4702    1    0   08:46:40 tttyp4  0:00  -sh
  user3 4896  4702   18   09:57:26 tttyp4  0:00  ps -f
$
```

Пояснения

Каждому процессу, инициированному в системе, присваивается уникальный идентификационный номер, известный как идентификатор процесса (**PID**). Команда **ps** выводит информацию о процессах, которые выполняются (или спят) в системе в данный момент, в том числе для каждого процесса его PID и PID его родителя (**PPID**). Анализируя значения идентификаторов PID и PPID, можно проследить цепочку родственных процессов для каждого процесса, выполняющегося в системе. Команда **ps** сообщает также, кто является владельцем каждого процесса, с какого терминала запущен каждый процесс и другую полезную информацию.

Обычно команда **ps** задается без опций, что позволяет получить краткий отчет о процессах, связанных только с текущим терминалом пользователя, например:

```
$ ps
PID  TTY   TIME COMMAND
4702 tttyp4 0:00 sh
4894 tttyp4 0:00 ps
```

Из приведенного примера можно видеть, что команда обнаружила только два работающих в системе процесса, связанных с текущим терминалом пользователя, – это shell, **sh**, и команда **ps**. Выведены значения PID двух процессов. Если команда **ps** вызывается с опцией **-f**, как показано на слайде, тогда она генерирует *полный (full)* отчет, который включает еще и значения PPID, и другую дополнительную информацию. Из результатов можно увидеть, что

сама команда **ps -f** выполняется как процесс, порожденный командой **sh**, так как значение его PPID совпадает со значением PID процесса shell.

Следует помнить, что shell – это обычная программа, такая же, как и любая другая команда UNIX. Если в ответ на приглашение текущего POSIX shell'а выполнить команду **ksh**, также будут выполнены этапы **fork** и **exec** и будет создан и начнет исполняться порожденный процесс Korn shell. Если теперь еще раз выполнить команду **ps -f**, можно увидеть, что, как и ожидалось, **ksh** выполняется как процесс, порожденный исходным shell'ом – **sh**, а новая команда **ps** выполняется как процесс, порожденный Korn shell'ом.

В shell'е есть встроенная команда **exec**; она доступна пользователям и может быть использована непосредственно. Если вместо обычной команды **ps -f** ввести и выполнить команду **exec ps -f**, программный код, соответствующий команде **ps**, заместит программный код текущего процесса (**ksh**). Это можно увидеть, так как PID процесса **ps -f** имеет такое же значение, какое имел процесс **ksh**. Когда процесс, соответствующий команде **ps -f**, завершится, значит, завершится процесс, созданный исходным POSIX shell'ом, и на экран будет выведено первичное приглашение исходного POSIX shell'а.

8–12. СЛАЙД: Порожденные процессы и среда

Порожденные процессы и среда

```
export color=lavender
$ ksh          (создается порожденный процесс shell)
$ ps -f
  UID      PID  PPID  C  STIME     TTY      TIME  COMMAND
  user3    4702      1   0  08:46:40 tttyp4    0:00    sh
  user3    4896   4702   1  09:57:20 tttyp4    0:00    ksh
  user3    4898   4896  18  09:57:26 tttyp4    0:00    ps -f
$ echo $color
lavender
$ color=red
$ echo $color
red
$ exit        (порожденный shell завершается)
$ ps -f       (возврат в родительский shell)
  UID      PID  PPID  C  STIME     TTY      TIME  COMMAND
  user3    4702      1   0  08:46:40 tttyp4    0:00    sh
  user3    4895   4702   1  09:58:20 tttyp4    0:00    ps -f
$ echo $color
lavender
```

Пояснения

На слайде показано, что порожденные процессы не могут изменять среду родительского процесса.

```
$ ps -f
  UID      FSID      PID  PPID  C  STIME     TTY      TIME  COMMAND
  user3  default_system  4702      1   0  08:46:40 tttyp4    0:00    -sh
  user3  default_system  4895   4702   1  09:58:20 tttyp4    0:00    ps -f
```

Когда выполняется приведенная выше команда **ps -f**, она обнаруживает, что в данный момент в системе исполняется только процесс, соответствующий входному shell'у, **sh** (и, конечно, процесс **ps**). Как показано на слайде, переменной *color* присваивается значение *lavender* и эта переменная экспортируется в среду. Затем создается порожденный процесс: выполняется команда **ksh**, которая и создает порожденный процесс Korn shell. Следующая команда **ps -f** подтверждает создание нового процесса. Среда родительского процесса была передана порожденному процессу Korn shell, и можно увидеть, что переменная *color* имеет значение *lavender*. Изменим теперь значение переменной *color* – присвоим ей значение *red*. Команда **echo** подтвердит, что в среде порожденного процесса значение переменной *color* изменено. Если теперь завершить порожденный shell и вернуться в родительский shell, можно увидеть, что среда родительского процесса *не* была изменена порожденным процессом, и переменная *color* сохранила свое первоначальное значение *lavender*.

8–13. УПРАЖНЕНИЯ: Среда shell

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Используя подстановку команд, присвойте переменной *today* значение, соответствующее текущей дате.
2. Как легче всего вывести содержимое входного каталога другого пользователя?
3. Присвойте переменной shell'а с именем *MYNAME* значение вашего имени. Как можно увидеть значение этой переменной?
4. Создайте порожденный процесс shell, выполнив команду **sh**. Проверьте значение переменной *MYNAME*. Что произошло? Завершите порожденный shell (используйте **Ctrl**+**d** или **exit**). Теперь родительский процесс знает что-нибудь о переменной *MYNAME*?
5. Выполните в родительском shell'е команду, которая позволит порожденному shell'у увидеть значение переменной *MYNAME*. Как можно увидеть все переменные, унаследованные порожденным shell'ом от родительского?

6. Создайте еще один порожденный shell. Проверьте значение переменной *MYNAME*. Теперь измените значение этой переменной, присвоив ей значение имени вашего коллеги. Где теперь размещена переменная *MYNAME* – в среде или в области локальных данных? Выведите список значений переменных среды. Какое значение имеет переменная *MYNAME*?
7. Теперь удалите переменную *MYNAME* из порожденного shell'a. После этого будет ли переменная *MYNAME* существовать в области локальных данных порожденного shell'a или в его среде? Почему?
8. Завершите порожденный shell и вернитесь в свой входной shell. Продолжает ли существовать в нем переменная *MYNAME*? Почему? Какую команду вы использовали для проверки?
9. Модифицируйте первичное приглашение shell'a так, чтобы выводился текст: **good_day\$**. Что произойдет с этим приглашением, если вы завершите сеанс работы в системе, а затем вновь войдете в систему?

10. Измените первичное приглашение shell'a так, чтобы выводилось ваше входное имя. Например, если вы зарегистрированы в системе под именем **user3**, первичное приглашение должно выглядеть так: **user3\$**. (ПОДСКАЗКА: есть ли переменная среды, значением которой является входное имя пользователя?)

11. Присвойте переменной *dir* значение **/usr/bin/ls**. Как можно использовать эту переменную, чтобы выполнить команду **ls**? Допускает ли такое использование переменной *dir* указание имени каталога или файла в качестве аргумента?

Модуль 9 — Генерация имен файлов

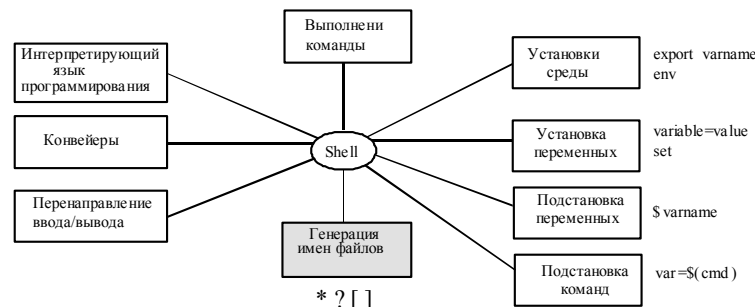
Цели:

Изучив этот модуль, вы сможете:

- Использовать специальные символы для генерации имен файлов в командной строке
- Экономить время при вводе с клавиатуры за счет использования символов генерации имен файлов
- Выбирать такие имена файлов, чтобы использование символов генерации имен было наиболее эффективным

9–1. СЛАЙД: Генерация имен файлов: введение

Генерация имен файлов: введение



- Символы генерации имен файлов интерпретируются shell'ом
- Shell генерирует имена файлов, которые удовлетворяют заданному шаблону
- Генерация имен файлов выполняется *перед* выполнением команды
- Команда обрабатывает сгенерированные имена файлов

Пояснения

Shell предоставляет возможность экономить время при вводе с клавиатуры имен файлов. Эта возможность носит название **генерации имен файлов** или **расширения имен файлов**. Можно найти имена файлов, которые удовлетворяют некоторому правилу (шаблону). Например, найти все имена файлов, которые заканчиваются на `.c`, или все имена файлов, которые начинаются символами `draw`. При вводе имени файла вводятся специальные символы, которые будут заменены одним или несколькими символами из имени файла. Shell расширит введенный шаблон, заменив его *перед* выполнением команды соответствующим именем файла. Поэтому символы генерации имен файлов позволяют экономить время при вводе с клавиатуры.

Возможность генерации имен файлов оказывается очень полезной, поскольку в большинстве приложений поддерживаются определенные соглашения по именованию своих файлов. Зная такие соглашения, пользователь может использовать механизм расширения имен файлов, чтобы получить доступ именно к тем файлам, имена которых удовлетворяют требуемому шаблону. Например, исходные тексты программ, написанных на языке C, размещаются в файлах, имена которых заканчиваются на `.c`, а текстовые процессоры могут использовать в именах файлов, содержащих текстовые документы, окончание `.doc`.

9–2. СЛАЙД: Символы генерации имен файлов

Символы генерации имен файлов

- | | |
|-----|---|
| ? | Соответствует одному любому символу, кроме лидирующей точки |
| [] | Определяет группу символов |
| - | Определяет диапазон символов |
| ! | Задаёт отрицание группы символов |
| * | Соответствует произвольной цепочке символов, в том числе и пустой, за исключением тех, которые начинаются с точки |

Пояснения

Ниже перечислены специальные символы, которые shell использует для генерации имен файлов:

- | | |
|-----|--|
| ? | Соответствует одному любому символу (за исключением лидирующей точки). |
| [] | Задаёт группу символов, из которых в качестве соответствующего символа в имени файла может быть выбран только один (если только этим символом не является лидирующая точка). При задании группы символов между двумя ASCII символами (буквами или цифрами) может быть использовано тире (-), что означает, что в группу входят <i>все</i> символы из указанного диапазона, включая и сами символы. Если группа символов начинается восклицательным знаком (!), это означает, что в качестве соответствующего символа в имени файла может быть использован символ, не совпадающий ни с одним из указанных в группе (т.е. восклицательный знак задаёт <i>отрицание</i> группы символов). |
| * | Соответствует произвольной цепочке символов в имени файла, в том числе и пустой (если только эта цепочка не начинается с точки). |

В дальнейшем каждый из этих символов будет подробно рассмотрен.

9–3. СЛАЙД: Генерация имен файлов и файлы с точкой

Генерация имен файлов и файлы с точкой

- Символы генерации имен файлов *никогда* не создадут имя файла, *начинающееся с точки*
- Для генерации имен файлов, начинающихся с точки, лидирующая точка должна быть указана явно

Пояснения

Файлы с точкой – это файлы, имена которых начинаются символом «точка» (.), как, например, **.profile**, **.kshrc** и **.exrc**. Эти файлы обычно *невидимы* (скрыты); чтобы вывести на экран их имена, надо использовать команду **ls -a**.

Файлы с точкой скрыты также и от символов генерации имен файлов. Поэтому символы генерации имен файлов никогда не создадут имя файла, которое начинается с точки. Если нужно сгенерировать имена файлов, которые начинаются с точки, необходимо в шаблоне явно указать лидирующую точку как часть имени файла.

9–4. СЛАЙД: Генерация имен файлов: символ ?

Генерация имен файлов: символ ?

? соответствует одному любому символу, кроме
лидирующей точки

```
$ ls -a  
. .. .zz abc abcd abcdef abcz bbabb cyz zzayy
```

\$ echo ???	Результат генерации:	echo abc cyz
\$ echo abc?	Результат генерации:	echo abcd abcz
\$ echo ??a??	Результат генерации:	echo bbabb zzayy
\$ echo .??	Результат генерации:	echo .zz
\$ echo ?	Результат генерации:	echo ?

Пояснения

Вопросительный знак соответствует одному любому символу, за исключением лидирующей точки.

Генерация имен файлов выполняется shell'ом перед запуском команды на исполнение. Так, в приведенных на слайде примерах shell сначала генерирует имена файлов, удовлетворяющих заданным шаблонам. Полученные в результате имена файлов передаются в качестве аргументов команде **echo**. Если в каталоге нет файла, имя которого удовлетворяет шаблону, тогда сам шаблон передается команде в качестве аргумента.

ПРИМЕЧАНИЕ: Механизм генерации имен файлов наиболее часто используется в командах манипулирования файлами, таких как **ls**, **more** или **cp**. Для того чтобы проверить, как именно shell выполнит расширение указанного шаблона, особенно когда шаблоны используются в командах удаления, таких как **rm**, полезно использовать команду **echo**. Следует помнить: если файл удаляется, он исчезает.

9–5. СЛАЙД: Генерация имен файлов: символы []

Генерация имен файлов: символы []

[] определяет группу символов, из которой в качестве соответствующего символа выбирается только один

```
$ ls -a
. . . .zz 1G 2G 7G 15G Ant Cat Dog abc abcdef ba cyz

$ echo [abc]??      Результат генерации: echo abc cyz
$ echo [1-9][A-Z]   Результат генерации: echo 1G 2G 7G
$ echo [!A-Z]??     Результат генерации: echo 15G abc cyz
```

Пояснения

Квадратные скобки используются для задания **группы символов**. Группа символов соответствует одному любому символу из тех, которые перечислены в группе.

Восклицательный знак (!), указанный внутри квадратных скобок первым, задает отрицание группы, т.е. включает в группу допустимых для выборки символов те символы, которые *не* указаны внутри квадратных скобок.

Если внутри квадратных скобок между двумя символами указано тире (-), группа символов включает в себя все символы из ASCII последовательности (см. **ascii(5)**) начиная от первого указанного в скобках символа и кончая вторым, включая и сами символы. Таким образом, группы [!123456789] и [!1-9] соответствуют одному любому символу, кроме цифр от 1 до 9 (включая сами эти цифры).

Лидирующая точка (.) не может быть задана группой символов.

9–6. СЛАЙД: Генерация имен файлов: символ *

Генерация имен файлов: символ *

- * Соответствует любой цепочке символов за исключением тех, которые начинаются с точки (.)

```
$ ls -a
. .. .profile ab.dat abcd.dat abcde abcde.data

$ echo *
Результат: echo ab.dat abcd.dat abcde abcde.data
$ echo .*
Результат: echo . .. .profile
$ echo *.dat
Результат: echo ab.dat abcd.dat
$ echo *e
Результат: echo abcde
```

Пояснения

Символ звездочки (*) соответствует строке, состоящей из произвольного количества символов, в том числе и равного нулю.

Как и во всех других способах генерации имен файлов, звездочка (*) не соответствует строке символов, начинающейся с точки.

9–7. СЛАЙД: Генерация имен файлов: заключение

Генерация имен файлов: заключение

```
$ ls -a
.          Abc      e35f
..         Abcd     efg
.test1     abc      fe3f
.test2     abcdemf  fe3fg
```

Для приведенного здесь содержимого каталога создайте шаблоны, позволяющие получить имена файлов, которые:

- содержат *только* пять символов
- содержат, *по крайней мере*, пять символов
- начинаются с буквы **a** или **A**
- имеют, *по крайней мере*, четыре символа и начинаются с **a** или **A**
- заканчиваются последовательностью **e**, любая цифра и **f**
- начинаются с точки
- начинаются с точки, за исключением **.**
- начинаются с точки, за исключением **.** и **..**

Пояснения

На слайде приведено содержимое некоторого каталога. Определите, каким образом надо задать шаблоны генерации имен файлов, чтобы получить имена файлов, удовлетворяющих заданным требованиям.

Указанные файлы находятся в подкаталоге **filegen** вашего *HOME* каталога.

9–8. УПРАЖНЕНИЯ: Генерация имен файлов

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Перейдите в свой *HOME* каталог. Выполните команду **ls *** и объясните ее результаты.
2. Если при выполнении команды **echo ???XX** получен результат: **???XX**, что это означает?
3. Перейдите в свой *HOME* каталог. Какую команду здесь нужно выполнить, чтобы получить следующие результаты:
 - a. Вывести все имена файлов, заканчивающиеся символами **.c**.
 - b. Вывести все имена файлов, которые заканчиваются символами **.c** и начинаются словом **mod**.
 - c. Вывести все имена файлов, которые в любом месте содержат слово **file**.
 - d. Вывести все имена файлов, которые заканчиваются или символами **.c**, или **.f**, или **.p**.
4. Создайте каталог **c_source**. Используя символы генерации имен файлов, переместите из текущего каталога в каталог **c_source** все файлы, имена которых оканчиваются на **.c**.

5. Создайте в своем *HOME* каталоге каталог **dir_1**. Что произойдет, если вы выполните команду: **cd dir***?

6. Вернитесь в свой *HOME* каталог и создайте в нем каталоги **dir_2**, **dir_3** и **dir_4**. Теперь снова попробуйте выполнить команду **cd dir***. Объясните, что случилось.

7. С помощью команды **touch** (синтаксис: **touch имя_файла**) создайте файлы, для которых будут выполняться перечисленные ниже требования:
Шаблону **?xx** соответствует имя только ОДНОГО файла.
Шаблону **? .xx** соответствуют имена только ДВУХ файлов.
Шаблону ***xx** соответствуют имена только ТРЕХ файлов.
Шаблону **xx.??** соответствует имя только ОДНОГО файла.
Шаблону **xx.*** соответствуют имена только ДВУХ файлов.
Используйте команду **echo**, чтобы проверить полученные результаты.

8. Используйте команду **rm** так, чтобы за один раз удалить все файлы, созданные в предыдущем упражнении (ПОДСКАЗКА: для большей уверенности можно использовать команду в виде **rm -i**)

Модуль 10 — Экранирование

Цели:

Изучив этот модуль, вы сможете:

- Использовать механизм экранирования для отмены особого смысла специальных символов, задаваемых в командной строке

10–1. СЛАЙД: Экранирование: введение

Экранирование: введение

- Многие (специальные) символы имеют для shell'а особый смысл, например:
 - пробел
 - возврат каретки
 - \$
 - #
 - *
 - < >
- Экранирование позволяет отменить особый смысл специальных символов

Пояснения

В системе UNIX имеется немало количество символов (называемых **специальными**), которые имеют для shell'а особый смысл. Например, пробел является разделителем между командой и аргументами. Возврат каретки сообщает shell'у о необходимости обработки введенной строки, а символ \$ используется для подстановки значения переменной или результата выполнения команды.

Однако могут возникнуть ситуации, когда не нужно, чтобы shell выполнял особую обработку тех или иных специальных символов. Эти символы в данном контексте должны рассматриваться как обычный текст. Поэтому система UNIX должна предоставить механизм, позволяющий отменить особый смысл тех или иных специальных символов. Этот механизм известен как механизм **использования кавычек**, или **экранирование**.

10–2. СЛАЙД: Символы экранирования

Символы экранирования

Обратный слеш	\
Одиночные кавычки	'
Двойные кавычки	"

Пояснения

Символ *обратный слеш* (\) отменяет особый смысл специального символа, следующего непосредственно за ним.

Одиночные кавычки (') также отменяют особый смысл специальных символов. Все специальные символы, заключенные в одиночные кавычки, экранируются. Сам символ одиночной кавычки не может быть экранирован, так как одиночные кавычки являются парными символами: вторая кавычка всегда рассматривается как парная и закрывает строку.

ПРИМЕЧАНИЕ: Одиночные кавычки (') не эквивалентны символу тупого удара (или обратному апострофу, `).

Двойные кавычки (") обладают меньшей силой воздействия. Большинство специальных символов, заключенных в двойные кавычки, теряют свой особый смысл. Исключение составляют символы \$ (когда он используется для подстановки переменной или команды), обратный слеш (\) и двойная кавычка ("), которая необходима для того, чтобы закрыть строку (этот символ также является парным). Внутри двойных кавычек можно использовать обратный слеш, чтобы отменить специальный смысл символов \$ или ".

10–3. СЛАЙД: Экранирование: символ \

Экранирование: символ \

Синтаксис:

\ Отменяет специальный смысл следующего за ним символа

Примеры:

```
$ echo the \\ escapes the next character
the \ escapes the next character
$ color=red\ white\ and\ blue
$ echo the value of\ $color is $color
the value of $color is red white and blue
$ echo one two \
> three four
one two three four
```

Пояснения

Обратный слеш всегда отменяет специальный смысл следующего непосредственно за ним символа, без каких-либо исключений.

10–4. СЛАЙД: Экранирование: символ '

Экранирование: символ '

Синтаксис:

' Отменяет специальный смысл всех символов, заключенных в
одиночные кавычки

Примеры:

```
$ color='red white and blue'
$ echo 'the value of \${color} is ${color}'
the value of \${color} is ${color}
$ echo 'the value of ${color} is' ${color}
the value of ${color} is red white and blue
$ echo 'this doesn't work'
> [Ctrl] + [C]
$ echo '*****'
*****
```

Пояснения

Одиночные кавычки отменяют особый смысл *всех* специальных символов, кроме самой одиночной кавычки.

10–5. СЛАЙД: Экранирование: символ "

Экранирование: символ "

Синтаксис:

" Отменяет специальный смысл всех символов, заключенных в двойные кавычки, кроме \, \${*имя_переменной*}, \$(*команда*)

Примеры:

```
$ color="red white and blue"
$ echo "the value of \${color} is ${color}
the value of ${color} is red white and blue
$ cur_dir="$LOGNAME"
$ echo $cur_dir
user3 - your current directory is /home/user3/tree
$ echo "they're all here, \\", ', \" "
they're all here, \, ', "
```

Пояснения

Двойные кавычки не обладают такой силой экранирования, как одиночные. Они позволяют отменить специальный смысл *большинства* символов, но не всех. Имеющиеся исключения, в частности, позволяют выполнять подстановку переменных *\${имя_переменной}* и подстановку команд *\$(команда)*.

ПРИМЕЧАНИЕ: Подстановка команд в Bourne shell'е обозначается с помощью знака тупого удара (``), например, ``pwd``; эта операция дает такой же результат, как и \$(pwd) в POSIX shell'е (знак тупого удара можно использовать также и в POSIX shell'е). Знак тупого удара сохраняет свой специальный смысл внутри двойных кавычек.

Могут возникнуть ситуации, когда нужно отменить специальный смысл таких символов и тогда, когда они используются внутри двойных кавычек. Для этих целей используется обратный слеш (\), который также сохраняет свой специальный смысл внутри двойных кавычек. Использование внутри двойных кавычек символа \ перед специальными символами позволяет отменить особый смысл таких символов.

ПРИМЕЧАНИЕ: Все перечисленные механизмы экранирования могут применяться в одной командной строке.

10–6. СЛАЙД: Экранирование: заключение

Экранирование: заключение

Символы	Действие
Обратный слеш	Отменяет специальный смысл символа, следующего за \
Одиночные кавычки	Отменяет специальный смысл всех символов, заключенных в ' '
Двойные кавычки	Отменяет специальный смысл всех символов, заключенных в " ", кроме \, <code>\${имя_переменной}</code> , <code>\$ (команда)</code>

Пояснения

На слайде приведены все символы экранирования и способы их воздействия.

10–7. УПРАЖНЕНИЯ: Экранирование

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Введите команду **echo** таким образом, чтобы она вывела на экран следующий текст:

```
$1 million dollars    ...    and that's a bargain !
```

2. Присвойте переменной с именем *long_string* следующее значение:

```
$1 million dollars    ...    and that's a bargain !
```

Выведите значение переменной *long_string*, чтобы убедиться в правильности присваивания.

3. Что произойдет при выполнении следующих команд?

```
$ banner good day  
$ banner 'good day'
```

Сколько аргументов указано в каждой приведенной выше командной строке?

ПРИМЕЧАНИЕ: Пользователям системы Linux, как правило, команда **banner** недоступна. К тому же, версия этой команды в системе Linux не допускает ее использования для вывода на экран. Если пользователи системы Linux попытаются выполнить приведенные выше команды в своей системе, они увидят только сообщения об ошибке.

4. Установите в качестве первичного приглашения строку: *Way to go ваше_входное_имя \$*

5. Как можно вывести на экран следующее сообщение?

```
Exercises #1, #2, and #3 are now complete.
```

6. Предположим, что переменная *abc* не определена. Что произойдет при выполнении следующей команды?

```
echo '$abc'
```

Что произойдет при выполнении следующей команды?

```
echo "$abc"
```

7. Используя команду **touch**, создайте файл, имя которого состоит:

- из слов `White Space`,
- из четырех пробелов,
- из трех пробелов.

Как будут выглядеть такие имена файлов при выводе содержимого каталога? В каком виде следует вывести содержимое каталога, чтобы можно было определить, сколько пробелов – 3 или 4 – использовано в имени файла?

Модуль 11 — Перенаправление ввода/вывода

Цели:

Изучив этот модуль, вы сможете:

- Изменять получателя результатов выполнения команд системы UNIX
- Изменять получателя сообщений об ошибках, генерируемых командами системы UNIX
- Изменять источник ввода для команд системы UNIX
- Определять фильтр
- Использовать некоторые простые фильтры, такие как **sort**, **grep** и **wc**

11-1. СЛАЙД: Перенаправление ввода/вывода: введение



Пояснения

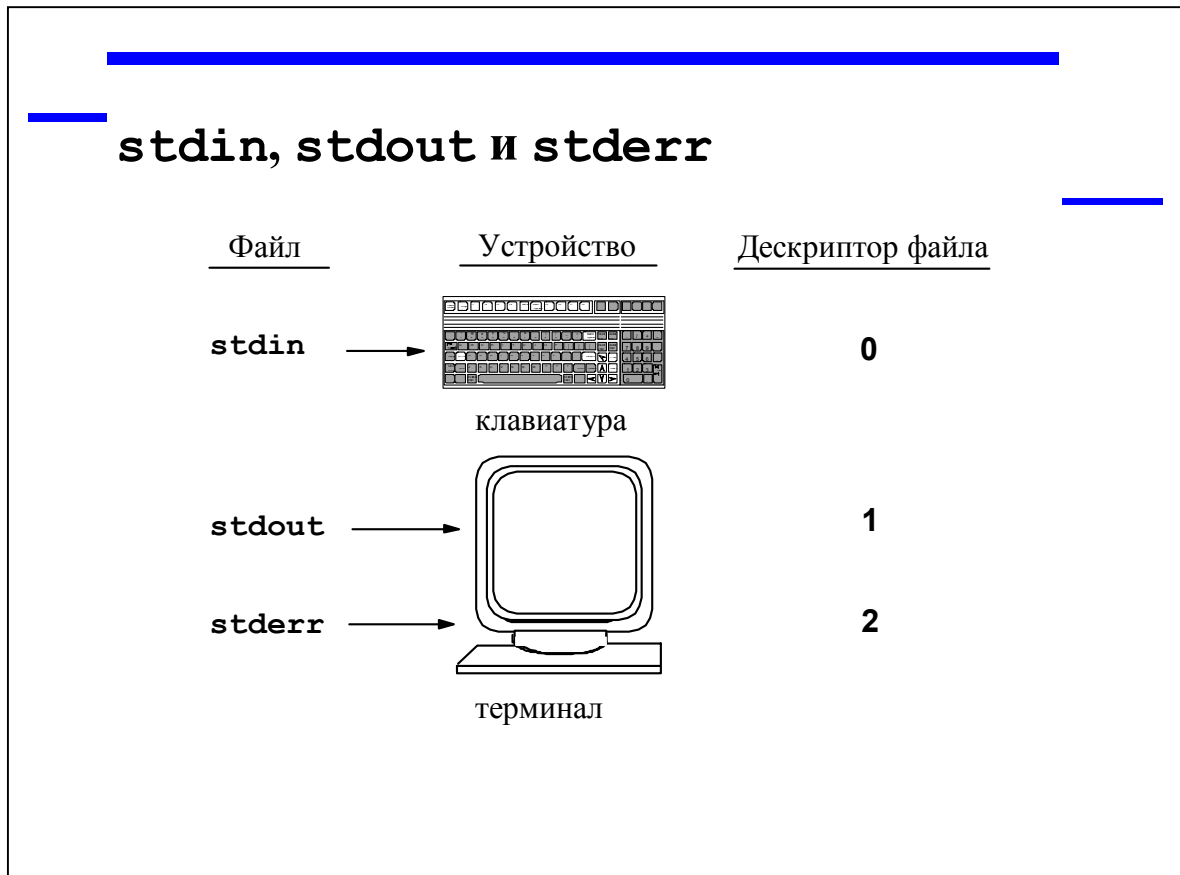
Еще одной характерной особенностью shell'a является его способность выполнять перенаправление ввода и/или вывода команд. Большое количество команд выводят результаты своей работы на терминал; примерами таких команд являются **date**, **banner**, **ls**, **who** и т.д. Другие команды вводят необходимую информацию с клавиатуры, например, **mail**, **write**, **cat**.

В системе UNIX *все* трактуется через понятие файл, включая терминал и клавиатуру. **Перенаправление вывода** позволяет выводить результаты работы команды не на терминал, а в некоторый файл. Аналогично, **перенаправление ввода** позволяет команде вводить исходные данные не с клавиатуры, а из некоторого файла.

Перенаправление вывода полезно, например, для перехватывания вывода команды с целью протоколирования или для последующей обработки. Перенаправление ввода позволяет вместо того, чтобы во время работы команды (например, **mail**) интерактивно вводить необходимый текст без возможностей редактирования, использовать редактор для создания файла и затем передать этот файл команде.

В этом модуле рассматриваются возможности перенаправления ввода/вывода и представляются некоторые фильтры системы UNIX. Фильтры – это утилиты, которые могут быть использованы для дальнейшей обработки содержимого файла.

11–2. СЛАЙД: `stdin`, `stdout` и `stderr`



Пояснения

При каждом запуске `shell`'а автоматически открываются три файла, сразу же доступных для использования. Эти файлы имеют собственные названия – `stdin`, `stdout` и `stderr`.

Файл `stdin` – это файл, из которого `shell` читает исходные данные. Обычно этот файл называют **стандартным вводом**. Этому файлу соответствует дескриптор файла, равный `0`, и обычно этот файл назначается на клавиатуру. Поэтому когда `shell` требует ввода данных, они (данные) должны быть введены с клавиатуры.

К командам, которые читают данные из стандартного ввода, относятся `mail`, `write` и `cat`. Каждая из этих команд характеризуется следующим: после того как при вводе команды в конце командной строки будет нажата клавиша `Return`, команда переходит в состояние ожидания – она ждет, когда с клавиатуры будут введены необходимые данные, подлежащие обработке. Ввод данных завершается нажатием клавиш `Return` `Ctrl` + `d`.

Файл `stdout` – это файл, в который `shell` записывает нормальные результаты работы. Обычно этот файл называют **стандартным выводом**. Этому файлу соответствует дескриптор файла, равный `1`, и обычно этот файл назначается на терминал. Поэтому когда `shell` выполняет вывод результатов, они отображаются на экране терминала.

Большинство команд системы UNIX генерируют стандартный вывод. К таким командам относятся `date`, `banner`, `ls`, `cat` и `who`.

Файл **stderr** – это файл, в который shell записывает сообщения об ошибках. Обычно этот файл называют **стандартным протоколом**. Этому файлу соответствует дескриптор файла, равный **2**. Как и **stdout**, файл **stderr** обычно назначен на терминал. Перенаправление файла **stderr** можно выполнять независимо от файла **stdout**.

Большинство команд системы UNIX генерируют сообщения об ошибке, если выполняется неправильный вызов команды. Чтобы увидеть пример сообщения об ошибке, можно ввести следующее: **cp** Return. На экране появится сообщение о правилах использования команды **cp**; в действительности же это сообщение было выведено в стандартный протокол.

Главная цель данного модуля – показать, как можно изменять установленные по умолчанию назначения для файлов **stdin**, **stdout** и **stderr**; это позволит читать входные данные из других файлов, а не с клавиатуры, и записывать результаты работы (и сообщения об ошибках) не на экран терминала, а куда-нибудь еще.

11–3. СЛАЙД: Перенаправление ввода: <

Перенаправление ввода: <

Любая команда, которая читает данные из стандартного ввода (**stdin**), может перенаправить его, чтобы читать данные из любого другого файла

Пример:

```
$ cat remind
Your mother's birthday is November 29
$ mail user3 < remind
$ mail
From user3 Mon July 15 11:30 EDT 1993
Your mother's birthday is November 29
?d
$
```

Пояснения

Для команд, которые читают исходные данные из стандартного ввода, можно выполнить перенаправление ввода, в результате которого команда будет вводить данные не с клавиатуры, а из какого-либо другого файла. Перенаправление ввода часто используется для команды **mail**. С помощью какого-либо редактора можно записать в файл текст сообщения, которое требуется отправить по электронной почте. После этого можно перенаправить стандартный ввод команды **mail** так, чтобы она использовала текст из этого файла. Такой подход очень удобен, если требуется отправить длинное сообщение или если нужно сохранить текст сообщения с тем, чтобы к нему можно было обратиться в последующем.

Команды, которые читают данные со стандартного входа, обладают следующей особенностью: после того как при вводе команды в конце командной строки будет нажата клавиша **Return**, команда переходит в состояние ожидания – она ждет, когда с клавиатуры будут введены необходимые данные. Ввод данных завершается нажатием клавиш **Return** **Ctrl** + **d**.

Многие команды, читающие данные со стандартного ввода, допускают также задание имени файла в качестве аргумента командной строки. Заданные таким способом файлы также обрабатываются командой. Хорошим примером такой команды является команда **cat**. Команда **cat** выводит на экран текст, который она вводит непосредственно с клавиатуры, содержимое файлов, заданных в качестве аргументов, или, наконец, содержимое файла, указанного для перенаправления стандартного ввода.

Чтение из <code>stdin</code>	Работа с аргументами командной строки	Перенаправление ввода
\$ <code>cat</code> Return <i>ввод текста</i> Ctrl + d для завершения ввода <i>Введенный текст отображается на экране</i>	\$ <code>cat</code> <i>имя_файла</i> <i>на экран выводится содержимое файла</i>	\$ <code>cat</code> < <i>имя_файла</i> <i>на экран выводится содержимое файла</i>

ПРИМЕЧАНИЕ: Перенаправление ввода не изменяет содержимое входного файла.

11–4. СЛАЙД: Перенаправление вывода: > и >>

Перенаправление вывода: > и >>

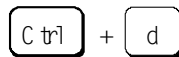
Любая команда, которая выводит результаты в стандартный вывод (**stdout**), может перенаправить его для вывода результатов в любой файл

Примеры:

создание/перезапись

```
$ date > date.out  
$ date > who.log  
$ cat > cat.out
```

ввод текста



создание/добавление

```
$ ls >> ls.out  
$ who >> who.log  
$ ls >> who.log
```

Пояснения

Многие команды выводят свои сообщения на экран. Перенаправление вывода позволяет перехватить результаты работы команды и записать их в текстовый файл.

Если в командной строке указан символ перенаправления вывода (>), за которым следует имя файла, стандартный вывод команды вместо экрана терминала будет направлен в указанный файл. Если файл с таким именем до вызова команды не существовал, он будет автоматически создан. Если же файл с указанным именем *существовал* до вызова команды, тогда этот файл будет *перезаписан*; вывод команды полностью заменит предыдущее содержимое файла.

В том случае, когда нужно добавить новые данные в конец файла без его перезаписи, можно использовать перенаправление вывода в режиме добавления (>>). Следует обратить внимание на то, что между двумя символами > *не должно* быть пробела. Такой режим также приведет к созданию файла, если он не существовал до выполнения команды.

ВНИМАНИЕ!

Shell не может один и тот же файл одновременно открыть и для перенаправления ввода, и для перенаправления вывода. Следовательно, единственным ограничением на одновременное использование перенаправления ввода и вывода является условие: входной и выходной файл *должны* быть разными. В противном случае исходное содержимое файла будет потеряно, и перенаправление вывода даст неверный результат. Пример: выполнение команды **cat f1 f2 > f1** приведет к потере содержимого файла **f1**.

11–5. СЛАЙД: Перенаправление протокола: 2> и 2>>

Перенаправление протокола: 2> и 2>>

Любая команда, которая выводит сообщения об ошибках в стандартный протокол (**stderr**), может перенаправить его для вывода сообщений в любой файл

Примеры:

```
$ cp 2> cp.err      создание/перезапись
$ cp 2>> cp.err     создание/добавление
$
$ more cp.err
Usage: cp [-f|-i] [-p] source_file target_file
      cp [-f|-i] [-p] source_file ...target_directory
      cp [-f|-i] [-p] -R|-r source_directory...target_directory
Usage: cp [-f|-i] [-p] source_file target_file
      cp [-f|-i] [-p] source_file ... target_directory
      cp [-f|-i] [-p] -R|-r source_directory...target_directory
```

Пояснения

Если команда вводится с ошибками и shell не может интерпретировать ее должным образом, генерируется сообщение об ошибке. И хотя сообщения об ошибках отображаются на экране, в действительности они передаются в другой файл, отличный от файла, в который команда выводит обычные сообщения. Сообщения об ошибках выводятся в поток, получивший название **stderr**, или стандартный протокол. Поток **stderr** определяется дескриптором файла **2**.

Поэтому, указывая перенаправление вывода сообщений об ошибках, необходимо указать, что требуется перехватить сообщения, выводимые в поток с дескриптором **2**. Для обозначения перенаправления **stderr** используется конструкция **2>**. Между цифрой **2** и символом **>** не должно быть пробела. Как и при перенаправлении стандартного вывода, такой способ перенаправления стандартного протокола приведет к созданию файла, если он не существует, или к перезаписи существующего файла. С помощью конструкции **2>>** можно добавить новое сообщение в конец существующего файла.

Рассматриваемый механизм весьма полезен с точки зрения администрирования системы. Достаточно часто интерес представляют только те ситуации, в которых программы работают неправильно (дают сбой) или решаются какие-либо экспериментальные задачи. Так как сообщения об ошибках отделены от обычных сообщений, можно легко перехватить их и вести специальный файл протокола, в котором будут регистрироваться проблемы, возникшие при выполнении программы.

11–6. СЛАЙД: Что такое фильтр?

Что такое фильтр?

- Читает из стандартного ввода *и* выводит в стандартный вывод
- Выполняет фильтрацию входного потока или содержимого файла
- Выводит результаты на экран, не изменяя входной поток или файл
- Обрабатывает результаты других команд, если они используют перенаправление вывода

Примеры: **cat**, **grep**, **sort**, **wc**

Пояснения

Выше было показано, как выполняется перенаправление ввода или вывода команды. Некоторые команды вводят исходные данные из стандартного ввода *и* выводят свои результаты в стандартный вывод. Такие команды называются **фильтрами**. Фильтры никогда не изменяют содержимое файла, который они обрабатывают. Результат обработки файла фильтром обычно направляется на терминал.

Фильтры бывают очень полезны для некоторой специфической обработки файла, например, для подсчета количества строк в файле (**wc**), упорядочения строк по алфавиту (**sort**) или поиска строк, удовлетворяющих некоторому образцу (**grep**).

Кроме того, фильтры могут быть использованы для последующей обработки вывода *любой* команды. Так как фильтры могут обрабатывать файлы, а вывод команд может быть перенаправлен в файл, эти две операции могут быть объединены для создания мощной и гибкой обработки результатов любой команды. Так как большинство фильтров выводят свои результаты в стандартный вывод, их можно перенаправить в файл, который в дальнейшем может быть обработан еще одним фильтром.

11–7. СЛАЙД: **wc** – подсчет количества слов

wc — ПОДСЧЕТ КОЛИЧЕСТВА СЛОВ

Синтаксис:

wc [-lwc] [*файл...*] Подсчитывает количество строк, слов
и символов в файле

Примеры:

```
$ wc funfile    funfile рассматривается как аргумент команды

116 529 3134 funfile
$
$ wc -l funfile
116 funfile
$
$ ls > ls.out
$
$ wc -w ls.out    подсчитывает количество элементов в каталоге
72 ls.out
```

Пояснения

Команда **wc** подсчитывает количество строк, слов и символов, прочитанных со стандартного ввода или из файла. У команды есть опции **-l**, **-w** и **-c**, которые влияют на результат, выводимый на экран. Если указана опция **-l**, тогда на экран выводится количество строк; если указана опция **-w** – количество слов; наконец, указание опции **-c** приводит к выводу на экран количества символов. В командной строке может быть указано несколько опций; в этом случае независимо от порядка записи опций, а так же если опции не указаны совсем, результаты выводятся на экран в порядке количество строк – количество слов – количество символов.

Так как команда **wc** может читать данные со стандартного ввода и выводит результаты своей работы на стандартный вывод, она является *филтром*. Команда **wc** при обработке файла не изменяет его содержимое, так как выводит результаты своей работы на экран.

Дополнительные примеры

```
$ wc Return    обрабатывает данные, вводимые со стандартного ввода
ab cde
fghijkl
mno pqr stuvwxyz
Ctrl + d
3 6 32
$ wc < funfile    стандартный ввод заменяется файлом funfile
```

105 718 3967	<i>имя файла не выводится</i>
\$ wc -w funfile	<i>обрабатывается файл funfile, заданный аргументом команды</i>
718 funfile	<i>выводится имя файла</i>

Как показано в первом примере, команда **wc** читает данные со стандартного ввода. Это позволяет при вызове команды **wc** использовать перенаправление ввода, в результате чего команда будет читать данные не со стандартного ввода, а из указанного файла. Кроме того, команда **wc** может также обрабатывать файлы, имена которых передаются как аргументы командной строки. В этом случае на экран выводится имя обрабатываемого файла.

11–8. СЛАЙД: **sort** – алфавитная или числовая сортировка

sort — алфавитная или числовая сортировка

Синтаксис:

sort [-ndutX] [-k номер_поля] [файл...] Сортирует строки

Примеры:

\$ sort funfile *funfile рассматривается как аргумент команды*

```
$ tail -1 /etc/passwd
user3:xyzbkd:303:30:studentuser3:/home/user3:/usr/bin/sh
  1      2      3  4      5      6      7
$ sort -nt: -k 3 < /etc/passwd
```

\$ who > whoson

\$ sort whoson *сортирует в алфавитном порядке список работающих в системе пользователей*

\$ sort -u -k 1,1 whoson *сортирует список работающих пользователей с исключением повторяющихся строк*

Пояснения

Команда **sort** обладает большими возможностями и гибкостью. Ее можно использовать для упорядочения строк одного или нескольких файлов в алфавитном порядке или по числовому значению. В строке может быть выбрано некоторое конкретное поле, по значению которого и выполняется сортировка. Команда **sort** также является фильтром, поскольку вводит данные со стандартного ввода; кроме этого, она может сортировать и файлы, имена которых ей передаются аргументами командной строки.

Для того чтобы указать, какой тип сортировки следует выполнить, в команде могут быть использованы следующие опции:

Опция команды	Тип сортировки
отсутствует	лексикографическая (коды ASCII)
-d	словарная (все символы, кроме букв, цифр и пробелов, игнорируются)
-n	числовая
-u	уникальная (исключаются повторяющиеся строки)

По умолчанию, символом-разделителем полей в строке является пробельный символ – пробел или знак табуляции. С помощью опции **-tX** можно указать любой другой

разделитель полей; здесь X представляет символ-разделитель. Так как двоеточие (:) не имеет в shell'е специального смысла, этот символ часто используется в качестве символа-разделителя полей в файле.

После того, как задан символ-разделитель полей, с помощью опции **-kn**, где n – номер поля, команде **sort** можно указать, по значению какого поля следует сортировать строки файла. Для команды **sort** нумерация полей начинается с единицы.

В команде **sort** используются и другие опции, позволяющие выполнять более сложные сортировки. Подробное описание команды можно найти в статье **sort(1)** *Справочника HP-UX*.

Дополнительные примеры

```
$ sort Return сортирует строки, вводимые со стандартного ввода
```

```
mmmmm
```

```
xxxx
```

```
aaaa
```

```
Ctrl + d
```

```
aaaa
```

```
mmmmm
```

```
xxxx
```

```
$ sort < funfile стандартный ввод заменяется файлом funfile
```

Как показано в первом примере, команда **sort** может читать данные со стандартного ввода. Следовательно, используя перенаправление ввода, можно сортировать строки любого файла.

ПРИМЕЧАНИЕ: Shell не может одновременно открыть один и тот же файл для перенаправления и ввода, и вывода. Однако в команде **sort** есть опция **-o выходной_файл**, которая предписывает выводить результаты сортировки не на стандартный вывод, а в указанный файл. В этом случае имя файла для результата может совпадать с именем входного файла.

Пример: команда **sort -o whoson -d whoson** выполнит словарную сортировку внутри файла **whoson**.

11–9. СЛАЙД: **grep** – поиск по модели

grep — ПОИСК ПО МОДЕЛИ

Синтаксис:

```
grep [-cinv] [-e] модель [-e модель] [файл...]  
grep [-cinv] -f файл_списка_моделей [файл...]
```

Примеры:

```
$ grep user /etc/passwd  
$ grep -v user /etc/passwd  
$ grep -in -e like -e love funfile  
  
$ who > whoson  
$ grep rob whoson
```

Пояснения

Команда **grep** – очень полезная команда. В качестве первого аргумента она использует некоторую модель поиска (обычно заключается в одиночные кавычки), а все остальные аргументы задают имена файлов. Можно сделать так, что команда **grep** будет осуществлять поиск по нескольким моделям. Для этого в вызове команды перед каждой моделью поиска необходимо указать опцию **-e**, или же указать опцию **-f**, за которой следует имя файла, содержащего модели поиска. Команда **grep** выводит на экран строки, которые сопоставляются с указанной моделью (или с одной из моделей, если их несколько).

Наиболее часто используются следующие опции команды **grep**: **-n**, **-v**, **-i** и **-c**.

- c** вывести только количество строк, сопоставляющихся с моделью
- i** игнорировать регистр букв в модели
- n** вывести перед каждой строкой ее номер
- v** вывести строки, которые *не сопоставляются* с моделью

Команда **grep**, как и любой другой фильтр, при отсутствии аргументов командной строки читает данные со стандартного ввода и выводит результат в стандартный вывод.

Команда **grep** может выполнять весьма сложный поиск. Прежде всего, нужно построить модель текста, который нужно искать. Такая модель называется *регулярным выражением*. Ниже приводятся некоторые специальные символы, используемые при записи регулярных выражений (подробное описание находится в статье **regex (5) Справочника**).

^	выполнять поиск текста по модели в самом начале строки
\$	выполнять поиск текста по модели в самом конце строки
.	соответствует одному любому символу
*	предшествующий элемент модели должен быть повторен нуль или большее число раз
[]	класс символов; определяет совокупность символов
[-]	знак тире (–) задает диапазон символов
[^]	инвертирует процесс выбора

Для того чтобы shell не обрабатывал свои специальные символы, использованные в записи модели, рекомендуется заключать модель поиска в одиночные кавычки.

Система Linux



Версия команды **grep** в GNU/Linux имеет несколько дополнительных опций. Кроме того, некоторые опции имеют формат, отличный от формата опции в HP-UX. Для того чтобы увидеть все дополнительные опции, необходимо обратиться к соответствующей статье Справочника.

11–10. СЛАЙД: Перенаправление ввода/вывода: заключение

Перенаправление ввода/вывода: заключение

<code>команда < файл</code>	Перенаправление стандартного ввода команды на файл
<code>команда > файл</code>	Перенаправление стандартного вывода команды на файл
<code>команда >> файл</code>	Перенаправление стандартного вывода команды на файл в режиме добавления
<code>команда 2> файл.err</code>	Перенаправление стандартного протокола команды на файл.err
фильтр	Команда, которая вводит из stdin и выводит в stdout
wc	Подсчет количества строк, слов и символов
sort	Сортировка строк в алфавитном или числовом порядке
grep	Поиск строк, удовлетворяющих модели

Пояснения

11–11. УПРАЖНЕНИЯ: Перенаправление ввода/вывода

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Выполните перенаправление результатов команды **date** в файл с именем **date.out**, находящийся в вашем *HOME* каталоге.
2. Добавьте в файл **date.out** результаты команды **ls**. Посмотрите содержимое файла **date.out**. Что вы заметили?
3. Используя перенаправление ввода, пошлите электронной почтой файл **date.out** вашему коллеге.
4. Используя команду **cat** и перенаправление вывода, создайте в своем *HOME* каталоге два коротких файла – **f1** и **f2**.
5. Используя команду **cat**, посмотрите содержимое созданных в предыдущем упражнении файлов. С помощью этой же команды создайте новый файл **f.join**, содержащий информацию из этих файлов – **f1** и **f2**. Видите ли вы что-нибудь на экране?

6. Используя команду **cat**, выведите на экран содержимое файлов **f1**, **f2** и **f.new**. ПРИМЕЧАНИЕ: файл **f.new** *НЕ ДОЛЖЕН* существовать. Что вы увидите на экране? Можно ли определить, какие сообщения выводились в стандартный вывод, а какие – в стандартный протокол?

7. Используя команду **cat**, снова выведите на экран содержимое файлов **f1**, **f2** и **f.new**. ПРИМЕЧАНИЕ: файл **f.new** опять *НЕ ДОЛЖЕН* существовать. На этот раз перехватите сообщения об ошибках, генерируемые командой, и направьте их в файл **f.error**. Что вы увидите на экране? Создан ли новый файл? Проверьте его содержимое.

8. Используйте опять команду **cat** для вывода содержимого файлов **f1**, **f2** и **f.new**. ПРИМЕЧАНИЕ: файл **f.new** и на этот раз *НЕ ДОЛЖЕН* существовать. Теперь перехватите сообщения, выводимые в стандартный вывод, и направьте их в файл **f.good**, и *В ЭТОЙ ЖЕ КОМАНДЕ* направьте вывод диагностических сообщений в файл **f.bad**. Что вы увидите на экране? Созданы ли какие-либо новые файлы? Проверьте их содержимое.

9. Выполните команду **cp** без аргументов. Что произошло? Попробуйте перенаправить стандартный вывод команды в файл **cp.error**. Что произошло? Что нужно сделать, чтобы перенаправить в файл сообщения об ошибках? Выводит ли команда **cp** какие-либо результаты в стандартный вывод?

10. Выведите на экран содержимое файла **/etc/passwd**, упорядочив его по входному имени пользователя.
11. Выведите содержимое файла **/etc/passwd**, упорядоченное по третьему полю. Что получилось? Снова выполните эту команду, указав в ней числовое упорядочение по третьему полю. Есть ли отличия в результатах выполнения этих двух команд?
12. Выберите из файла **/etc/passwd** все строки, в которых встречается строка *stud*. Сохраните результаты в файле **grepped**. С помощью фильтра определите, сколько строк файла **/etc/passwd** содержат строку *stud*.
13. Используя перенаправление вывода и фильтры, определите, сколько пользователей в данный момент работает в системе.
14. Сколько пользователей зарегистрировано в вашей системе? Какие команды вы использовали для ответа на поставленный вопрос? (ПОДСКАЗКА: Каждому пользователю, зарегистрированному в системе, соответствует одна строка в файле **/etc/passwd**)

15. Упорядочите файл **names** из вашего входного каталога и сохраните результат в файле **names.sort**. Снова отсортируйте этот файл, но теперь в обратном порядке, и сохраните результат в файле **names.rev**. Какие команды вы использовали? Обратитесь к статье справочника **sort(1)** и найдите опцию, которая позволит команде сохранить результаты своей работы в исходном файле **names**.
16. Направьте вывод команды **banner** на терминал вашего коллеги. ПОДСКАЗКА: Вы знаете, какой файл устройства ассоциирован с терминалом вашего коллеги? Если вы получите сообщение **Permission denied**, что это значит?

Система Linux



Так как в системе Linux команда **banner** недоступна, используйте в данном упражнении команду **echo** и направьте ее вывод на терминал вашего коллеги.

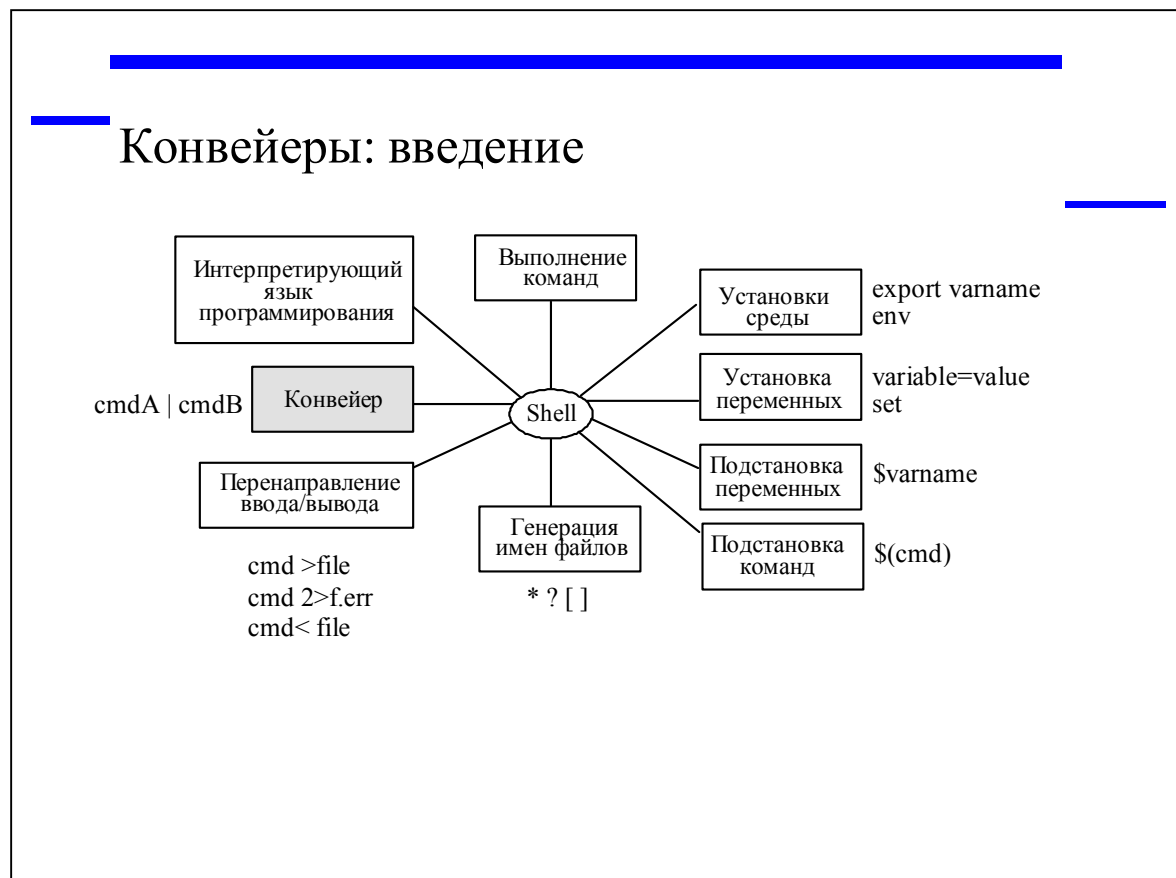
Модуль 12 — Конвейеры

Цели:

Изучив этот модуль, вы сможете:

- Объяснить использование конвейеров
- Создавать конвейеры, направляя стандартный вывод одной команды на стандартный ввод другой
- Использовать фильтры `tee`, `cut`, `tr`, `more` и `pr`

12–1. СЛАЙД: Конвейеры: введение

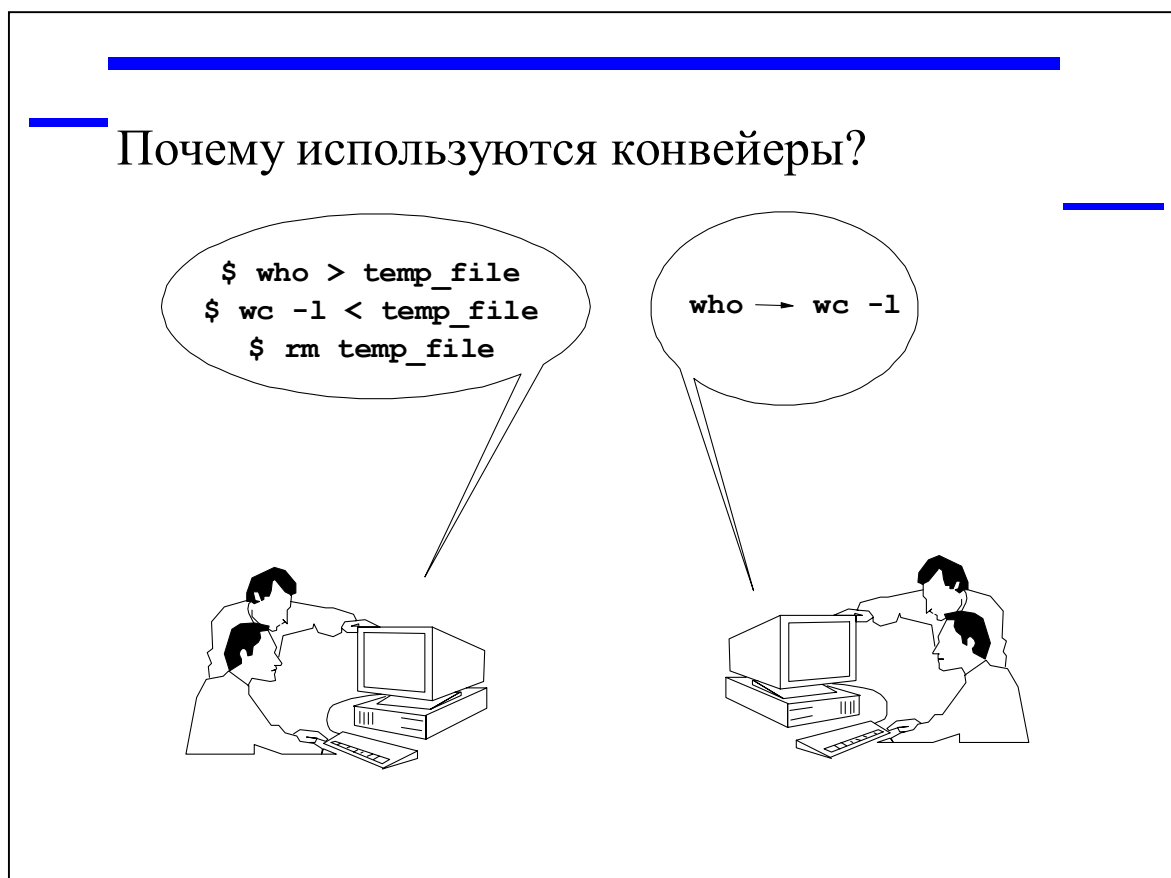


Пояснения

Одной из полезных возможностей shell'a является его способность соединять команды между собой, образуя конвейер. В состав операционной системы UNIX входят различные команды, позволяющие разными способами обрабатывать (фильтровать) содержимое файлов. Конвейеры же предоставляют возможность фильтровать вывод команды.

В данной главе представляются конвейеры и рассматриваются некоторые фильтры (**cut**, **tr**, **tee** и **pr**), позволяющие обрабатывать содержимое файлов или вывод команды.

12–2. СЛАЙД: Почему используются конвейеры?



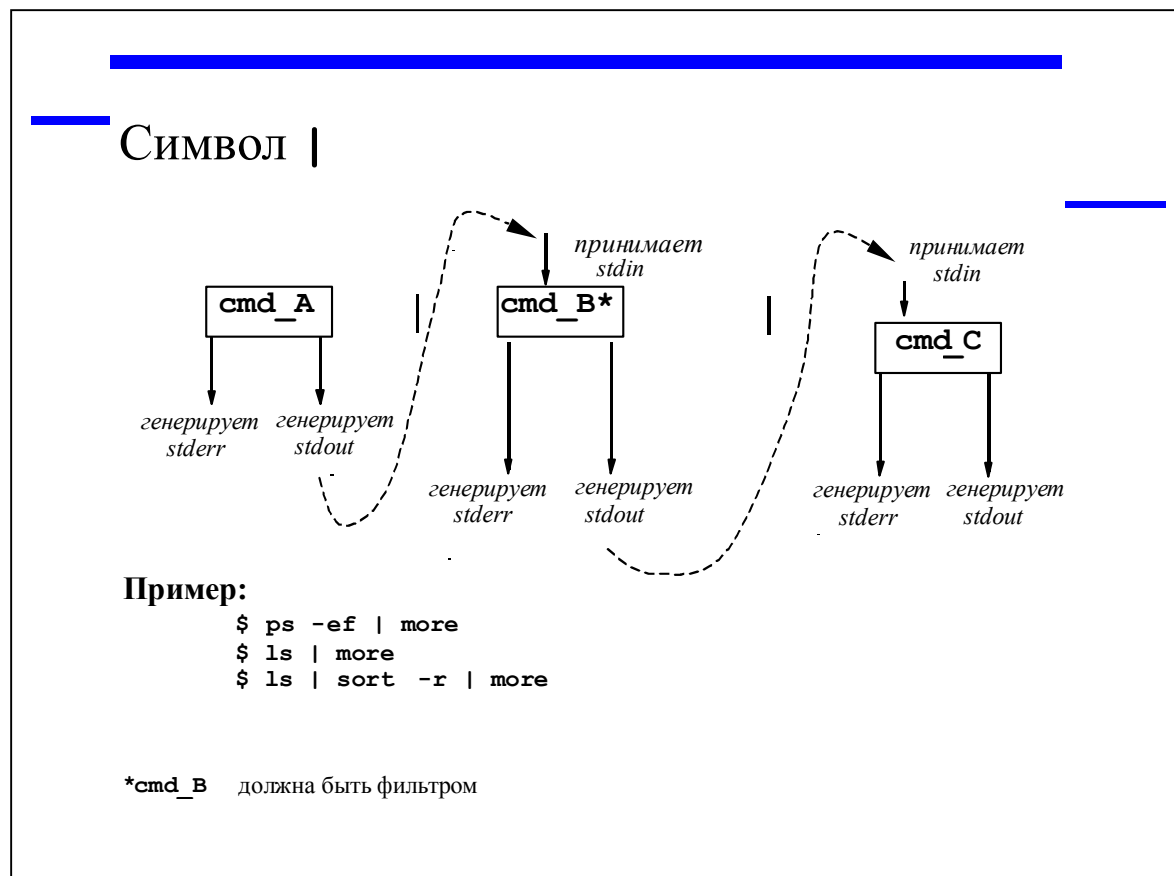
Пояснения

Если требуется обработать результаты работы команды, выводимые в стандартный вывод, можно использовать механизм перенаправления ввода/вывода: вывод команды перехватывается и направляется во временный файл, содержимое которого в дальнейшем и обрабатывается. По окончании обработки временный файл необходимо удалить, так как он не нужен для дальнейшей работы. И хотя перенаправление ввода/вывода предоставляет широкие возможности, использование такого механизма не очень удобно из-за необходимости удалять временные файлы.

Конвейеры позволяют напрямую использовать вывод одной команды в качестве ввода другой. При этом не нужно создавать промежуточный файл, а следовательно, не потребуется и удалять что-либо по окончании обработки.

В этом механизме проявляются гибкость и мощность системы UNIX. Можно соединять друг с другом команду за командой, реализуя тем самым в пределах одной командной строки самую разнообразную обработку.

12–3. СЛАЙД: СИМВОЛ |



Пояснения

Символ **|** (читается как **pipe** символ – символ конвейера) используется для связывания двух команд. Стандартный вывод (**stdout**) команды, расположенной слева от символа **|**, используется как стандартный ввод (**stdin**) команды, расположенной справа. Следовательно, команда, которая находится в середине конвейера, должна вводить данные из стандартного ввода и выводить результаты в стандартный вывод.

Такие фильтры, как **wc**, **sort** и **grep**, читают данные из стандартного ввода и направляют результаты в стандартный вывод, следовательно, они могут появиться в середине конвейера. Соединяя вместе (в одно целое) команды и фильтры, можно выполнять очень сложную обработку.

Ниже приводятся требования, предъявляемые к командам, включенным в конвейер, в зависимости от их позиции в конвейере:

- Любая команда, расположенная слева от символа **|**, должна направлять свои результаты (выводить) в стандартный вывод (**stdout**).
- Любая команда, расположенная справа от символа **|**, должна читать из стандартного ввода (**stdin**).
- Любая команда, расположенная между двумя символами **|**, должна читать из стандартного ввода и выводить в стандартный вывод (т.е. должна быть фильтром).

Команда **more**

Команда **more** используется для вывода на экран содержимого файлов поэкранно. Эта команда может также выводить на экран данные, прочитанные и из стандартного ввода. Поэтому она может появиться в конвейере справа от символа **|** и использоваться для управления выводом *любой* команды, которая направляет свои результаты в стандартный вывод. Эту команду полезно использовать, когда некоторая команда выводит на экран большой объем информации, которая не умещается на одном экране, и эти результаты желательно выводить и просматривать по одному экрану.

12–4. СЛАЙД: Конвейеры или перенаправление ввода/вывода

Конвейеры или перенаправление ввода/вывода

Перенаправление ввода/вывода

Синтаксис:

`cmd_out > файл`

и

`cmd_in < файл`

Пример:

`who > who.out`

`sort < who.out`

Конвейер

`cmd_out | cmd_in`

`who | sort`

Пояснения

Перенаправление ввода/вывода используется всегда между командой и файлом. Перенаправление вывода перехватывает стандартный вывод команды и направляет его в файл. Перенаправление вывода используется обычно в целях протоколирования или длительного хранения вывода команды. Перенаправление ввода заставляет команду вводить данные из файла, вместо того чтобы читать их с клавиатуры. Перенаправление ввода редко используется в явном виде, потому что большинство команд, которые могут читать из стандартного ввода, также разрешают задавать имена файлов в качестве аргументов командной строки (исключением являются команды **mail** и **write**). Однако способность читать из стандартного ввода является необходимой для команд, которые могут появиться справа от символа конвейера.

Конвейеры всегда используются для связывания двух команд. Если необходимо результаты одной команды обработать еще и с помощью другой команды, которая может читать из стандартного ввода, следует из этих двух команд построить конвейер. Перенаправление ввода/вывода используется для установления связи между командой и файлом. Конвейеры используются для установления связи между процессами.

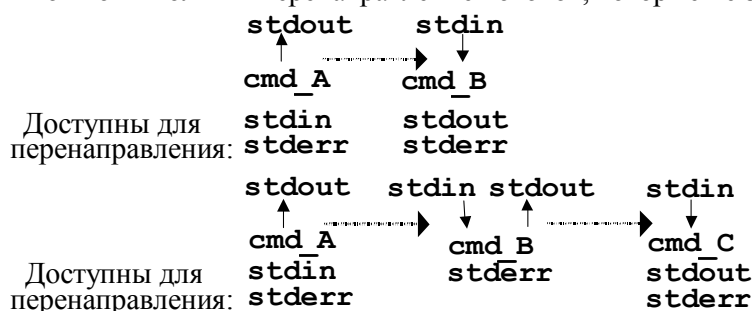
12–5. СЛАЙД: Перенаправление в конвейере

Перенаправление в конвейере

Три потока для каждой команды:

-stdin
-stdout
-stderr

Можно выполнять перенаправление потоков, которые не заняты конвейером:



Пример: `$ grep user /etc/passwd | sort > sort.out`

Пояснения

Каждая команда имеет три потока: стандартный ввод (**stdin**), стандартный вывод (**stdout**) и стандартный протокол (**stderr**). Каждая команда, используемая в конвейере, будет занимать для конвейера соответствующие потоки. Потоки, которые не используются в конвейере, могут быть перенаправлены.

Ниже приводятся правила перенаправления доступных потоков для разных компонентов конвейера:

- Первой команде, расположенной слева от символа конвейера, доступны для перенаправления только стандартный ввод и стандартный протокол, поскольку ее стандартный вывод связан со следующей командой в конвейере.
- Последней команде, расположенной справа от символа конвейера, доступны для перенаправления только стандартный вывод и стандартный протокол, поскольку ее стандартный ввод связан с предшествующей командой в конвейере.
- Любой команде, расположенной между двумя символами конвейера, доступен для перенаправления только стандартный протокол, поскольку ее стандартный ввод связан с предшествующей командой конвейера, а стандартный вывод – со следующей командой.

Примеры

Чаще всего используется перенаправление вывода последней команды в конвейере, чтобы сохранить окончательные результаты обработки данных. В этом случае появится ли что-либо на экране?

```
$ grep user /etc/passwd | sort > sorted.users  
$ grep user < /etc/passwd 2> grep.err | sort > sorted.users 2> sort.err  
$ grep user < /etc/passwd | sort 2> sort.err | wc -l > wc.out 2> wc.err
```

В приведенных выше примерах результаты обработки данных конвейером будут записаны в файл; стандартный вывод команд не будет отображаться на экране.

12–6. СЛАЙД: Фильтры

Фильтры

cut	Вырезает указанные позиции или поля и выводит в stdout
tr	Преобразует символы
tee	Передаёт вывод в файл <i>и</i> в stdout
pr	Форматирует вывод и направляет в stdout

Пояснения

Такие фильтры, как **sort** или **grep**, предоставляют удобный механизм для обработки вывода многих команд. В оставшейся части данного модуля будут рассмотрены три новых фильтра, часто используемые в конвейерах. Как и все другие фильтры, эти команды читают данные из стандартного ввода, следовательно, они могут появиться в правой части конвейера; кроме того, они выводят результаты в стандартный вывод, следовательно, могут появиться в левой части конвейера. И, конечно, эти команды могут появиться и в середине конвейера.

Команда **cut** позволяет вырезать отдельные позиции или поля из строк текста, читаемого из стандартного ввода или из файла, и вывести их в **stdout**.

Команда **tee** позволяет одновременно направить вывод команды в файл *и* в **stdout**.

Команда **pr** используется для форматирования вывода. Обычно она используется для подготовки файла к выводу на печать.

Как и все другие фильтры, эти команды не изменяют содержимое исходного файла. Полученные результаты работы этих команд направляются в стандартный вывод.

12–7. СЛАЙД: Команда **cut**

Команда **cut**

Синтаксис:

```
cut -с список [файл...]          Вырезает позиции или поля
cut -f список [-д символ] [-s] [файл...]  из строк файла или stdin
```

Примеры:

```
$ date | cut -c1-3
$ tail -1 /etc/passwd
user3:mdhbmkdj:303:30:student user3:/home/user3:/usr/bin/sh
  1      2      3  4      5      6      7
$ cut -f1,6 -d: /etc/passwd
$ cut -f1,6 -d: /etc/passwd | sort -r
$ ps -ef | cut -c49- | sort -d
```

Пояснения

Команда **cut** используется для выделения определенных позиций или полей из строк текста, прочитанных из стандартного ввода или из файла. Выделенные позиции или поля выводятся в стандартный вывод. Опция **-c** используется для указания требуемых позиций (колонок), а опция **-f** – для указания полей. Команда **cut** читает исходные данные из стандартного ввода или из файла. Так как команда читает из стандартного входа, она может быть указана справа от символа конвейера.

Список – это последовательность чисел, указывающая команде **cut**, какие поля или позиции нужно выделить. Поля в команде **cut** определяются так же, как и в команде **sort**. Для указания списка полей или позиций используются несколько способов:

- | | |
|------------|--|
| A–B | Все поля или позиции, заключенные между <i>A</i> и <i>B</i> (включая <i>A</i> и <i>B</i>) |
| A– | Все поля или позиции, начиная от <i>A</i> и до конца строки |
| –B | Все поля или позиции, начиная от начала строки и до позиции или поля <i>B</i> |
| A,B | Поля или позиции <i>A</i> и <i>B</i> |

Допускается использовать любые комбинации перечисленных выше способов задания полей или позиций. Например, команда:

```
cut -f1,3,5-7 /etc/passwd
```

из каждой строки файла `/etc/passwd` выделит поля с номерами 1, 3, 5, 6, 7.

По умолчанию, символом разделителем полей для команды `cut` является символ `Tab`. Если нужно использовать другой символ разделитель полей, нужно указать в команде опцию `-d СИМВОЛ`, где `СИМВОЛ` – это символ разделитель полей в строках исходного файла. (Эта опция аналогична опции `-t X` в команде `sort`.) Часто в качестве такого разделителя используется двоеточие, так как этот символ не имеет для shell'а специального смысла.

При выделении полей указание опции `-s` приводит к тому, что строки, в которых нет символа разделителя, будут пропускаться. Если эта опция не указана, тогда такие строки выводятся в стандартный вывод без каких-либо изменений.

Примеры

```
$ cut -c1-3 Return
12345
123
abcdefgh
abc
Ctrl + d

$ date | cut -c1-3
```

12–8. СЛАЙД: Команда **tr**

Команда **tr**

Синтаксис:

tr [-s] [*строка1*] [*строка2*] Преобразует символы

Примеры:

```
$ who | tr -s " "  
$  
$ date | cut -c1-3 | tr "[:lower:]" "[:upper:]"
```

Пояснения

Команда **tr** используется для преобразования символов. Она может читать данные и из файла, и из стандартного ввода; следовательно, ее можно использовать в конвейере.

Команда **tr** может быть использована для замены нескольких подряд расположенных символов пробела одним символом; эта возможность представлена в первом примере на слайде. Можно заметить, что многие команды системы UNIX вставляют между полями произвольное количество пробелов. Поэтому, когда при использовании команды **cut** желательно оперировать полями, разделенными *единственным символом пробела*, команде **cut** в конвейере может предшествовать команда **tr**.

Команду **tr** также можно использовать для подстановки текстовых строк или для преобразования строк из нижнего регистра к верхнему или наоборот, как показано во втором примере на слайде.

12–9. СЛАЙД: Команда **tee**

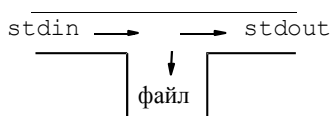
Команда **tee**

Синтаксис:

tee [-a] *файл* [*файл* . . .] Разветвляет конвейер

Примеры:

```
$ who | sort
$ who | tee unsorted | sort
$ who | tee unsorted | sort | tee sorted
$ who | wc -l
$ who | tee whoson | wc -l
```



Пояснения

Обычно, когда выполняется сложный конвейер, вывод промежуточных команд передается следующей команде конвейера, и нет возможности увидеть промежуточные результаты. Команда **tee** используется для разветвления конвейера. Эта команда читает данные из стандартного ввода и выводит их в стандартный вывод *и* в указанный файл. Если используется опция **-a**, тогда команда **tee** добавляет свой вывод в конец файла, а не перезаписывает его.

Команда **tee** используется преимущественно в двух ситуациях:

- Для сбора промежуточных результатов конвейера.
Когда команда **tee** размещается в середине конвейера, появляется возможность перехватить промежуточные результаты обработки (т.е. записать их в файл) и одновременно передать их следующей команде конвейера.
- Для вывода окончательных результатов обработки на экран и записи их в файл.
Это удобный механизм протоколирования. Может возникнуть необходимость при интерактивном выполнении команды видеть на экране ее результаты, но также и сохранять их в файле. Следует помнить, что если вывод команды перенаправляется в файл, он не отображается на экране. Следовательно, для решения такой задачи можно указать команду **tee** в конце конвейера или после любой команды, выводящей в стандартный вывод.

12–10. СЛАЙД: Команда **pr**

Команда **pr**

Синтаксис:

pr [-опции] [файл...] Форматирует **stdin** и результат
направляет в **stdout**

Примеры:

```
$ pr -n3 funfile
$ pr -n3 funfile | more
$ ls | pr -3
$ grep home /etc/passwd | pr -h "User Accounts"
```

Пояснения

Команда **pr** предназначена для *вывода («печати»)* в *стандартный вывод*. Она используется для форматирования стандартного ввода или содержимого указанных файлов. Результаты форматирования команда выводит в стандартный вывод, а не на принтер. Команда **pr** обычно используется для форматирования файлов перед отправкой их на печать.

Команда **pr** полезна для печати длинных файлов, так как она вставляет в верхнюю часть каждой страницы заголовок, содержащий имя файла (или заголовок, заданный опцией **-h**), и номер страницы.

Команда **pr** поддерживает много опций. Ниже приведены некоторые наиболее часто используемые опции.

- | | |
|------------|--|
| -k | Осуществляет вывод в k колонок; колонки выводятся сверху вниз |
| -a | Осуществляет вывод в несколько колонок; используется вместе с опцией -k ; печать осуществляется построчно |
| -t | Удаляет заголовки и завершающие записи («хвостовики») страниц |
| -d | Разделяет строки пустой строкой (удваивает пробелы) |
| -wN | Устанавливает длину строки равной N символам |

- lN** Устанавливает длину страницы равной **N** строкам
- nCK** Задаёт **K**-разрядную нумерацию строк, разделяя номер строки и строку символом **C**; если **C** не указан, используется Tab
- oN** Задаёт смещение вывода на **N** позиций от левого края страницы
- p** Устанавливает режим ожидания нажатия клавиши Return перед выводом каждой страницы
- h строка** Задаёт использование в качестве заголовка указанной **строки**

12–11. СЛАЙД: Печать из конвейера

Печать из конвейера

... | **lp** Ставится в конце конвейера; направляет вывод на принтер

Примеры:

```
$ pr -l58 funfile | lp
Request id is laser226 (standard input).
$
$ ls -F $HOME | pr -3 | tee homedir | lp
Request id is laser227 (standard input).
$
$ grep home /etc/passwd | pr -h "user accounts" | lp
Request id is laser228 (standard input).
```

Пояснения

Команда **lp** используется для постановки задания в очередь к принтеру. Задание определяется именем файла, указанным в качестве аргумента команды **lp**. Команда **lp** может также вводить из стандартного ввода, поэтому ее можно использовать в конвейере. Это позволяет вывести на печать результаты любой команды, выводимые в стандартный вывод.

Обычно, прежде чем передать команде **lp** для печати вывод команды, для его форматирования используется команда **pr**.

Так как большинство конвейеров выводят обработанные результаты в **stdout**, их вывод легко направить на принтер. Если нужно одновременно сохранить результаты обработки и направить их на принтер, перед командой **lp** в конвейере нужно поместить команду **tee**.

Система Linux



ПРИМЕЧАНИЕ: В системе Linux вместо команды **lp** нужно использовать команду **lpr**.

12–12. СЛАЙД: Конвейеры: заключение

Конвейеры: заключение

Конвейер	<code>cmd_out cmd_in</code> <code>cmd_out cmd_in_out cmd_in</code>
cut	Выделяет позиции или поля и направляет их в стандартный вывод
tee	Направляет свой ввод в стандартный вывод и в указанный файл
pr	Выводит на экран форматированные данные; обычно используется с командой lp
tr	Преобразует символы

Пояснения

12–13. УПРАЖНЕНИЯ: Конвейеры

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Создайте конвейер, который подсчитывает количество пользователей, в настоящее время работающих в системе.
2. Создайте конвейер, подсчитывающий количество строк в файле `/etc/passwd`, которые удовлетворяют модели поиска `home`. Затем подсчитайте количество строк, которые не удовлетворяют модели поиска.
3. Модифицируйте конвейер, созданный в предыдущем упражнении, так, чтобы все строки файла `/etc/passwd`, удовлетворяющие модели поиска `home`, были сохранены в файле `all.users`, прежде чем будет подсчитано их количество.
4. Создайте конвейер, который упорядочит содержимое файла `names` из вашего *HOME* каталога и выведет результаты в три колонки без заголовка и хвостовика.
5. Создайте псевдоним `whoson`, который выводит упорядоченный по алфавиту список всех пользователей, работающих в данный момент в системе.

6. Создайте конвейер, который позволит получить список, состоящий только из имен пользователей, в настоящее время работающих в системе.

7. Создайте конвейер, который позволит получить список имен файлов из вашего рабочего каталога и их прав доступа.

8. Создайте конвейер, который для каждого файла из вашего *HOME* каталога запишет в файл **listing.out** имя владельца файла, размер файла и имя файла и одновременно выведет на экран общее количество файлов.

9. Создайте конвейер, который для каждого пользователя системы выберет имя пользователя, его числовой идентификатор и имя *HOME* каталога. Выведите полученный список, упорядочив его по имени пользователя. Измените конвейер так, чтобы список был упорядочен по числовым идентификаторам пользователей. ПОДСКАЗКА: необходимую информацию можно получить из файла **/etc/passwd**.

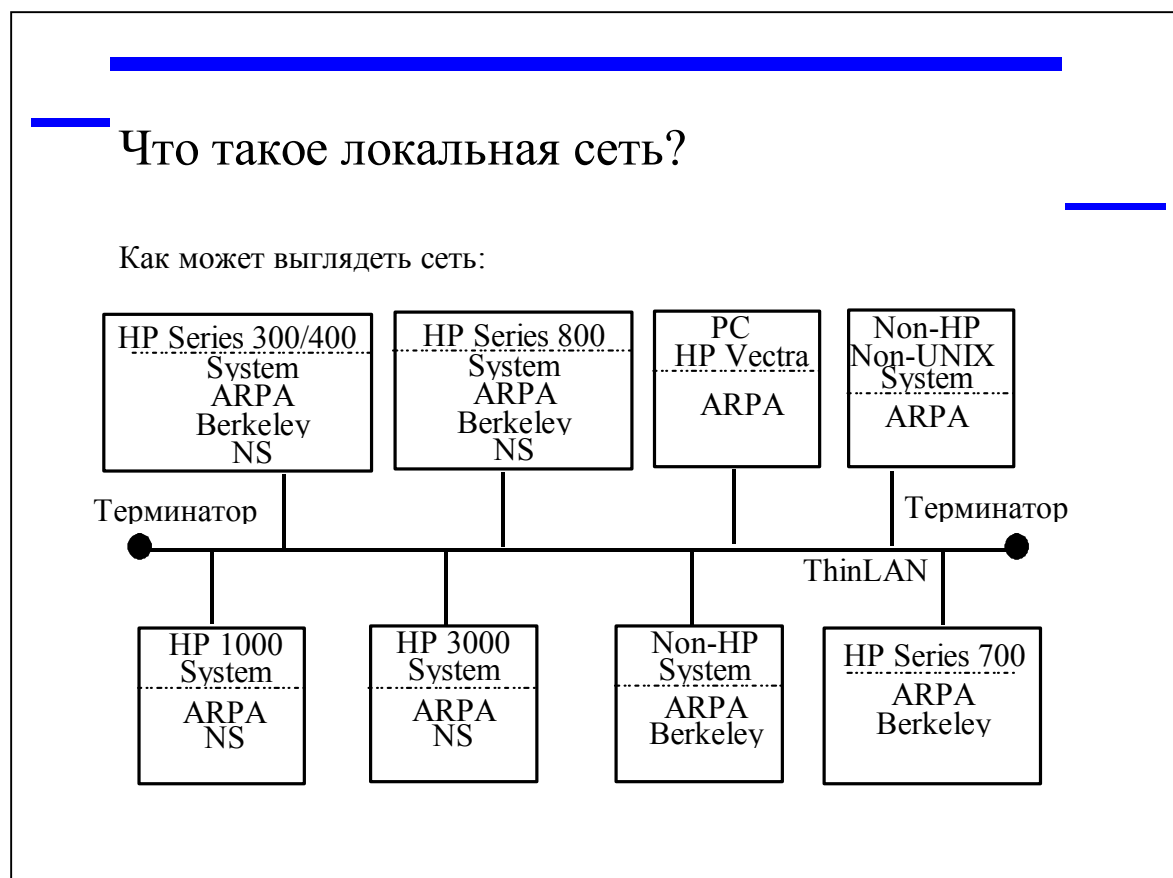
Модуль 13 — Использование сетевых возможностей

Цели:

Изучив этот модуль, вы сможете:

- Описывать различные сетевые сервисы, используемые в HP-UX
- Объяснить назначение и функции локальной сети (LAN)
- Определить имя локального компьютера и других компьютеров, включенных в локальную сеть
- Использовать ARPA/Berkeley сервисы для входа на удаленную машину, передачи данных между удаленными машинами и выполнения команд на удаленной машине

13–1. СЛАЙД: Что такое локальная сеть?



Пояснения

Локальная сеть (Local Area Network – LAN) представляет собой способ соединения друг с другом двух или более компьютерных систем, расположенных на небольшом удалении друг от друга. На любом предприятии, имеющем более одного компьютера, компьютеры объединяются в локальную сеть, что позволяет пользователям работать на различных компьютерах без необходимости физического присутствия у того или иного компьютера.

Сервисы локальной сети (LAN), обсуждаемые в данном модуле, представляют собой программы, которые позволяют использовать сеть для решения многих задач, требующих использования разных компьютеров. К таким задачам относятся:

- Копирование файлов с одного компьютера на другой. При отсутствии LAN для решения этой задачи нужно было бы сделать копию файлов на магнитной ленте, перенести ленту на другой компьютер и на этом компьютере загрузить с нее файлы.
- Вход на удаленный компьютер с терминала, подсоединенного к локальному компьютеру. В обычной ситуации (без сети), для того чтобы начать сеанс работы на каком-либо компьютере, нужно, прежде всего, подойти к этому компьютеру.
- Выполнение команд на удаленном компьютере и просмотр результатов их выполнения на дисплее локального компьютера. Опять же, при отсутствии LAN нужно было бы физически идти к удаленному компьютеру и на нем выполнять команду и смотреть ее результаты.

- Получение доступа к файлам, размещенным на удаленном компьютере. Это означает, что можно использовать файлы, размещенные на диске удаленного компьютера, не копируя их на локальный диск.

13–2. СЛАЙД: Сервисы локальной сети (LAN)

Сервисы локальной сети (LAN)

- Две группы сервисов локальной сети:
 - ARPA сервисы
 - Berkeley сервисы
- Сервисы позволяют:
 - входить в систему на удаленной машине
 - копировать файлы с удаленной машины или на нее
 - осуществлять доступ к удаленным файлам

Пояснения

В данном модуле рассматриваются две различные *группы* сервисов, обеспечивающих выполнение основных функций локальной сети. К ним относятся:

- ARPA сервисы
- Berkeley сервисы

ARPA сервисы были впервые определены Управлением перспективных исследований и разработок Министерства обороны США (Defense Advanced Research Projects Agency – DARPA) в конце 60-х годов. Эти сервисы стали стандартом для организации взаимодействия многих компьютеров различных моделей, объединенных в одну сеть. В данном модуле рассматриваются ARPA сервисы **telnet** и **ftp**.

Для разработки этих сервисов DARPA привлек Калифорнийский университет в Беркли (University of California, Berkeley) и сотрудников BBN, штат Massachusetts, Болта (Bolt), Баранека (Baranek) и Ньюмана (Newman). В середине 70-х годов в Беркли (Berkeley) начались работы по созданию новой операционной системы UNIX. В ходе этих работ были созданы более устойчивые сервисы, обеспечивающие взаимодействие компьютеров, работающих под управлением операционной системы UNIX. Они получили название Berkeley сервисов. В данном модуле рассматриваются следующие Berkeley сервисы: **rcp**, **rlogin** и **remsh**.

13–3. СЛАЙД: Команда `hostname`

Команда `hostname`

Синтаксис:

hostname Сообщает сетевое имя компьютера

Пример:

```
$ hostname
fred
$
$ more /etc/hosts
192.1.2.1    fred
192.1.2.2    barney
192.1.2.3    wilma
192.1.2.4    betty
```

Пояснения

Каждый компьютер в сети имеет сетевое имя, которое идентифицирует в LAN систему. Чтобы узнать это имя, используется команда **hostname**.

```
$ hostname
fred
```

Если нужно организовать взаимодействие с другим компьютером, включенным в сеть, нужно знать его имя. Это имя можно узнать у системного администратора соответствующего компьютера. Кроме того, на удаленной машине должна быть учетная запись пользователя, который хочет организовать взаимодействие с этой машиной.

ПРИМЕЧАНИЕ: Для того чтобы использовать какие-либо сервисы локальной сети, пользователь должен быть надлежащим образом зарегистрирован на удаленном компьютере.

Имена компьютеров, объединенных в локальную сеть, можно найти в файле **/etc/hosts**. Однако, если локальная сеть достаточно велика, этот файл может содержать несколько сотен имен, что затрудняет поиск нужного имени.

13–4. СЛАЙД: Команда telnet

Команда telnet

Синтаксис:

telnet *имя_компьютера* ARPA сервис для удаленного входа
на другой компьютер

Пример:

```
$ telnet fred
Trying ...
Connected to fred.
Escape character is '^]'.

HP-UX fred 10.0  9000/715
login:
```

Пояснения

telnet – это механизм удаленного входа, предоставляемый ARPA сервисами.

Если выполнить команду:

```
$ telnet имя_компьютера
```

тогда на экране можно будет увидеть приглашение для входа в систему (**login:**) того компьютера, имя которого было указано в командной строке. Если в ответ на это приглашение ввести имя пользователя и пароль, под которыми пользователь зарегистрирован на удаленной машине, можно будет начать сеанс работы на удаленной машине.

После этого локальный терминал будет представлять собой терминал удаленного компьютера. Пользователь получает возможность выполнять любые команды и программы shell'a и даже использовать принтер удаленного компьютера. *Все действия, выполняемые пользователем, будут выполняться на удаленном компьютере.* Локальный же компьютер только передает по сети информацию, вводимую с терминала и выводимую на него.

Чтобы закрыть соединение **telnet**, нужно просто завершить сеанс работы на удаленном компьютере, нажав **Ctrl**+**d** или выполнив команду **exit**.

13–5. СЛАЙД: Команда **ftp**

Команда **ftp**

Синтаксис:

ftp *имя_компьютера* ARPA сервис для копирования файлов с удаленной машины и на нее

Команды **ftp** :

get	Получить файл с удаленной машины
put	Передать локальный файл на удаленную машину
ls	Получить список файлов, находящихся на удаленной машине
?	Получить список всех команд ftp
quit	Выйти из ftp

Пояснения

Для копирования файлов с удаленной машины и на нее с помощью ARPA сервисов используется команда **ftp**. Имя **ftp** расшифровывается как *протокол передачи файлов – file transfer protocol*. Как и в команде **telnet**, в команде **ftp** необходимо указать имя удаленного компьютера:

```
$ ftp имя_компьютера
```

При выполнении команды **ftp** на экран будет выведено приглашение ввести имя пользователя и пароль, под которыми пользователь зарегистрирован на удаленном компьютере. Для использования команды **ftp** пользователь *должен* иметь на удаленном компьютере пароль. Если входные данные введены правильно, будет установлено соединение с компьютером, имя которого указано в команде.

Начиная с этого момента, на экран будет выводиться приглашение в виде **ftp>**. В ответ на это приглашение можно вводить разнообразные команды **ftp**, необходимые для выполнения нужных действий. Ниже приводится описание некоторых команд **ftp**, используемых для передачи файлов между локальным и удаленным компьютерами:

get r_файл l_файл Копировать файл с именем **r_файл**, размещенный на удаленной машине, в файл с именем **l_файл** на локальной машине. При задании имен файлов можно использовать и абсолютные, и относительные маршрутные имена.

put <i>l_file</i> <i>r_file</i>	Копировать файл с именем <i>l_файл</i> с локальной машины в файл с именем <i>r_файл</i> на удаленной машине.
ls	Вывести список файлов, размещенных на удаленном компьютере. Команда выполняется аналогично команде ls , которую мы рассматривали ранее.
?	Вывести список всех команд, входящих в состав ftp .
help <i>команда</i>	Вывести краткое (очень краткое) описание <i>команды</i> .
quit	Отсоединиться от удаленного компьютера и выйти из ftp .

Если, например, нужно скопировать файл с именем **funfile**, находящийся на локальном компьютере, в каталог **/tmp** другого компьютера, имя которого **fred**, сеанс работы в **ftp** может выглядеть следующим образом (подчеркиванием выделен текст, который вводится пользователем):

```
$ ftp fred
Connected to fred.
220 fred FTP server (Version 1.7.109.2 Tue Jul 28 23:46:52 GMT 1992)
ready.

Name (fred:gerry): Return
Password (fred:gerry): вводится пароль пользователя и нажимается Return
331 Password required for gerry.
230 User gerry logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put funfile /tmp/funfile
200 PORT command successful.
150 Opening BINARY mode data connection for /tmp/funfile.
226 Transfer complete.
3967 bytes sent in 0.19 seconds (20.57 Kbytes/sec)

ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls /tmp.
-rw-rw-rw-  1 root sys  347 Jun 14 1993 exercises
-rw-rw-rw-  1 root sys   35 Oct 23 1993 cronfile
-rw-r-----  1 root sys  41 Jul  6 17:19 fio
-rwxrw-rw-  1 root sys 153 Oct 23 1993 initlaserjet
-rw-rw-rw-  1 root sys  37 Nov 21 1994 funfile
226 Transfer complete.
ftp> quit
221 Goodbye.
```

Первое, на что обращаешь внимание при использовании **ftp** – это то, что команда весьма многословна. Она выводит ответ на каждую введенную пользователем команду. (Можно заметить, что эта команда не входила первоначально в состав средств системы UNIX!)

Система Linux



Благодаря политике безопасности, применяемой по умолчанию в системе Linux, пользователям более сложно с помощью команды **ftp** выполнять копирование файлов с машины, на которой установлена система Linux, или на нее.

13–6. СЛАЙД: Команда **rlogin**

Команда **rlogin**

Синтаксис:

rlogin *имя_компьютера* Berkeley сервис для удаленного входа на другой компьютер; **rlogin** пытается начать сеанс работы на удаленной машине, используя локальное входное имя пользователя

Пример:

```
$ hostname  
barney  
$ rlogin fred  
Password:  
$ hostname  
fred  
$ exit  
$ hostname  
barney
```

Пояснения

Команда **rlogin** выполняет функции, аналогичные функциям команды **telnet**. Если пользователь выполнит команду:

```
$ rlogin имя_компьютера
```

он автоматически начинает сеанс работы на компьютере, имя которого было указано в команде. Команда **rlogin** подразумевает, что пользователь входит на удаленный компьютер под тем же именем, под которым он начал сеанс работы на локальном компьютере. Поэтому эта команда не запрашивает имя пользователя.

Если администратор системы сконфигурировал файл с именем **/etc/hosts.equiv**, команда **rlogin** не будет даже запрашивать ввод пароля. Поэтому с ее помощью можно очень быстро и легко начать сеанс работы на удаленном компьютере. Кроме того, в *HOME* каталоге пользователя может быть создан файл с именем **.rhosts**, который также позволяет входить на удаленный компьютер без ввода пароля. Подробное описание формата файла **.rhosts** можно найти в статье **hosts.equiv (4)** Справочника.

Как и при использовании команды **telnet**, для отсоединения от удаленного компьютера достаточно просто завершить на нем сеанс работы.

13–7. СЛАЙД: Команда **rcp**

Команда **rcp**

Синтаксис:

rcp *файл_источник* *файл_приемник* Berkeley сервис для копирования файлов на удаленную машину или с нее; работает аналогично команде **cp**. Имена файлов на удаленной машине задаются в виде:
имя_компьютера:маршрутное_имя

Пример:

```
$ rcp funfile fred:/tmp/funfile
$
```

Пояснения

Имя команды **rcp** означает удаленное копирование – **remote cp**. Такое название объясняется тем, что эта команда выполняет те же функции, что и команда **cp**. Она пересылает файлы между двумя компьютерами, на которых работают Berkeley сервисы. Общий формат команды:

```
$ rcp компьютер1:источник компьютер2:приемник
```

где аргументы означают копирование файла **источник** с **компьютера1** в файл **приемник** на **компьютер2**. **Источник** и **приемник** должны быть заданы с помощью абсолютного маршрутного имени.

Если при копировании один из файлов размещается на локальном компьютере, при указании его имени можно не указывать имя локального компьютера (вместе с символом **:**) и использовать относительное маршрутное имя. Приведенные ниже примеры помогут более четко понять команду **rcp**:

- Скопировать файл **funfile**, расположенный на локальной машине (имя локальной машины **bambam**), в файл **/tmp/funfile** на удаленную машину с именем **fred**:

```
$ rcp funfile fred:/tmp/funfile
```
- Скопировать файл **/tmp/funfile** с машины **fred** в каталог **/tmp** на машину **barney**:

```
$ rcp fred:/tmp/funfile barney:/tmp
```

Все правила, относящиеся к команде **cp**, применимы также и к команде **rcp**.

ПРИМЕЧАНИЕ: Для того чтобы команда **rcp** работала, должен быть правильно сконфигурирован файл **/etc/hosts.equiv** или **.rhosts**.

Система Linux



Благодаря политике безопасности, применяемой по умолчанию в системе Linux, конечным пользователям более сложно с помощью команды **rcp** выполнять копирование файлов с машины, на которой установлена система Linux, или на нее.

13–8. СЛАЙД: Команда **remsh**

Команда **remsh**

Синтаксис:

remsh *имя_компьютера команда* Berkeley сервис для запуска команды на удаленной машине

Пример:

```
$ hostname
barney
$ remsh fred ls /tmp
backuplist
croutOqD00076
fred.log
Update.log
$ ls /tmp
EX000662    tmpfile    Update.log
$
```

Пояснения

Команда **remsh** позволяет выполнять команды на удаленном компьютере и видеть результаты их исполнения на локальном терминале. Команда имеет следующий формат:

```
$ remsh имя_компьютера команда
```

Например, если нужно увидеть, какие процессы исполняются на компьютере **fred**, можно выполнить команду:

```
$ remsh fred ps -ef
```

Вывести список файлов, содержащихся в каталоге **/tmp** компьютера **fred**:

```
$ remsh fred ls /tmp
fredfile
funfile
reconfig.log
update.log
```

Или нужно увидеть содержимое файла **/etc/hosts** на компьютере **fred**:

```
$ remsh fred cat /etc/hosts | more
```

Следует заметить, что только команда **cat /etc/hosts** выполняется на удаленном компьютере **fred**. Результаты исполнения команды передаются на локальный терминал и здесь по конвейеру передаются на вход команды **more**.

С помощью команды **remsh** можно также напечатать файлы на принтере, подсоединенном к другому компьютеру:

```
$ cat myfile | remsh fred lp
```

ПРИМЕЧАНИЕ: Для того чтобы команда **remsh** работала, должен быть правильно сконфигурирован файл **/etc/hosts.equiv** или **.rhosts**.

Система Linux



В системе Linux для запуска удаленного shell'a используется команда **rsh**. Как упоминалось ранее, в системе Linux существует применяемая по умолчанию политика безопасности, которая может препятствовать запуску удаленного shell'a.

13–9. СЛАЙД: Команда **rwho**

Команда **rwho**

rwho Генерирует такой же результат, как и команда **who**

rwho Выводит имена пользователей, работающих на тех машинах в сети LAN, на которых работает демон **rwhod**

Пример:

```
$ rwho
user1    barney:tty0p1 Jul 18    8:23    :10
user2    wilma:tty0p1  Jul 18   10:13    :03
user3    fred:tty0p1   Jul 18   11:32    :06
```

Пояснения

Команда **rwho** работает аналогично команде **who**, но выдает информацию обо всех пользователях, работающих на всех тех компьютерах, включенных в локальную сеть, на которых исполняется демон **rwhod**.

Система Linux



Кроме команды **rwho**, в системе Linux есть еще команда **rusers**. Предполагается, что обе команды выводят имена пользователей, работающих на компьютерах, включенных в локальную сеть. Однако на практике обе команды имеют тенденцию работать не так, как ожидается, и постоянно ничего не выводят.

13–10. СЛАЙД: Команда `ruptime`

Команда `ruptime`

ruptime Выводит на экран состояние каждой машины, включенной в локальную сеть.

На каждой машине, включенной в сеть, должен работать демон **rwhod**.

Пример:

```
$ ruptime
barney  up          3:10   1 users load 1.32, 0.80, 0.30
fred    up          1+5:15  4 users load 1.47, 1.16, 0.80
wilma   down        0:00
```

Пояснения

Команда **ruptime** выводит на экран информацию о состоянии всех компьютеров, включенных в локальную сеть: работают они или выключены, сколько пользователей работают в настоящее время, а также насколько интенсивно работает система.

Для машины **fred**, приведенной на слайде, эта информация имеет следующий вид:

- в настоящий момент машина **fred** работает
- машина **fred** работает в течение 1 дня, 5 часов и 15 минут
- на машине **fred** работают 4 пользователя
- в течение последнего 1-минутного интервала времени в очереди на обработку находилось в среднем 1.47 заданий
- в течение последнего 5-минутного интервала времени в очереди на обработку находилось в среднем 1.16 заданий
- в течение последнего 15-минутного интервала в очереди на обработку находилось в среднем 0.80 заданий

Система Linux



Команда **ruptime** в системе Linux также зависит от демона **rwhod**.

13–11. УПРАЖНЕНИЯ

Указания

Узнайте у вашего преподавателя, какие упражнения вы сможете выполнить на занятии. Также узнайте имена компьютеров, с которыми вы сможете установить связь.

Система Linux



ПРИМЕЧАНИЕ: Пользователи системы Linux не смогут выполнить упражнения 3 и 4 из-за применяемой в системе по умолчанию политики безопасности. Поэтому пользователям системы Linux не следует пытаться выполнять эти упражнения.

1. Используя команду **hostname**, определите сетевое имя локального компьютера, на котором вы работаете. С какими компьютерами в сети вы можете установить связь?
2. Используя команду **telnet**, установите соединение с другим компьютером. Используя команду **hostname**, проверьте, что вы установили связь именно с тем компьютером, с каким хотели. После этого завершите сеанс работы с удаленным компьютером.
3. С помощью команды **ftp** перешлите один из своих файлов в свой *HOME* каталог на удаленном компьютере, затем с помощью команды **rcp** скопируйте другой файл на удаленный компьютер. Обратите внимание на различия.

4. Используя команду **remsh**, выведите содержимое своего *HOME* каталога на удаленной машине, чтобы проверить, что копирование файлов в предыдущем упражнении было выполнено.

Модуль 14 — Введение в редактор vi

Цели:

Изучив этот модуль, вы сможете:

- Эффективно использовать vi для редактирования текстовых файлов

14–1. СЛАЙД: Что такое vi?

Что такое vi?

- Экранный текстовый редактор
- Включен в большинство версий системы UNIX
- Управляется командами
- Включает следующие категории команд:
 - общее управление
 - перемещение курсора
 - ввод текста
 - удаление текста
 - вставка текста
 - изменение текста

Пояснения

vi (сокращение от слова *visual* – *визуальный*) – это стандартный текстовый редактор, входящий в состав практически всех версий системы UNIX. Текстовый редактор – это интерактивная программа, позволяющая вводить и редактировать текстовые файлы. Редактор **vi** можно использовать для создания новых или изменения существующих файлов.

Редактор **vi** разработан Вильямом Джоем (William Joy) из Калифорнийского университета в Беркли (University of California, Berkeley). Он представляет собой интерактивный экранный редактор. Содержимое файла выводится на экран, и все изменения, вносимые в файл, сразу же отображаются на экране. (Система UNIX также поддерживает пакетные редакторы, такие как **ed**, **sed** и **awk**, которым для выполнения необходимых изменений файла передается пакет, содержащий команды редактирования.)

Редактор **vi** разрабатывался как редактор, независимый от типа терминала; его команды отображаются почти на каждую клавишу стандартной клавиатуры. Первоначально он использовался на телетайпах, не имевших специальных функциональных клавиш и клавиш со стрелками (курсоров). Поэтому он может и не использовать преимущества специальных клавиш, имеющихся на конкретном терминале.

Преимущества такой философии проектирования заключаются в том, что **vi** может использоваться почти на всех типах терминалов. Так как он доступен почти во всех версиях

системы UNIX, вам не придется каждый раз, когда вы переходите к другой версии системы UNIX, учить новый редактор или новый набор клавиш редактирования.

Интерфейс редактора **vi** легко настраивается. Карты отображения (mappings) позволяют настроить любую клавишу на клавиатуре, включая и специальные клавиши.

В этом модуле рассматриваются основные возможности редактора **vi**. Профессиональное использование **vi** требует определенной практики работы с ним. Чем больше практика работы с редактором, тем более профессиональным вы становитесь. Этот модуль даст вам хорошую основу для основных операций редактирования файлов и позволит вам повышать свое мастерство в соответствии с вашим темпом работы.

Система Linux



В системе Linux вместо **vi** используется текстовый редактор **vim**. Редактор **vim** (**vi Improved** – улучшенный **vi**) совместим (снизу вверх – от **vi** к **vim**) с редактором **vi**. Для выяснения всех подробностей, отличающихся от **vi** и совпадающих с ним, обратитесь к соответствующим статьям Справочника. Редактор **vim** можно вызвать, также используя в качестве имени команды **vi**.

14–2. СЛАЙД: Почему vi?

Почему vi?

- Существует на любой платформе UNIX
- Работает на терминалах любого типа
- Очень мощный редактор
- Используется командным стеком shell'a
- Используется другими средствами системы UNIX

Пояснения

Вне зависимости от того, на какой UNIX машине или в какой UNIX операционной системе вы работаете, **vi** всегда в ней есть. Редактор **vi** представляет собой экранный редактор и предназначен для работы с любым типом ASCII терминала вне зависимости от того, кто является его производителем. Экранные редакторы обычно требуют терминалы определенного типа, но редактор **vi** работает по-другому. Действительно, редактор **vi** не является очень дружелюбным, но он всегда есть под рукой, когда вам нужно редактировать файл.

Многие более новые редакторы в действительности представляют собой текстовые процессоры (системы подготовки текстов – word processors), которые могут также выполнять определенные функции редактирования. Текстовые процессоры обычно имеют очень дружелюбный интерфейс и интуитивно понятны. Для небольших файлов вполне подходят текстовые процессоры или редакторы оконного типа.

При редактировании большой shell-процедуры опытные пользователи **vi** обнаружат, что с помощью редактора **vi** они работают более продуктивно, нежели с помощью редактора Windows. Несмотря на свою загадочность, **vi** является очень эффективным и мощным редактором, может редактировать одновременно несколько файлов и быстро вырезает тексты из одного файла и вставляет их в другой.

Если вы используете Korn shell или POSIX shell, вы также могли заметить, что команды, используемые для работы со стеком команд, представляют собой команды редактора **vi**. И Korn shell, и POSIX shell используют **vi** как более привилегированный редактор.

Многие другие инструментальные средства системы UNIX при необходимости изменить конфигурацию поместят вас в сеанс работы **vi**. Для того чтобы вас считали опытным пользователем системы UNIX, вы должны профессионально работать в этом редакторе.

14-3. СЛАЙД: Начало сеанса работы в vi

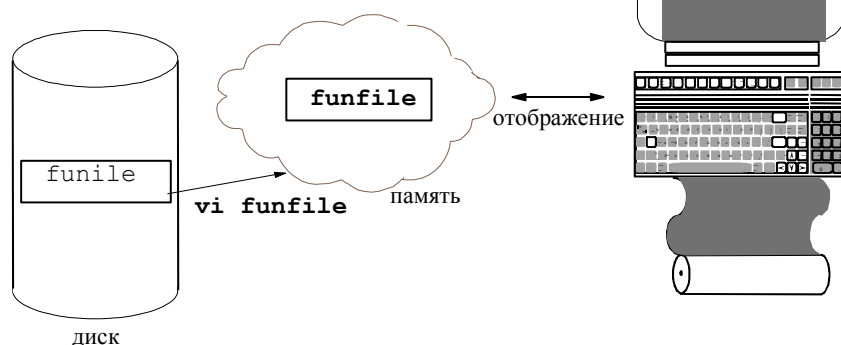
Начало сеанса работы в vi

Синтаксис:

vi [*имя_файла*] Начинает сеанс vi редактирования файла

Пример:

\$ vi funfile



Все изменения выполняются над копией файла, помещенной в оперативную память

Пояснения

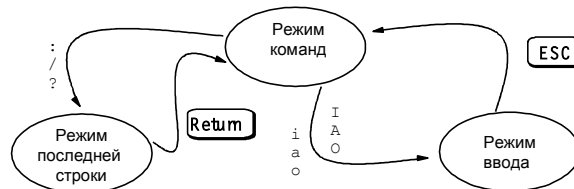
Сеанс редактирования начинается вызовом команды **vi**. Если указанный в командной строке файл уже существует, на экран выводится первый фрагмент текста. В противном случае, т.е. если в командной строке указан новый файл, будет выведен пустой экран, на котором в первой позиции каждой строки помещен символ «тильда» (~) – признак того, что указанные строки не принадлежат редактируемому файлу. Редактор **vi** помещает копию файла во временный **буфер**, размещенный в оперативной памяти, и все изменения будут выполняться над этой копией. Копия файла на диске будет обновлена только тогда, когда выполняется команда сохранения буфера на диске. Поэтому если вдруг окажется, что файл был изменен некорректно, от временного буфера можно просто отказаться, и тогда файл на диске не будет изменен.

При редактировании файла экран становится окном, через которое можно видеть редактируемый файл. Обычно изменения в файле затрагивают символ, слово или строку, отмеченные курсором. Поэтому *все время надо следить за тем, где находится курсор*. Как только в текст файла вносятся какие-либо изменения, они сразу же отображаются на экране.

14-4. СЛАЙД: Режимы vi

Режимы vi

- **Режим команд** – нажатие клавиш интерпретируется как ввод команды
 - исходный режим; устанавливается, когда начинается сеанс работы vi
 - нажатие клавиши **ESC** переводит в режим команд
 - команды **vi** (**i**, **a**, **o**, и т.д.) не отображаются на экране
- **Режим последней строки** – **ex** и поиск текста
 - команды **ex**, которым предшествует символ двоеточия (:), отображаются на экране
 - команды поиска (**/** и **?**) отображаются на экране
- **Режим ввода** – **vi** записывает в файл коды, соответствующие нажатым клавишам



Пояснения

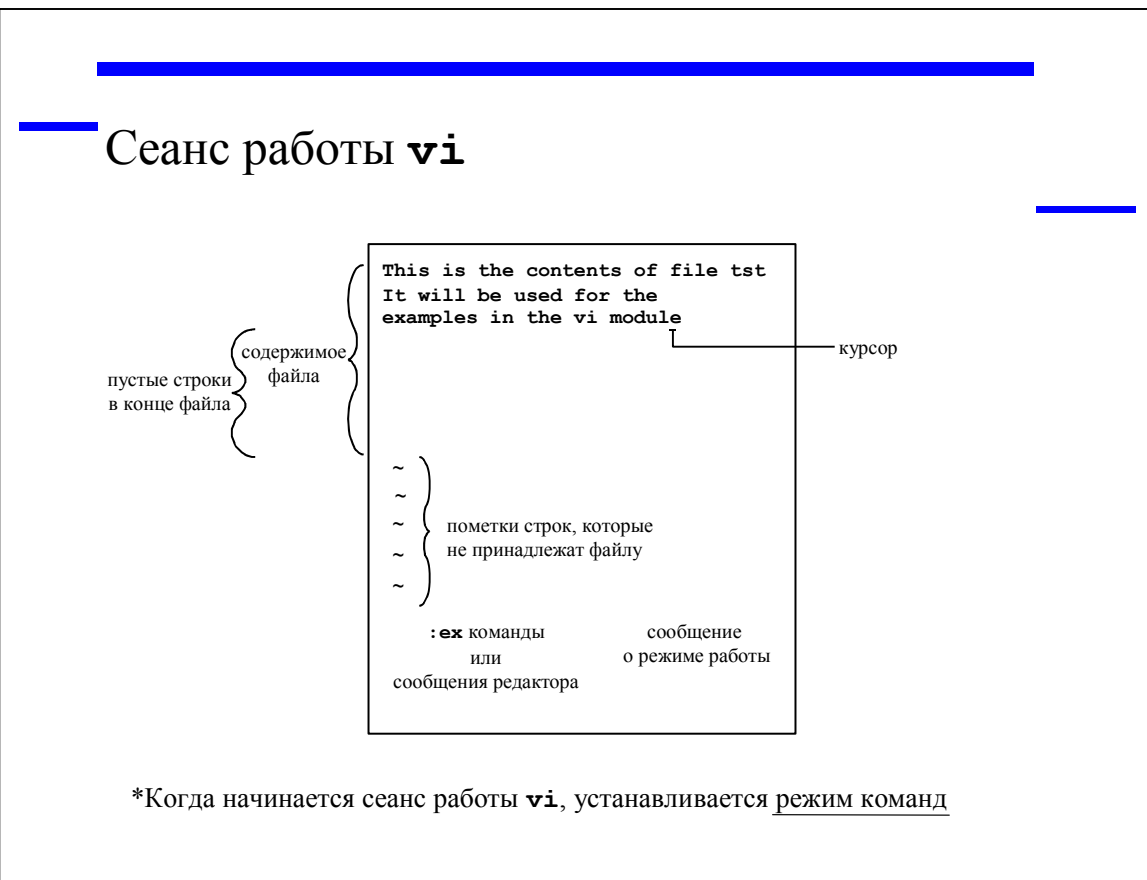
vi представляет собой редактор, управляемый командами. Когда пользователь начинает сеанс редактирования в **vi**, он попадает в режим команд. Поэтому если нажимается какая-либо клавиша на клавиатуре, **vi** пытается выполнить команду, ассоциированную с этой клавишей. Почти с каждой клавишей клавиатуры **vi** связывает какую-либо функцию. Существуют команды, позволяющие вводить текст, перемещать курсор, модифицировать, удалять и вставлять текст. Обычно команды **vi** работают молча, т.е. когда вводится какая-либо команда **vi**, она не отображается на экране, и пользователь может видеть на экране только результаты ее исполнения.

В системе UNIX существует (и достаточно широко используется) редактор **ex** – строковый редактор с расширенными возможностями; его команды доступны из сеанса редактирования **vi**. Команды редактора **ex** вводятся в ответ на приглашение, представленное символом «двоеточие», и в отличие от команд **vi**, они отображаются на экране и выполняются только после нажатия клавиши **Return**. Эти команды обычно используются для модификации сразу нескольких строк и для настройки сеанса редактирования.

Определенные команды **vi** переводят редактор в режим ввода, в котором нажатие любой клавиши приводит к вводу в файл соответствующего текста. Чтобы вернуться в режим команд, нужно просто нажать клавишу **Esc**.

ПРИМЕЧАНИЕ: Некоторые команды **vi** требуют нажатия нескольких клавиш. Если во время ввода команды **vi** вы ошиблись, для того чтобы прервать ввод текущей команды, просто нажмите клавишу **Esc**.

14-5. СЛАЙД: Сеанс работы vi



Пояснения

Когда начинается сеанс работы vi, на экране появляется что-то аналогичное тому, что показано на слайде. На экране можно увидеть текст файла как бы через *окно*. На экране можно увидеть следующие компоненты отображения редактора vi:

текстовый курсор	указывает на некоторый символ файла
область текста	отображает содержимое файла
область команд ex	отображает команды ex и сообщения редактора vi
область сообщения о режиме	отображает режим работы

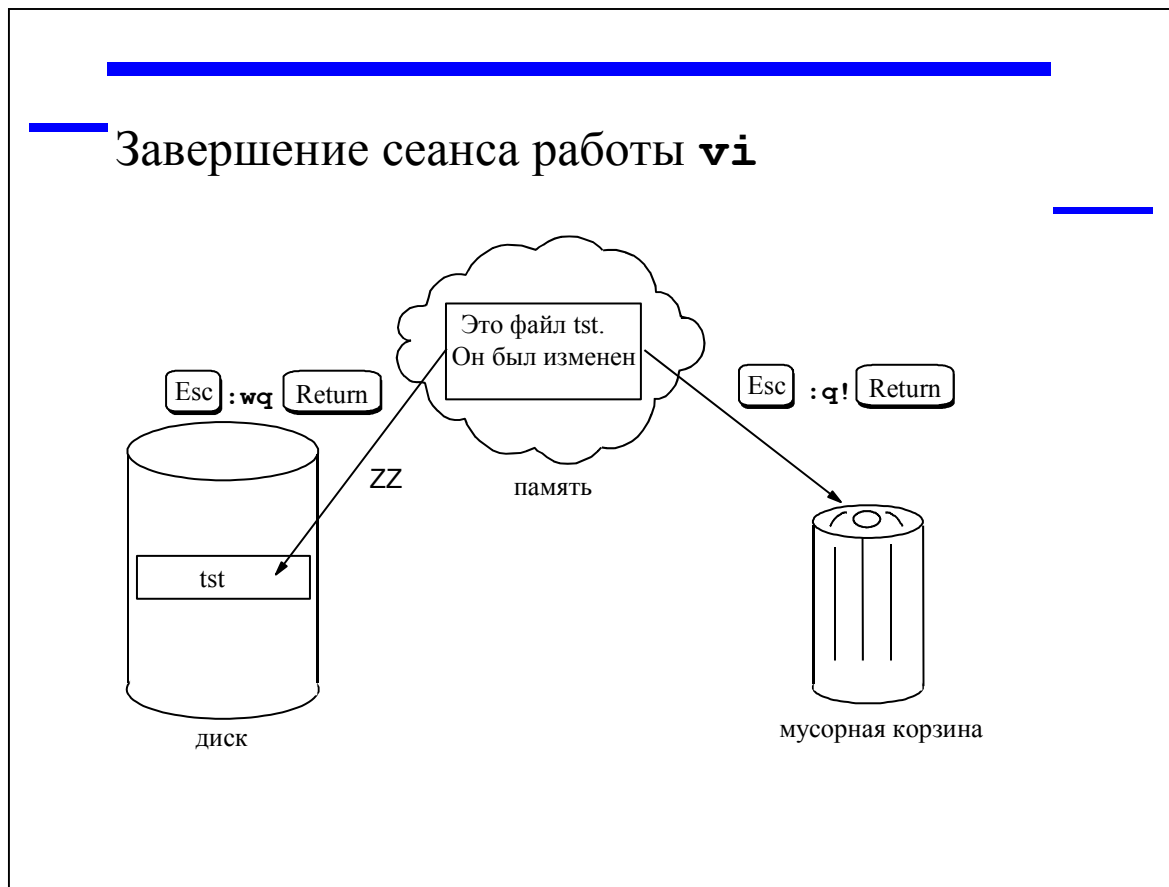
Нужно всегда следить за тем, где находится курсор, так как он указывает на символ, слово или строку, которые будут изменены при редактировании.

Также нужно обращать внимание на то, какое сообщение выводится в области *сообщения о режиме работы*. Редактор может напоминать, например, о том, что он находится в режиме *ввода* или *замены*. Эти визуальные подсказки могут оказать весьма существенную помощь во время сеансов редактирования.

Знак тильда (~), который вы видите на слайде, используется при выводе для пометки свободного пространства. Указанные строки не являются строками файла. Если в конце файла должны быть пустые строки, их необходимо ввести явно.

ПРИМЕЧАНИЕ: Если вы вошли в режим *ввода* и не увидели сообщение о режиме работы редактора, указывающее режим *ввода* (*input mode*), нажмите следующие клавиши: `[Esc] :set showmode [Return]`

14–6. СЛАЙД: Завершение сеанса работы vi



Пояснения

По окончании редактирования файла необходимо сохранить содержимое временного буфера на диске. Это можно сделать с помощью одной из двух команд. Одна из этих команд является командой редактора **vi**, а вторая – командой редактора **ex**. Чтобы выполнить любую из этих команд, необходимо находиться в *режиме команд*; поэтому для подтверждения того, что вы находитесь в режиме команд, не забывайте нажимать клавишу **Esc**.

Esc **ZZ**

команда **vi** — не отображается на экране
поместить файл на свое место на диске

Esc **:wq** **Return**

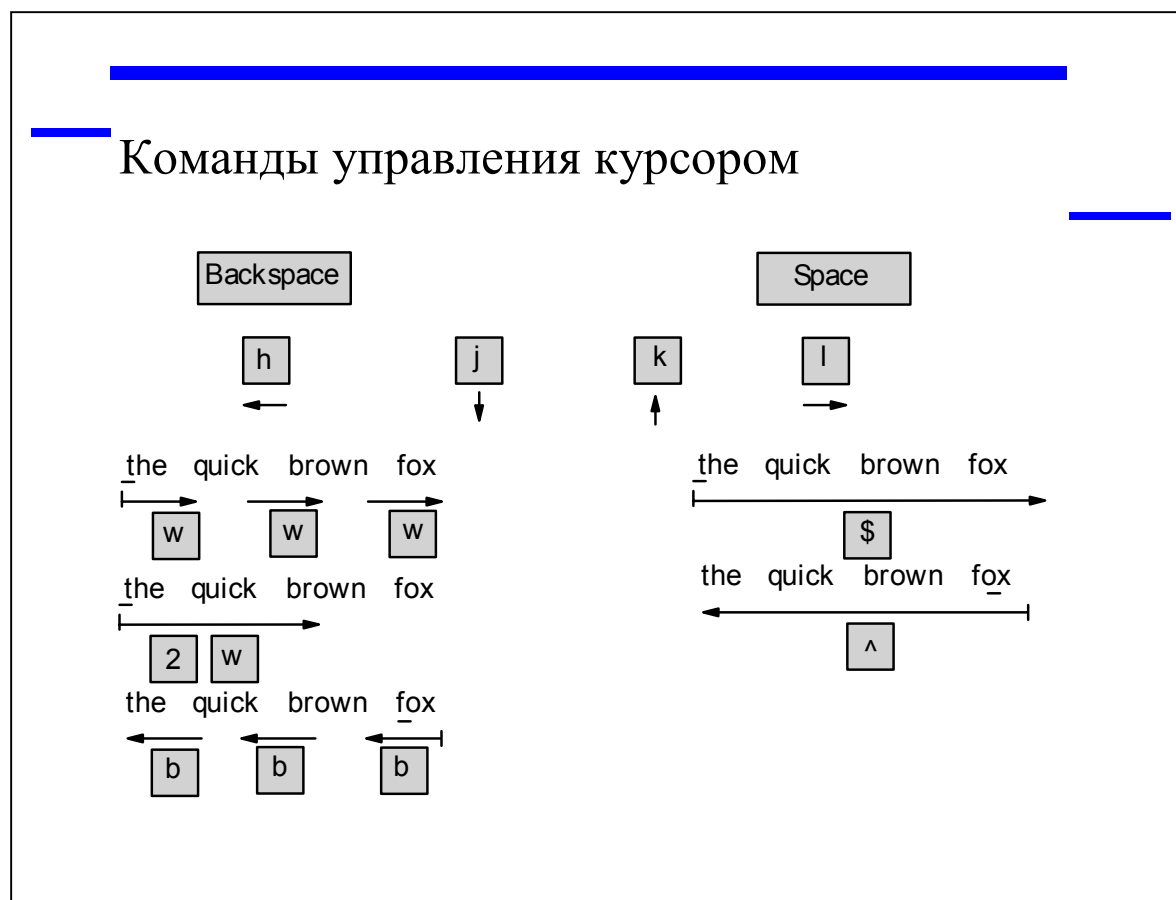
команда **ex** — ей предшествует двоеточие, отображаемое в левом нижнем углу экрана
write and quit – записать и выйти

Всегда могут возникнуть ситуации, когда *не надо* сохранять на диске изменения, внесенные во временный буфер. Для этой цели используется соответствующая команда редактора, которая позволяет отказаться от временного буфера:

Esc **:q!** **Return**

Означает “Выйти! Я имею в виду именно это!” (Quit! I really mean it!
– я знаю, что отказываюсь от всех изменений)

14–7. СЛАЙД: Команды управления курсором



Пояснения

Первая группа команд, которую мы рассмотрим, позволяет перемещать курсор по тексту файла. Следует помнить, что курсор указывает именно на ту позицию файла, в которой нужно выполнять модификацию.

Простое перемещение курсора выполняется с помощью команд **h**, **j**, **k** и **l**. Вспомним, что на телетайпах не было клавиш со стрелками, поэтому перемещение курсора должно было выполняться с помощью других клавиш. Большинство используемых в настоящее время версий редактора **vi**, тем не менее, поддерживают использование для перемещения курсора клавиш со стрелками ($\downarrow$, $\leftarrow$, $\rightarrow$, $\uparrow$).

Большинство команд **vi** используют в названии аббревиатуру слов, имеющих значение, соответствующее выполняемой команде. Если запомнить аббревиатуры и соответствующие им значения, будет легче вспоминать команды. В дальнейшем в кратком описании команд там, где это возможно, будет приведено и смысловое значение команды.

Команды управления курсором (обзор)

h или Backspace	Перемещение на один символ влево
j	Перемещение на одну строку вниз
k	Перемещение на одну строку вверх
l или Space	Перемещение на один символ вправо
# w	Перемещение по словам (word) вперед (при выделении слов команда W игнорирует знаки пунктуации)
# b	Перемещение по словам назад (backwards) (при выделении слов команда B игнорирует знаки пунктуации)
# e	Перемещение на конец (end) следующего слова (при выделении слова команда E игнорирует пунктуацию)
\$	Перемещение на конец текущей строки
^ или 0	Перемещение на начало текущей строки

Многим командам **vi** может предшествовать число, означающее, сколько раз должна быть выполнена эта команда. Поэтому если нужно переместиться вперед на *6 слов*, можно выполнить одну команду **6w**, или если нужно переместиться на *3 слова назад*, достаточно выполнить команду **3b**.

14–8. СЛАЙД: Команды управления курсором (продолжение)



Пояснения

Дополнительные команды перемещения курсора позволяют перемещаться на указанную строку файла и прокручивать экран.

Команды управления курсором (продолжение)

G	Перейти (Go) в конец файла
#G	Перейти (Go) на строку с номером #
: #	Перейти на строку с номером #
Ctrl + g	Вывести номер строки, на которую выполнен переход (текущая строка)
Return	Перейти к первому не пустому символу следующей строки
Ctrl + b	Прокрутить назад (back) к предыдущему окну текста
Ctrl + f	Прокрутить вперед (forward) к следующему окну текста
Ctrl + u	Прокрутить вверх (up) на половину окна текста
Ctrl + d	Прокрутить вниз (down) на половину окна текста

L	Перейти на последнюю (Last) строку экрана
M	Перейти на среднюю (Middle) строку экрана
H	Вернуться на экране домой (Home – первая строка, первый символ экрана)
Ctrl + 1	Перерисовать экран (полезная команда в тех случаях, когда во время сеанса редактирования на экран выводится чье-то сообщение)

ПРИМЕЧАНИЕ: Если во время редактирования файла вы хотите видеть на экране номера строк, нужно выполнить следующую команду:

:set number Return

Отменить вывод на экран номеров строк можно с помощью следующей команды:

:set nonumber Return

14–9. СЛАЙД: Режим ввода: i, a, O, o

Режим ввода: i, a, O, o

quik brown fox
[i] c [ESC] вставка буквы c

quik brown fox
[a] c [ESC] Добавление буквы c

quik brown fox
[I] The [space] [ESC] [A] es [ESC] Добавление букв es

вставка The
Open [O] пустая строка над текущей
верхний регистр
open [o] пустая строка под текущей
нижний регистр

* Помните! [ESC] завершает работу в режиме ввода

Пояснения

Для того чтобы вводить текст в файл, необходимо перейти в *режим ввода*. В редакторе есть несколько команд, которые позволяют переключиться в режим *ввода*.

Команды переключения в режим ввода

- a** добавить (append) новый текст после символа, отмеченного курсором
- i** вставить (insert) новый текст перед символом, отмеченным курсором
- O** открыть (Open) для ввода текста новую строку над строкой, отмеченной курсором
- o** открыть (open) для ввода текста новую строку под строкой, отмеченной курсором
- A** добавить (Append) новый текст в конец текущей строки (отмеченной курсором)
- I** вставить (Insert) новый текст в начало текущей строки (отмеченной курсором)

Когда редактор находится в режиме ввода, в нижнем правом углу экрана появляется сообщение *input mode* (режим ввода).

Как только пользователь переключится в *режим ввода*, он получает возможность вводить в файл текст. В *режиме ввода* нажатие клавиши **Return** приводит к вставке в файл новой

пустой строки. Если нужно разбить некоторую строку на две, нужно переместить курсор в ту позицию, с которой должна начинаться вторая строка, и затем вставить в файл символ возврата каретки (в *режиме команд* нажатие клавиши **Return** *перемещает* курсор на первый не пустой символ в следующей строке).

ПРИМЕЧАНИЕ: Чтобы вернуться в *командный режим*, нужно нажать клавишу **Esc**. Обратите внимание на то, что когда вы переключаетесь из режима ввода в режим команд, курсор возвращается (перемещается влево) на один символ.

Исправление опечаток

Находясь в *режиме ввода*, с помощью клавиши **Backspace** можно вернуться, стирая ввод, к той позиции, в которой допущена ошибка, после чего заново ввести текст. *Обратите внимание:* когда вы с помощью клавиши **Backspace** возвращаетесь по строке, на экране символы не стираются, но они удаляются из буфера.

Клавишу **Backspace** можно использовать для исправления опечаток только во время *текущей* работы в режиме ввода (текущего сеанса ввода). Нажатие клавиши **Esc** завершает текущий сеанс ввода. Для исправления ошибок, допущенных во время предыдущих сеансов ввода, необходимо использовать соответствующие команды редактора **vi**.

14–10. СЛАЙД: Удаление текста: x, dw, dd, dG

Удаление текста: x, dw, dd, dG

There are tooooo many worods

x **3 x** **x**

There are too many words

d w

There are many words

2 d w

d d → many characters
2 d d → many words
 → many lines
d G → many pages
 → many chapters
 → many volumes
 → последняя строка файла

*Внимание: **u** отменяет последнее изменение
U восстанавливает текущую строку к исходному виду

Пояснения

Для удаления текста используются две команды:

#x удаление (**x** out) символа, отмеченного курсором

#добъект удаление (**delete**) указанного объекта

Команда **d** (удаление) требует указания того объекта, который надо удалить. Удаляется именно указанный объект. Удаляемые объекты идентифицируются с помощью команд перемещения курсора.

Команды удаления

#dw удалить текущее слово (**delete word**)

#dd удалить текущую строку (соглашение **vi**: если действие, определяющее команду, повторяется, команда воздействует на всю строку)

dG удалить все строки, начиная с текущей и кончая последней строкой файла

d\$ удалить конец строки (от текущей позиции и до конца строки)

d^ удалить начало строки (от начала строки и до символа, предшествующего текущему)

Командам удаления также может предшествовать число, указывающее, сколько раз должна быть выполнена указанная команда. Поэтому, если надо *удалить 6 слов*, достаточно выполнить команду **6dw**; если нужно *удалить 3 строки*, достаточно выполнить команду **3dd**.

ПРИМЕЧАНИЕ: Обращайте внимание на положение курсора. Большинство объектов определяется по отношению к текущему положению курсора.

Команда отмены (Undo)

Начинающие пользователи **vi** могут удалить или модифицировать то, что не должно было быть удалено или модифицировано. В таких ситуациях, несомненно, на помощь придет команда **u** (undo).

u отменить (**undo**) последнее изменение

U отменить (**Undo**) *все* изменения *текущей строки*

Команда **u** отменяет последнее изменение, выполненное в строке файла. Если сразу же выполнить вторую команду **u**, она отменит предыдущую команду (отмена отмены); в результате строка файла вернется к тому состоянию, которое было перед выполнением первой команды отмены. Если в строку текста внесено несколько изменений, можно выполнить команду **U**; она вернет строку текста в то состояние, которое было перед началом редактирования этой строки (т.е. в момент, когда курсор впервые перед выполнением всех изменений переместился на данную строку). Для того чтобы команда **U** отменила все изменения, она должна быть выполнена *прежде, чем курсор будет перемещен на другую строку*.

14–11. УПРАЖНЕНИЯ: Ввод и удаление текста, перемещение курсора

Указания

Используя **vi**, начните сеанс редактирования файла **tst** из вашего *HOME* каталога. Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Запустите **vi** для файла **tst**.
2. Вставьте слово *only* между словами *will be*.
3. Добавьте слова *many, many* в конце строки "*It will be used for*".
4. Добавьте новую строку в конец файла. Введите в ней свое имя. По окончании ввода *HE* НАЖИМАЙТЕ Esc!
5. Используя клавишу Backspace, удалите свое имя и введите имя своего коллеги.
6. Откройте новую строку в начале файла (ПОДСКАЗКА: **o**).

7. Напечатайте **12345**.
8. Нажмите два раза `Backspace`. Исчезли с экрана какие-либо цифры?
9. Введите **1234**. Что произошло с теми цифрами, для которых нажималась клавиша `Backspace`?
10. Нажмите `Backspace` три раза.
11. Нажмите `Esc`. Что произошло с символами, для которых нажималась клавиша `Backspace`? Где теперь находится курсор?
12. Напечатайте четыре буквы **a**. Сколько букв **a** появилось на экране? Почему?

13. Нажмите `Backspace` 5 раз. Что произошло? Почему?

14. Нажмите `Esc`. Что произошло?

15. Завершите сеанс работы `vi`, сохранив все изменения в файле `tst`.

14–12. СЛАЙД: Перемещение текста: **p**, **P**

Перемещение текста: **p**, **P**

these are too words

d

w

ddelete word – удаление слова

these are words_

p

нижний регистр paste after – вставка после

these are words too

here is one mroe

x

x

p

меняет символы местами

here is one moe

p

нижний регистр

here is one more

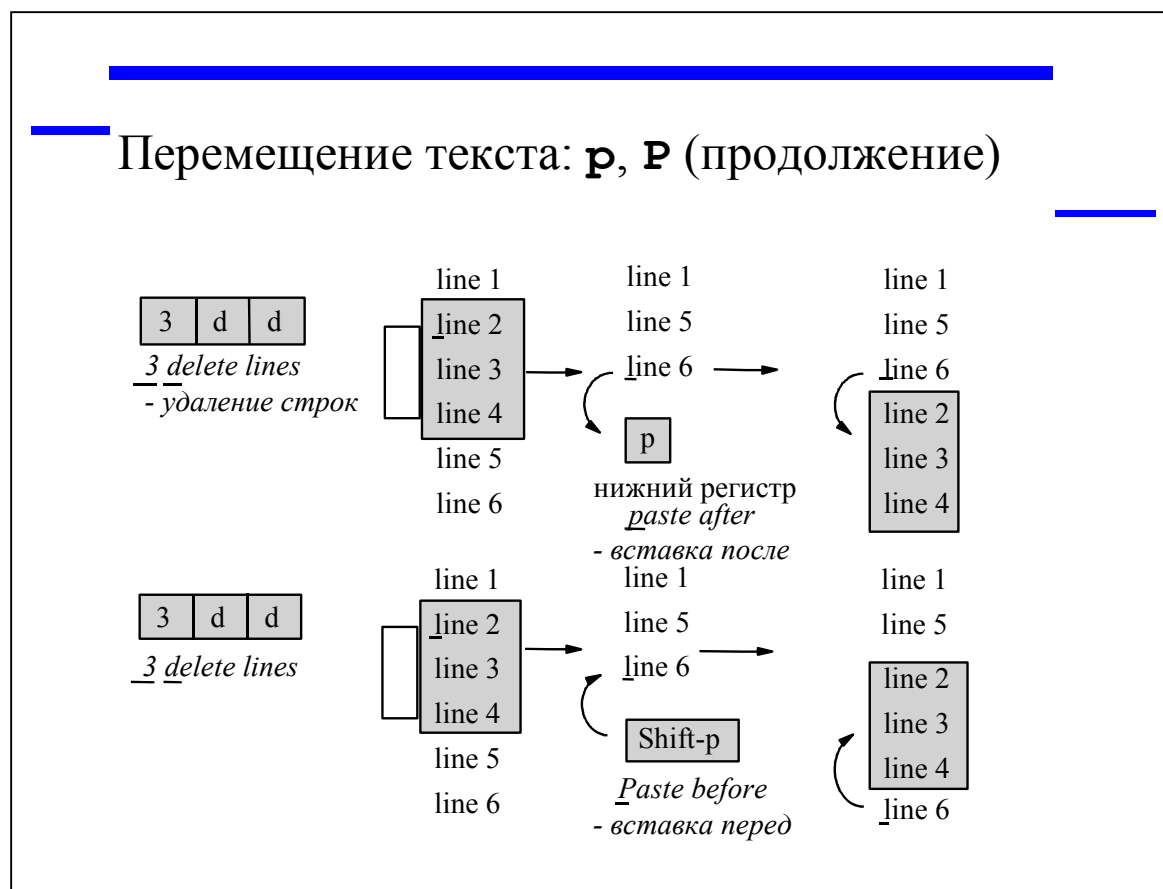
Пояснения

Когда удаляется какой-либо объект, он запоминается в специальном временном буфере обмена. При выполнении следующей операции удаления вновь удаленный объект заменит старое содержимое этого буфера. Команда **p** (paste – вставка) позволяет восстановить текст из буфера обмена и вставить его в файл. Текст может быть вставлен в любом месте файла, определяемом положением курсора.

Можно легко перемещать текст с одного места в файле в другое, если удалить нужный фрагмент текста, записав его тем самым в буфер обмена, переместить курсор в нужное место файла и вставить содержимое буфера в этом месте, отмеченном курсором.

Так как содержимое буфера обмена замещается при выполнении следующей операции удаления, нужно быть очень внимательным и восстанавливать его содержимое *перед* выполнением очередного удаления.

14–13. СЛАЙД: Перемещение текста: **p**, **P** (продолжение)



Пояснения

Команды удаления и вставки

добъект удаление **объекта** и занесение его в буфер обмена

p (нижний регистр) вставка (**paste**) содержимого буфера обмена в текст *после* курсора

P (верхний регистр) вставка (**Paste**) содержимого буфера обмена в текст *перед* курсором

Операции удаления и вставки позволяют легко перемещать фрагменты текста или отдельные символы. Если в буфере обмена находятся строки, команда **p** (нижний регистр) откроет новые строки *под* текущей строкой, а команда **P** (верхний регистр) – *над* текущей строкой, после чего в эти строки будет вставлено содержимое буфера.

14–14. СЛАЙД: Копирование текста: **yw**, **yy**

Копирование текста: **yw**, **yy**

these are words too

y w

yank word – копирование слова

theseare words too

p

paste after – вставка после

these words are words too

3 y y

3 yank lines – копирование строк

line 1

line 2

line 3

line 4

line 5

line 6

line 1

line 2

line 3

line 4

line 5

line 6

line 1

line 2

line 3

line 4

line 5

line 6

line 2

line 3

line 4

line 2

line 3

line 4

paste after

нижний регистр

Пояснения

Команда **y** (*yank* – дергать) также является командой, требующей указания объекта, с которым нужно выполнить операцию. Указанный объект заносится (копируется) в буфер обмена. Можно легко вставить этот текст в другое место файла, используя команды перемещения курсора и вставки содержимого буфера обмена. Эта операция похожа на операцию перемещения.

Команды занесения в буфер обмена

#yw	занести в буфер обмена (yank) текущее слово (word)
#yy	занести в буфер обмена текущую строку (соглашение vi : если действие, определяющее команду, повторяется, команда воздействует на всю строку)
yG	занести в буфер обмена все строки, начиная с текущей и кончая последней строкой файла
y\$	занести в буфер обмена конец строки (от текущей позиции и до конца строки)
y^	занести в буфер обмена начало строки (от начала строки и до символа, предшествующего текущему)

Команды копирования и вставки

y*объект* занести в буфер обмена (**yank**) указанный **объект**

p (нижний регистр) вставить (**paste**) содержимое буфера обмена *после* курсора

P (верхний регистр) вставить (**Paste**) содержимое буфера обмена *перед* курсором

Операции копирования и вставки позволяют легко копировать фрагменты текста. Если в буфере обмена находятся строки, команда **p** (нижний регистр) откроет новые строки *под* текущей строкой, а команда **P** (верхний регистр) – *над* текущей строкой, после чего в эти строки будет вставлено содержимое буфера.

14–15. СЛАЙД: Изменение текста: **r**, **R**, **cw**, **.**

Изменение текста: **r**, **R**, **cw**, **.**

the quick brown fox

r u r o

***r**replace – заменить символ*

the quick brown fox



c w silver-gray ESC

***cw**change word – заменить слово*

the quick silver-gray fox



c w slow ESC

***cw**change word*

the slow silver-gray fox



R brand new text ESC

***R**big Replace – длинная замена*

the brand new textay fox

* Примечание: . позволяет повторить последнее изменение

Пояснения

Умение вводить и удалять текст позволяет вносить в файл любые изменения. Но такая процедура внесения изменений является весьма громоздкой, так как постоянно приходится переключаться вручную между режимом ввода и режимом команд. Процедура внесения изменений может оказаться более удобной, если использовать специальные команды, предназначенные для изменения текста.

В редакторе **vi** есть три базовые команды, позволяющие изменять текст:

r СИМВОЛ заменить (**replace**) символ, отмеченный курсором, указанным **СИМВОЛОМ**

R заменить (большая замена – **REPLACE**) все символы, пока не будет нажата клавиша ESC (редактор вместо режима вставки использует при вводе символов режим замены)

с_{объект} изменить (**change**) указанный **объект**. Указанный объект замещается вводимым текстом (конец объекта помечается символом **\$**). Окончание ввода текста также отмечается нажатием клавиши ESC.

Команда **c** (изменение) также является командой, требующей указания объекта, с которым нужно выполнить операцию. Объекты идентифицируются с помощью команд перемещения курсора.

Команды изменения

#cw	изменить (change) текущее слово (word)
#cc	изменить всю текущую строку (дублирование действия — помните?)
cG	изменить все строки, начиная с текущей и кончая последней строкой файла
c\$	изменить конец строки (от текущей позиции и до конца строки)
c^	изменить начало строки (от начала строки и до символа, предшествующего текущему)

Команда "точка" (.)

Команда «точка» — одна из наиболее хитрых команд, входящих в состав команда редактора **vi**. Она позволяет повторить последнюю операцию изменения текста (к этим операциям относится также и операция удаления).

Пример

Предположим, что некоторая строка файла имеет такой вид:

```
walk walk walk walk walk walk
```

а нам нужно, чтобы она выглядела так:

```
run walk run walk run walk
```

Можно, конечно, просто удалить эту строку и ввести нужную. Но посмотрите, что может сделать для вас команда «точка»:

1. Переместите курсор под начало первого слова, которое нужно заменить.
2. Выполните команду редактора **vi** «заменить слово» **cw**, затем введите: **run** ESC
3. Переместите курсор под начало следующего слова, которое нужно заменить: **ww**
4. Теперь вместо того, чтобы выполнять команду **cw**, просто введите точку (.). Это приведет к повторению последнего изменения, а этим изменением было *заменить текущее слово текста словом run*. Такую операцию можно повторять столько раз, сколько нужно.

14–16. СЛАЙД: Поиск текста: /, n, N

Поиск текста: /, n, N

There is one here
and one more here
and yet one more
but not this **ONE**
nor this One

/one <cr>

n

next – следующее

N

предыдущее

Пояснения

В общем случае при редактировании файла всегда приходится *искать* некоторую конкретную строку текста. Команда / позволяет найти место в тексте размещения (вхождение) заданной строки. Команда n позволяет найти в тексте **следующее** вхождение этой строки.

Команды поиска текста

/ текст	Поиск текста циклически по всему файлу, начиная с текущей строки файла в направлении вниз до конца файла с последующим переходом на его начало
? текст	Поиск текста циклически по всему файлу, начиная с текущей строки файла в направлении вверх к началу файла с последующим переходом на его конец
n	поиск следующего вхождения заданной строки текста (файл просматривается в направлении, определенном предшествующей командой поиска)
N	поиск предыдущего вхождения заданной строки текста (файл просматривается в направлении, обратном определенному предшествующей командой поиска)

Циклический просмотр файла означает, что если указанный текст не был найден до конца файла – поиск по /, поиск продолжается с его начала (или, если указанный текст не был найден до начала файла – поиск по ?, поиск продолжается с его конца).

14–17. СЛАЙД: Поиск текста по модели

Поиск текста по модели

<code>[oO]ld_text</code>	Искать <i>old_text</i> и <i>Old_text</i>
<code>^текст</code>	Искать текст , расположенный в начале строки
<code>текст\$</code>	Искать текст , расположенный в конце строки
<code>.</code>	Искать один любой символ
<code>СИМВОЛ*</code>	Искать нуль или более подряд расположенных СИМВОЛОВ
<code>. *</code>	Искать нуль или более подряд расположенных любых символов

Пояснения

Как было показано на предыдущем слайде, при поиске строк учитывается, в каком регистре вводятся буквы. В предыдущем примере поиск завершался успешно только в том случае, если обнаруживалось точное соответствие символов текста. Конструкции, приведенные на данном слайде, позволяют выполнять поиск строк, соответствующих некоторым моделям. Такие модели известны как **регулярные выражения** и распознаются различными утилитами системы UNIX.

Правила записи регулярных выражений

<code>[a-zA-Z0-9]</code>	Определяет класс символов, из которого выбираются символы для сопоставления. Запись a-z означает диапазон символов, из которого выбираются символы для сопоставления. Скобки [] определяют только <i>один</i> символ текста.
<code>^текст</code>	Прикрепляет текст к началу строки
<code>текст\$</code>	Прикрепляет текст к концу строки
<code>.</code>	Соответствует одному любому символу
<code>СИМВОЛ*</code>	соответствует любому числу повторений СИМВОЛА , в том числе и нулевому

Примеры

<code>/[Tt]he</code>	Поиск вхождения строк (цепочек символов) <i>The</i> или <i>the</i> .
<code>/[oO][nN][eE]</code>	Поиск вхождения строки (цепочки символов) <i>one</i> , в записи которой использованы буквы в любом регистре
<code>/bo*t</code>	Поиск вхождения строки (цепочки символов), начинающейся буквой <i>b</i> , за которой следует любое количество букв <i>o</i> (в том числе и нулевое), и заканчивающейся буквой <i>t</i> . Такой модели поиска соответствуют цепочки символов <i>bt</i> , <i>bottom</i> , <i>boot</i> , <i>booot</i> , и так далее, размещенные в любом месте строки.
<code>/^[abc].*</code>	Поиск строки (line), которая начинается одним из символов <i>a</i> , <i>b</i> или <i>c</i> . Приведенное выражение читается так: строка, которая начинается буквой <i>a</i> , или <i>b</i> , или <i>c</i> , за которой следуют нуль или более любых символов. Такой модели поиска соответствуют строки “ <i>a</i> ”, “ <i>after morning</i> ”, “ <i>bc</i> ” и т.д., но не соответствуют строки “ <i>-a</i> ”, “ <i>xbc</i> ” и т.д.
<code>/finally.\$</code>	Поиск строки (line), которая заканчивается словом <i>finally</i> , за которым следует только один любой символ. Такой модели поиска соответствует строка, заканчивающаяся так: “... <i>finally.</i> ”, а также строки, заканчивающиеся на “ <i>finallyA</i> ” или “ <i>finallyZ</i> ”.

14–18. СЛАЙД: Глобальный поиск и замена — команды **ex**

Глобальный поиск и замена — команды **ex**

```
:m,ns/модель/новый_текст/  
:1,$s/one/two/
```

В каждой строке файла, начиная с первой и до конца файла (\$) заменить *только первое* вхождение найденного слова "**one**" строкой "**two**".

```
:m,ns/модель/новый_текст/g  
:.,10s/[oO][nN][eE]/two/g
```

В каждой строке файла, начиная с текущей и до 10 строки, заменить *каждое* (**global** – глобально) вхождение слова "**one**", записанного прописными или строчными буквами, строкой "**two**".

Пояснения

Возможность выполнения операций глобального поиска и замены в **vi** реализуется за счет использования команд редактора **ex**. **ex** представляет собой строковый редактор, который для выполнения необходимых операций над файлом использует адреса строк. Ниже приведен формат команды глобального поиска и замены, в котором показаны различные синтаксические конструкции команды **ex**:

```
:m,ns/модель/новый_текст/g
```

m и n	определяют строки файла, с которыми будет выполняться операция
s	определяет команду замены (substitute)
модель	задает <i>модель поиска</i> текста
новый_текст	задает <i>строку</i> , которой будет заменен найденный фрагмент текста, соответствующий модели
g	требуется глобальное (globally) выполнение команды по всей строке

Примеры

```
:1,$s/one/two/
```

Во всех строках файла (с 1-ой и до конца файла – **1,\$**) заменить *только первое вхождение* слова *one* в каждой строке словом *two*.

```
:.,10s/[oO][nN][eE]/two/g
```

В строках файла, начиная с текущей и по 10-ю, заменить глобально по всей строке каждое вхождение слова *one*, записанного с помощью букв верхнего и/или нижнего регистров, словом *two*.

14–19. СЛАЙД: Другие команды ex

Другие команды ex

:w	Записать текущий буфер на диск
:m, n w файл	Записать строки с m и до n текущего буфера в файл
:w файл	Записать текущий буфер в файл
:e файл	Загрузить файл в буфер редактора, заменив старое состояние буфера
:e!	Отменить <i>все</i> изменения, загрузив в буфер редактируемый файл с диска
:r файл	Вставить содержимое файла после текущего положения курсора
:! cmd	Выполнить команду shell'a cmd
:set all	Показать все опции сеанса редактирования
:set nu	Включить режим нумерации строк

Пояснения

Существуют специальные опции, облегчающие редактирование с помощью **vi**. Эти опции устанавливаются соответствующей командой редактора **ex**. Для того чтобы *включить* опцию, используется команда

```
:set опция
```

Для того чтобы *выключить* опцию, используется команда

```
:set noопция
```

Перечень всех опций и их значения можно вывести с помощью команды

```
:set all
```

Некоторые часто используемые опции представлены на слайде. Ниже приводятся и другие опции.

:set autoindent	при нажатии клавиши Return в режиме ввода устанавливать курсор в позицию с отступом, соответствующим отступу в предыдущей строке; этот режим отменяется нажатием клавиш Ctrl + d
------------------------	---

<code>:set tabstop = n</code>	перемещать курсор при нажатии клавиши <code>Tab</code> на <i>n</i> позиций
<code>:set wrapmargin = n</code>	за <i>n</i> символов до конца строки автоматически прерывать ввод слова и вставлять символ новой строки
<code>:set showmatch</code>	показывать соответствующую открывающую скобку (<code>{</code> , <code>(</code> , <code>[</code>), когда вводится закрывающая скобка (<code>}</code> , <code>)</code> , <code>]</code>)
<code>:set redraw</code>	перерисовывать экран vi после каждого обновления экрана. При использовании устройства с малой скоростью передачи данных (с модемом) можно эту опцию отключить. Можно выполнить принудительно перерисовку экрана с помощью команды vi <code>Ctrl + r</code> .
<code>:set showmode</code>	включить вывод сообщения о режиме работы
<code>:map</code>	вывести карту отображения клавиш, используемую в режиме команд
<code>:map!</code>	Вывести карту отображения клавиш, используемую в режиме ввода

Если желательно, чтобы каждый раз, когда вызывается **vi**, некоторые опции устанавливались автоматически, нужно в своем *HOME* каталоге создать файл **.exrc** и включить в него следующие строки:

`set опция` (обратите внимание на то, что здесь отсутствует двоеточие)

14–20. ТЕКСТОВАЯ СТРАНИЦА: Команды vi: обзор

Управление курсором	Режим ввода	Удаление текста	Изменение текста	Запись и выход
h или Backspace	a	x	r	:wq
j	i	dw	R	ZZ
k	o	dd	cw	:q!
l или Space	O	dG	cc	:w <i>файл</i>
w	A	d\$	c\$	:m,n w <i>файл</i>
b	I	d^	c^	:e!
e				:e <i>файл</i>
\$				
^				
G				
#G		Копирование текста	Вставка текста	Дополнительные команды
:#		yw	р (нижний регистр)	u
Ctrl + g		yy	Р (верхний регистр)	U
Return		y\$		.
Ctrl + b		y^		/ <i>текст</i>
Ctrl + f				n
Ctrl + u				: !команда
Ctrl + d				
L				
M				
H				
Ctrl + l				

14–21. УПРАЖНЕНИЯ: Модификация текста

Указания

Используйте редактор **vi** для выполнения следующих упражнений:

1. Начните сеанс **vi** редактирования файла **vi.tst**; выполните те изменения, которые указаны в тексте этого файла. Ниже приводится текст файла **vi.tst** (в скобках дается перевод предлагаемых действий).

```
Enter your name here -> (Введите здесь свое имя)
```

(Замените в следующих предложениях названия цвета, цветка и книги своими)

```
Change the following to your favorite color -> lavender
```

```
Change the following to your favorite flower -> rose
```

```
Change the following to your favorite book -> A Tale of Two Cities
```

(Исправьте опечатки в следующих двух строках:)

```
Correct the typos in the next two lines:
```

```
Corect teh typooos im thiss line.
```

```
Ther awe mroe mistakkes in thsi linne.
```

(Приведенные выше две строки должны читаться так)

```
The above two lines should read:
```

```
Correct the typos in this line.
```

```
There are more mistakes in this line.
```

(удалите в следующей строке все вхождения слова “jog”)

```
Delete every occurrence of the word "jog" in the next line:
```

```
walk jog run walk jog run walk jog run walk jog run
```

(В предыдущей строке замените каждое слово “walk” словом “WALK”)

```
Change every occurrence of the word "walk" to "WALK" in the above line.
```

```
line1  
line2  
line3  
line4  
line5  
line6  
line7  
line8
```

(Выполните следующие упражнения с представленными выше строками – от 1-й до 8-й)

```
Complete the following exercises on line1 through line8 above:
```

(Удалите строки, содержащие от line1 до line5, и вставьте их после строки, содержащей line8)

1. Move the lines containing line1 through line5 and paste them after the line containing line8.

(Скопируйте строки, содержащие от line2 до line4, и вставьте их перед строкой, содержащей line6, а также после строки, содержащей line3)

2. Copy the lines containing line2 through line4 and paste them before

the line containing line6, and also after the line containing line3.

(Завершите сеанс редактирования файла vi.tst, сохранив в нем все сделанные изменения)
Quit your edit session on "vi.tst" saving the changes that you have made.

2. Начните новый сеанс редактирования файла **funfile** из вашего *HOME* каталога и замените в файле все вхождения слова *bug* словом *FEATURE*.
3. Запишите первые 40 строк файла **funfile** в файл **new.40**.
4. Перейдите к последней строке файла **funfile**.
5. Найдите и выполните команду, которая перемещает курсор в нижнюю половину экрана. Замените половину экрана следующей строкой:
This file is silly.
6. Не выходя из **vi**, запишите новую версию файла в файл **funfile.123**.

7. Не выходя из **vi**, загрузите в буфер редактора файл **new.40**, перезаписав прежнее состояние буфера.
8. Используя опцию *number* редактора **ex**, включите режим нумерации строк.
9. Найдите в файле **new.40** вхождение слова *FEATURE*.
10. Замените найденное слово *FEATURE* словом *BUG* и сохраните файл в файле **new.new.40**.
11. Скопируйте файл **funfile** в **funfile.new**. В файле **funfile.new** найдите все вхождения строк *System* или *system* и с помощью */*, **cw**, **n** и **.** замените все эти слова, кроме одного, словом *XXXXX*.
12. Сохраните результаты редактирования и выйдите из редактора **vi**.

Модуль 15 — Управление процессами

Цели:

Изучив этот модуль, вы сможете:

- Использовать команду **ps**
- Запускать процесс в фоновом режиме
- Наблюдать за исполняющимися процессами с помощью команды **ps**
- Запускать фоновый процесс, который не реагирует на сигнал **hangup**, генерируемый при завершении сеанса работы в системе (log off)
- Переводить процесс из фонового режима в оперативный
- Приостанавливать процесс
- Завершать выполнение процессов, посылая им соответствующие сигналы

15–1. СЛАЙД: Команда **ps**

Команда **ps**

Синтаксис:

ps [-efl] Выводит информацию о состоянии процессов

Примеры:

```
$ ps
```

PID	TTY	TIME	COMMAND
1324	ttyp2	0:00	sh
1387	ttyp2	0:00	sh

```
$ ps -ef
```

UID	PID	PPID	C	STIME	TTY	TIME	COMMAND
root	0	0	0	Jan 1	?	0:20	swapper
root	1	0	0	Jun 23	?	0:00	init
root	2	0	0	Jun 23	?	0:16	vhand
root	3	0	0	Jun 23	?	12:14	statdaemon
user3	1324	1	3	18:03:21	ttyp2	0:00	-sh
user3	1390	1324	22	18:30:23	ttyp2	0:00	ps -ef

Пояснения

Каждому процессу, который создается в системе, присваивается уникальный числовой идентификатор, известный как идентификатор процесса (Process ID – **PID**). Команда **ps** выводит информацию о процессах, которые в настоящее время выполняются в системе (или находятся в состоянии ожидания), включая PID каждого процесса и PID каждого родительского процесса (Parent Process ID – **PPID**). Анализируя значения PID и PPID, можно проследить «линию жизни» каждого процесса, исполняющегося в системе. Команда **ps** выводит также информацию о том, кто является владельцем каждого процесса и с какого терминала каждый процесс запущен.

Команда **ps** обычно вызывается без опций, что позволяет получить краткую информацию только о тех процессах, которые связаны с управляющим терминалом пользователя. Опция **-e** позволяет получить информацию обо всех процессах, существующих в системе, а не только принадлежащих данному пользователю. Опции **-f** и **-l** позволяют получить полную (full) информацию и информацию в длинном (long) формате, которые включают в себя дополнительные сведения о процессах.

На слайде приведены два варианта вызова команды **ps**. Первый вариант вызова команды позволяет получить информацию о процессах, ассоциированных с управляющим терминалом пользователя. Как и следовало ожидать, процессами, ассоциированными с управляющим терминалом, являются shell (входной интерактивный shell) и команда **ps**, которая выполняется в системе в данный момент времени.

Во втором примере показана часть информации, выводимой командой **ps** при создании *полного* (опция **-f**) листинга о *каждом* (опция **-e**) процессе, существующем в системе.

ПРИМЕЧАНИЕ: Следует иметь в виду, что команда **ps** очень интенсивно использует центральный процессор, что приводит к задержке ее ответа.

15–2. СЛАЙД: Фоновые процессы

Фоновые процессы

Синтаксис:

командная_строка > *файл.out* &

Пример:

```
$ grep user * > grep.out &  
[1] 194
```

```
$ ps  
  PID  TTY    TIME  COMMAND  
  164  tty2  0:00   sh  
  194  tty2  0:00  grep  
  195  tty2  0:00   ps
```

Пояснения

Задание командной строки в следующем виде:

командная_строка > *файл.out* &

приводит к выполнению следующих действий:

- планируется запуск **командной строки** в виде задания, исполняющегося в фоновом режиме;
- первичное приглашение shell'а выводится на экран сразу же после запуска задания;
- выполняется перенаправление стандартного вывода команды, запущенной в фоновом режиме, в *файл.out*, в результате чего вывод команды не будет накладываться на результаты выполнения интерактивных команд;
- фоновые процессы завершаются при завершении сеанса работы пользователя в системе. Когда пользователь попытается выйти из системы, он увидит на экране предупреждение: "There are running jobs". Чтобы завершить сеанс работы, необходимо еще раз выполнить команду **exit** или нажать клавиши **Ctrl** + **d**.

Некоторые задачи, например, поиск некоторого файла по всему диску, или использование какого-либо текстового процессора для форматирования и печати текстовых документов, требуют для своего исполнения длительного времени. Операционная система UNIX

позволяет запустить выполнение программы, требующей большого времени, в фоновом режиме, в котором сама система будет управлять выполнением таких программ. В данном случае *shell* *не будет* ждать завершения выполнения команды, запущенной в фоновом режиме, и выведет на экран свое первичное приглашение сразу же, как только команда будет запущена, позволяя пользователю выполнять другие действия.

Чтобы запустить выполнение команды в фоновом режиме, нужно закончить командную строку символом «амперсant» (&). Обычно для команд, запускаемых на исполнение в фоновом режиме, следует выполнять перенаправление стандартного вывода, чтобы вывод, генерируемый этими командами, не мешал интерактивному сеансу работы. Если перенаправление вывода не сделано, любые результаты, которые обычно команда, выполняющаяся в фоновом режиме, выводит в стандартный вывод, будет отображаться на экране терминала.

Так как стандартным вводом управляет *shell*, команды, которые выполняются в фоновом режиме, не могут вводить исходные данные со стандартного ввода. Поэтому при запуске в фоновом режиме команды, которая вводит данные со стандартного ввода, необходимо выполнить перенаправление ввода, чтобы команда читала необходимые ей данные из файла.

Когда *shell* переводит команду в фоновый режим, он сообщает для этой команды номер задания и идентификатор процесса, если установлена опция **monitor** (**set -o monitor**). Номер задания идентифицирует фоновый процесс в рамках текущего сеанса работы с системой, тогда как идентификатор процесса уникально определяет процесс среди всех процессов и назначается системой UNIX для каждого исполняющегося в системе процесса. Наличие опции **monitor** обуславливает также вывод сообщения на экран при завершении фонового процесса:

```
[1]+ Done grep user * > grep.out &
```

Так как команда, которая выполняется в фоновом режиме, не имеет связи с клавиатурой терминала, нельзя завершить процесс, выполняющийся в фоновом режиме, нажатием клавиши прерывания **Ctrl** + **c**. Фоновый процесс можно завершить с помощью команды **kill** или завершив сеанс работы в системе.

ПРИМЕЧАНИЕ: Для фонового процесса настоятельно рекомендуется выполнять перенаправление *всех* стандартных потоков — ввода, вывода и протокола.

ПРИМЕЧАНИЕ: Фоновое задание может состоять из нескольких команд. Если заключить список команд в круглые скобки (**cmd1**; **cmd2**; **cmd3**), тогда операционная система будет рассматривать этот список как одно задание.

15–3. СЛАЙД: Перевод заданий в фоновый/оперативный режим

Перевод заданий в фоновый/оперативный режим

jobs



Вывести список выполняющихся заданий
Приостановить задание, исполняющееся в
оперативном режиме: **stty susp ^Z**

fg [%номер]

Перевести в оперативный режим задание с
указанным *номером*, или

fg [%строка]

задание, запущенное командной строкой,
начинающейся указанной *строкой*

bg [%номер]

Перевести в фоновый режим задание с
указанным *номером*, или

bg [%строка]

задание, запущенное командной строкой,
начинающейся указанной *строкой*

Пояснения

В POSIX shell'е процессы могут выполняться в оперативном или фоновом режиме. Если в данный момент времени в оперативном режиме исполняется длительный процесс, можно ввести символ приостановки *susp* (обычно это **[Ctrl] + z**). Символ приостановки обычно определяется в файле **.profile** с помощью команды **stty susp ^Z** и устанавливается при входе в систему. Ввод этого символа позволяет временно приостановить оперативный процесс и получить на экране первичное приглашение shell'a. Теперь для того, чтобы перевести задание в фоновый режим, можно выполнить команду **bg %номер** или **bg %строка**. Здесь *номер* – это номер задания, который возвращает команда **jobs**, а *строка* – это начальные символы командной строки, из которой запущено задание.

Аналогично, для того чтобы перевести в оперативный режим задание, которое выполняется в фоновом режиме, нужно использовать команду **fg**. Команда, исполняющаяся в оперативном режиме, будет теперь управлять терминалом до тех пор, пока она не завершится или не будет приостановлена.

15–4. СЛАЙД: Команда **nohup**

Команда **nohup**

Синтаксис:

nohup *командная_строка* & Защищает команду от получения сигнала hangup при завершении сеанса связи (logout) – no hangup

Пример:

```
$ nohup cat * > bigfile &
[1] 972
$  + 
login: user3
Password:
.
.
$ ps -ef | grep cat
UID      PID    PPID      COMMAND
user3   972      1      cat * > bigfile &
```

Пояснения

Команда **nohup** операционной системы UNIX позволяет команде не завершать свою работу, если возникает разрыв линии или пользователь завершает работу в системе. Команда **nohup** относится к группе **префиксных команд** системы UNIX, которые указываются перед другой командой. Наиболее часто команда **nohup** используется с теми командами, которые должны выполняться в фоновом режиме. Следует помнить, что завершение сеанса работы обычно приводит к завершению фоновых заданий. Если же команда, которая должна выполняться в фоновом режиме, запускается вместе с **nohup**, пользователь может завершить сеанс работы, а система UNIX продолжит выполнение процесса, несмотря даже на то, что родительский процесс shell более не исполняется. Следует обратить внимание на то, что когда shell, являющийся родительским процессом для команды, запущенной вместе с **nohup**, завершается, команда принимается («усыновляется») процессом с идентификатором 1 (процесс **init**). Позднее можно войти в систему и увидеть состояние или результаты выполнения команды, запущенной вместе с **nohup**.

Обычно при использовании команды **nohup** выполняется перенаправление вывода в файл. Если пользователь *не укажет* выходной файл, команда **nohup** автоматически выполнит перенаправление вывода в файл **nohup.out**. Следует отметить, что в файл **nohup.out** будут записываться результаты, выводимые и в **stdout**, и в **stderr**.

Система Linux



В системе Linux любой процесс, выполняемый в фоновом режиме, выполняется так, как будто перед ним указана команда **nohup**. Это делает лишним использование в командной строке в среде Linux команды **nohup**.

15–5. СЛАЙД: Команда **nice**

Команда **nice**

Синтаксис:

nice [**-N**] *командная_строка* запускает процесс
с пониженным приоритетом
N – число в диапазоне
от 1 до 19

Примеры:

```
$ nice -10 cc myprog.c -o myprog
$ nice -5 sort * > sort.out &
$
```

Пояснения

Операционная система UNIX является системой разделения времени, и то, как часто программа будет получать системные ресурсы, решается на основании приоритетов процессов. Задания с более низким приоритетом будут получать доступ к системе реже, чем задания с более высоким приоритетом. Например, сеанс работы с терминалом имеет более высокий приоритет, чтобы гарантировать интерактивное взаимодействие с системой.

Команда **nice** – это еще одна префиксная команда, которая позволяет выполнять программу с пониженным приоритетом. Такой режим полезен при запуске команд, для которых не требуется немедленное выполнение, например, форматирование всех статей *Справочника*.

Синтаксис команды:

nice [*-приращение*] *командная_строка*

Здесь *приращение* – целое число в диапазоне от 1 до 19. Значение приращения по умолчанию – 10. Процесс с более высоким значением **nice** (поправки к приоритету) будет иметь в системе более низкий относительный приоритет. Значение **nice** не является абсолютным значением, модифицирующим приоритет.

Приоритеты процессов можно увидеть, если выполнить команду **ps -l**. Значения приоритетов выводятся в колонке, озаглавленной PRI. Более высокий приоритет заданий

обозначается *меньшим* значением приоритета. Значение **nice** выводится в колонке, озаглавленной NI.

В большинстве систем значение **nice** по умолчанию равно 20 для оперативных процессов и 24 для фоновых процессов. Максимальное значение равно 39, поэтому максимальное значение приращения равно 19 и 15, соответственно. Если будет указано большее значение приращения, значение **nice** все равно не станет больше 39. Только администратор системы (**root**) может указывать отрицательное приращение.

15–6. СЛАЙД: Команда **kill**

Команда **kill**

Синтаксис:

```
kill [-s имя_сигнала] PID [PID...]  Посылает сигнал  
указанным процессам
```

Примеры:

```
$ cat /usr/share/man/cat1/* > bigfile1 &  
[1] 995  
$ cat /usr/share/man/cat2/* > bigfile2 &  
[2] 996  
$ kill 995  
[1] - Terminated      cat /usr/share/man/cat1/* > bigfile1 &  
$ kill -s INT %2  
[2] + Interrupt        cat /usr/share/man/cat2/* > bigfile2 &  
$ kill -s KILL 0
```

Пояснения

Команда **kill** может быть использована для прекращения выполнения любой команды, включая **nohup** и фоновые команды. Более точно, команда **kill** посылает процессу сигнал. По умолчанию, реакцией процесса на большинство сигналов является завершение процесса. Пользователь, посылающий сигнал процессу, должен быть владельцем этого процесса; нельзя использовать команду **kill** для прекращения выполнения команд, принадлежащих другим пользователям, за исключением случая, когда команду **kill** выполняет суперпользователь.

В системе UNIX невозможно действительно уничтожить процесс. В действительности система UNIX посылает процессу требование, чтобы он завершился сам. По умолчанию команда **kill** посылает указанным процессам сигнал **TERM** (программный сигнал завершения). Обычно этот сигнал приводит к завершению процесса, если только процесс не перехватывает или не игнорирует этот сигнал. Другие сигналы, перечисленные ниже, должны задаваться с помощью опции **-s**. Наибольшую гарантию того, что выполнение процесса будет действительно завершено, дает сигнал **KILL**.

Чтобы завершить процесс, необходимо указать его идентификатор или номер задания. Если указывается номер задания, перед ним должен быть указан метасимвол **%**. Если в качестве идентификатора процесса указывается **0**, команда **kill** завершит все процессы, ассоциированные с текущим shell'ом, включая и сам текущий shell.

Имя сигнала	Значение сигнала
EXIT	Нулевой сигнал
HUP	Сигнал разрыва линии связи
INT	Прерывание
QUIT	Завершение
ILL	Неправильная инструкция (не переустанавливается в случае перехватывания сигнала)
TRAP	Трассировка прерывания (не переустанавливается в случае перехватывания сигнала)
ABRT	Сигнал аварийного завершения процесса
EMT	Инструкция EMT
FPE	Исключительная ситуация, связанная с выполнением операций с вещественными числами
KILL	Завершение процесса (не может быть перехвачен или проигнорирован)
BUS	Ошибка шины
SEGV	Нарушение сегментации
SYS	Неправильный аргумент в системном вызове
PIPE	Запись в канал, из которого не читает ни один процесс
ALRM	Сигнал будильника
TERM	Программный сигнал прерывания, передаваемый командой kill
USR1	Сигнал 1, определяемый пользователем
USR2	Сигнал 2, определяемый пользователем
CHLD	Завершение или приостановка порожденного процесса
PWR	Индикация состояния питания
VTALRM	Сигнал от виртуального таймера
PROF	Сигнал таймера профилирования
IO	Сигнал асинхронного ввода/вывода
WINCH	Сигнал изменения размера окна
STOP	Сигнал останова (не может быть перехвачен или проигнорирован)
TSTP	Сигнал интерактивного останова
CONT	Продолжить выполнение остановленного процесса
TTIN	Попытка чтения с управляющего терминала, предпринятая процессом – членом группы фоновых процессов
TTOU	Попытка записи на управляющий терминал, предпринятая процессом – членом группы фоновых процессов
URG	Срочное условие на канале I/O
LOST	Потеря контроля над удаленным терминалом (NFS)

ПРИМЕЧАНИЕ: Команда **kill -l** выводит на экран все значения *имени_сигнала*, поддерживаемые соответствующей реализацией системы UNIX. При указании этой опции никакие сигналы не посылаются. Если указывается опция **-l**, символические имена всех сигналов записываются в стандартный вывод:

```
$ kill -l
HUP INT QUIT ILL TRAP ABRT EMT FPE KILL BUS SEGV
SYS PIPE ALRM TERM USR1 USR2 CHLD PWR VTALRM PROF
IO WINCH STOP TSTP CONT TTIN TTOU URG LOST
```

Система Linux



Значения сигналов, используемых в системе Linux, несколько отличаются от приведенных выше, но главные сигналы со значениями 1 (HUP), 2 (INT), 3 (QUIT), 9 (KILL) и 15 (TERM) идентичны таким же сигналам, используемым в системе HP-UX.

15–7. УПРАЖНЕНИЯ: Управление процессами

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Найдите в своем *HOME* каталоге файл с именем **infinite**. Запустите эту программу на исполнение в оперативном режиме и посмотрите, что произойдет. Нажмите Ctrl + **c** для завершения выполнения программы.

2. Запустите программу **infinite** на выполнение в фоновом режиме и перенаправьте ее вывод в файл **infin.out**.

```
$ infinite > infin.out &
```

Выполните команду **ps -f**. Обратите внимание на значения идентификаторов PID и PPID процесса **infinite**. Теперь выйдите из системы, потом войдите снова и выполните **ps -ef | grep user_id**, где *user_id* – входное имя пользователя. Где процесс **infinite**? Удалите файл **infin.out**, перед тем как выполнять следующее упражнение.

3. Команда **nohup** защищает процесс от завершения, когда завершается сеанс работы в системе. Вновь запустите команду **infinite** на выполнение в фоновом режиме, но защитите ее от завершения с помощью команды **nohup**.

```
$ nohup infinite > infin.out &
```

Теперь выйдите из системы и затем опять войдите в нее. Снова выполните команду **ps -ef | grep user_id**. Процесс **infinite** еще выполняется? Какой процесс теперь является для него родительским?

4. Используйте команду **kill** для завершения процесса **infinite**.

5. Запустите программу **infinite** на выполнение в *оперативном* режиме и перенаправьте ее вывод в файл **infin.out**. Приостановите выполнение программы, нажав клавиши **Ctrl** + **z**. На экране вы увидите сообщение о том, что процесс приостановлен. Переведите процесс **infinite** в фоновый режим, обратите внимание на сообщение. Завершите выполнение программы **infinite** с помощью команды **kill**.

Модуль16 — Введение в shell программирование

Цели:

Изучив этот модуль, вы сможете:

- Создавать простые shell процедуры
- Передавать аргументы в shell процедуры, используя переменные среды
- Передавать аргументы в shell процедуры, используя позиционные параметры
- Использовать специальные переменные shell'a * и #
- Использовать команды **shift** и **read**

16–1. СЛАЙД: Shell программирование: введение

Shell программирование: введение

- Shell процедура – это обычный файл, содержащий команды системы UNIX
- Должен быть разрешен доступ к файлу, по крайней мере, на чтение и исполнение
- Чтобы выполнить процедуру, нужно в ответ на приглашение shell'a ввести имя файла
- Данные в shell-процедуре можно передать, используя
 - переменные среды
 - аргументы командной строки
 - ввод

Пояснения

Shell представляет собой командный интерпретатор. Он интерпретирует команды, вводимые в ответ на его приглашение. Всегда может возникнуть необходимость многократно вводить некоторую группу команд shell'a. Shell предоставляет возможность записать эти команды в файл и выполнять этот файл точно так же, как и любую другую программу, предоставляемую системой UNIX. Такой командный файл называют **shell программой** или **shell процедурой**. Когда такая программа запускается на исполнение, она выполняется точно так же, как если бы команды, включенные в эту программу, интерактивно вводились в ответ на приглашение shell.

Для того чтобы shell мог выполнять такую shell процедуру, он должен иметь возможность *читать* командный файл и *выполнять* каждую строку этого файла. Поэтому для shell процедуры должны быть установлены права на чтение и исполнение. Для того чтобы shell смог найти командный файл, необходимо ввести полное маршрутное имя файла или же поместить файл в одном из каталогов, перечисленных в переменной среды *PATH*. Многие пользователи создают в своем *HOME* каталоге подкаталог **bin**, в котором размещают свои shell процедуры, и добавляют в свою переменную среды *PATH* каталог **\$HOME/bin**.

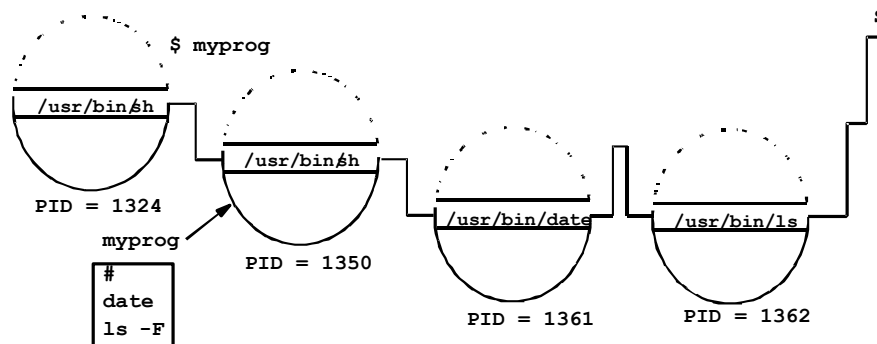
Можно разрабатывать достаточно сложные shell процедуры, так как shell позволяет работать с переменными и аргументами командной строки, использовать интерактивный ввод, реализовывать ветвление и циклы.

16–2. СЛАЙД: Пример shell процедуры

Пример shell процедуры

```
$ cat myprog
#this is the program myprog
date
ls -F
```

Выполнение **myprog**.



Пояснения

Рассмотрим следующий пример создания и выполнения shell процедуры:

```
$ vi myprog                                Файл, содержащий команды shell
# this is the program myprog
date
ls -F
$ chmod +x myprog                          Файлу добавляется право на исполнение
$ myprog                                    Для выполнения программы вводится имя файла
Thu Jul 11 11:10 EDT 1994

f1 f2 memo/ myprog*
```

Прежде всего, с помощью текстового редактора создается shell процедура **myprog**. Прежде чем выполнять программу, файлу процедуры должно быть добавлено право на исполнение. После этого в ответ на приглашение shell'a можно ввести имя файла. Когда выполняется файл **myprog**, создается порожденный процесс shell, что и показано на слайде. Порожденный shell читает командные строки не из стандартного ввода, а из программного файла **myprog**. Соответственно, порожденный shell выполняет каждую команду из shell процедуры. Когда все команды будут выполнены, порожденный shell завершится и вернет управление исходному родительскому shell'у.

Комментарии в shell процедуре

Рекомендуется включать в shell процедуру комментарии, которые определяют и поясняют программу. Комментарий начинается символом **#** и размещается в одной строке. Shell не пытается выполнять текст, размещенный после символа **#**; этот символ может появиться в любом месте в командной строке.

ПРИМЕЧАНИЕ: Никогда не следует называть shell процедуру именем **test**, так как **test** – это встроенная команда shell'а.

16–3. СЛАЙД: Передача данных в shell процедуры

Передача данных в shell процедуры

```
$ color=lavender

$ cat color1
echo You are now running program: color1
echo the value of the variable color is: $color

$ chmod +x color1

$ color1
You are now running program: color1
the value of the variable color is:

$ export color
$ color1
You are now running program: color1
the value of the variable color is: lavender
```

Пояснения

Один из способов передачи данных в shell процедуры – использование среды процесса. В примере, приведенном на слайде, переменной *color* присваивается значение *lavender*. Затем создается shell процедура **color1**; изменяются ее права доступа – добавляется право на исполнение; затем процедура выполняется. Процедура **color1** пытается вывести на экран значение переменной *color*. Но так как *color* – локальная переменная, принадлежащая родительскому shell'у, порожденный shell, выполняющий процедуру **color1**, не имеет доступа к этой переменной и поэтому не может вывести на экран ее значение. Если же переменную *color* перед выполнением процедуры экспортировать в среду, тогда она будет доступна командам, выполняемым порожденным shell'ом.

Кроме того, поскольку порожденный процесс не может изменить среду родительского процесса, изменение значения переменной среды в порожденном процессе не изменит значение переменной среды родительского процесса. Рассмотрим shell процедуру **color2** (ее можно найти в *HOME* каталоге):

```
echo The original value of the variable color is $color
echo This program will set the value of color to amber
color=amber
echo The value of color is now $color
echo When your program concludes, display the value of the color
variable.
```

Рассмотрим, что произойдет, если установить значение переменной *color*, экспортировать ее, а затем выполнить shell процедуру **color2**:

```
$ export color=lavender
$ echo $color
lavender
$ color2
The original value of the variable color is lavender
This program will set the value of color to amber
The value of color is now amber
When your program concludes, display the value of the color variable.
$ echo $color
lavender
```


16–4. СЛАЙД: Аргументы shell процедуры

Аргументы shell процедуры

Командная строка:

```
$ shell_процедура arg1 arg2 . . . argX
    $0          $1  $2  . . .  $X
```

Пример:

```
$ cat color3
echo You are now running program: $0
echo The value of command line argument \#1 is: $1
echo The value of command line argument \#2 is: $2

$ chmod +x color3

$ color3 red green
You are now running program: color3
The value of command line argument #1 is: red
The value of command line argument #2 is: green
```

Пояснения

Большинство команд системы UNIX допускают задание в командной строке аргументов, передающих команде имена файлов или каталогов, которые команда должна обрабатывать (например, **cp f1 f2**), опции, которые уточняют особенности выполнения команды (например, **ls -l**), или просто текстовые строки (например, **banner hi there**).

Для shell процедур также допустимо задание в командной строке аргументов. Аргументы предоставляют удобный способ передачи информации в программу. Если разрабатываемая процедура допускает задание в командной строке аргументов, в процедуру можно будет точно так же, как и при использовании команд системы UNIX, передать имена файлов или каталогов, которые нужно обрабатывать. Можно также указать в командной строке опции, с помощью которых можно из командной строки получить доступ к дополнительным возможностям shell процедуры.

Аргументы shell процедуры, передаваемые из командной строки, получили название **позиционных параметров**, так как значение каждого параметра зависит от положения (позиции) аргумента в командной строке. К аргументам командной строки можно обращаться с помощью специальных переменных, имена которых соответствуют номеру их позиции в командной строке; таким образом, именами специальных переменных являются числа **0,1,2** и так далее, пока не будет получен последний переданный параметр. Значения специальных переменных можно получить точно так же, как и значения обычных переменных shell'a – указав перед именем переменной символ **\$**. Следовательно, чтобы из shell процедуры получить значения аргументов командной строки, нужно записать **\$0, \$1,**

\$2 и так далее. Чтобы получить значения аргументов после **\$9**, имена переменных необходимо заключить в фигурные скобки: **\${10}**, **\${24}** и так далее; в противном случае shell будет трактовать запись **\$10** как **\$1**, к значению которого в его конец добавляется символ **0** (нуль). **\$0** *всегда* соответствует имени процедуры или команды.

Естественным недостатком разрабатываемых процедур, принимающих из командной строки аргументы, является то, что пользователь должен знать синтаксис и семантику (что они собой представляют) аргументов командной строки. Например, откуда вы знаете, что команда **cp** может копировать один файл в другой или несколько файлов в каталог? Что произойдет, если вы попытаетесь выполнить команду, передав ей в качестве аргументов имена трех файлов: **cp f1 f2 f3**? В вашем распоряжении есть *Справочное руководство* системы UNIX, в котором представлено необходимое описание, и, кроме того, система UNIX выводит сообщение о правилах записи команды, если выполняемая команда будет написана неправильно (попробуйте ввести **cp** Return). Рекомендуется при написании собственных программ выводить подобные сообщения о правилах вызова программы, помогающие другим пользователям, которые будут их использовать.

16–5. СЛАЙД: Аргументы shell процедуры (продолжение)

Аргументы shell процедуры (продолжение)

Данная shell процедура устанавливает программу, заданную аргументом командной строки, в каталог **bin** пользователя. Сначала создается процедура **my_install**.

```
$ cat > my_install
echo $0 will install $1 to your bin directory
chmod +x $1
mv $1 $HOME/bin
echo Installation of $1 is complete
Ctrl + d
$ chmod +x my_install

$ my_install color3
my_install will install color3 to your bin directory
Installation of color3 is complete
$
```

Пояснения

В приведенном примере рассматривается программа, для которой первый аргумент командной строки должен задавать имя файла, которому надо добавить право на исполнение и затем переместить его в каталог **bin**, находящийся в текущем каталоге.

Перед тем как исполнять данную процедуру, проверьте существование каталога **\$HOME/bin**! В соответствии с соглашениями системы UNIX, исполняемые программы хранятся в каталоге с именем **bin**. Можно в собственном *HOME* каталоге создать подкаталог **bin**, в котором и размещать shell процедуры. При этом необходимо добавить имя созданного каталога **bin** к переменной *PATH*, чтобы shell смог найти эти процедуры.

16–6. СЛАЙД: Специальные переменные shell'a — # и *

Специальные переменные shell'a — # и *

- # Количество аргументов командной строки
- * Строка, содержащая все аргументы командной строки

Пример:

```
$ cat color4
echo There are $# command line arguments
echo They are $*
echo The first command line argument is $1

$ chmod +x color4

$ color4 red green yellow blue
There are 4 command line arguments
They are red green yellow blue
The first command line argument is red
$
```

Пояснения

Рассмотренные ранее shell процедуры являются не очень гибкими. Так, процедура **color3** требует точно два аргумента командной строки, а процедура **my_install** – только один аргумент. Часто при создании shell процедуры, которая допускает задание аргументов командной строки, желательно разрешить пользователю при вызове этой процедуры передавать ей произвольное количество аргументов. Например, хотелось бы, чтобы процедура выполнялась успешно независимо от того, сколько аргументов – 1 или 20 – передать ей в командной строке.

Дополнительную гибкость shell процедурам при использовании списка аргументов командной строки придают специальные переменные shell'a # и *. Используя **\$#**, можно всегда узнать, сколько аргументов указано в командной строке, а с помощью **\$*** можно всегда получить весь список аргументов командной строки независимо от их количества. Следует обратить внимание на то, что имя команды (**\$0**) никогда не включается в список аргументов, получаемый с помощью **\$***.

С каждым аргументом командной строки по-прежнему можно работать индивидуально. Можно использовать весь список аргументов, указав **\$***, или же использовать конкретные аргументы из списка, указав **\$1**, **\$2**, **\$3** и так далее.

16–7. СЛАЙД: Специальные переменные shell'a — # и * (продолжение)

Специальные переменные shell'a — # и * (продолжение)

Усовершенствованный пример программы установки допускает задание в командной строке нескольких аргументов

```
$ cat > my_install12
echo $0 will install $# files to your bin directory
echo The files to be installed are: $*
chmod +x $*
mv $* $HOME/bin
echo Installation is complete
Ctrl + d
$ chmod +x my_install12

$ my_install12 color1 color2
my_install12 will install 2 files to your bin directory
The files to be installed are: color1 color2
Installation is complete
```

Пояснения

Теперь программа установки стала более гибкой. Если требуется установить несколько shell процедур, нужно выполнить эту программу только один раз, перечислив в командной строке имена всех процедур. Важно помнить, что если предполагается передавать командам весь список, эти команды должны допускать задание нескольких аргументов.

Рассмотрим следующую процедуру, которой пользователь с помощью аргумента командной строки передает имя каталога. Данная процедура выполняет переход в указанный каталог, выводит имя текущего каталога и его содержимое:

```
$ cat list_dir
cd $*
echo You are in the $(pwd) directory
echo The contents of this directory are:
ls -F
$ list_dir dir1 dir2 dir3
sh: cd: bad argument count
```

Так как в команде **cd** можно указать только один аргумент, эта программа выводит сообщение об ошибке.

16–8. СЛАЙД: Команда **shift**

Команда **shift**

Синтаксис:

```
shift [n]
```

- Сдвигает все строки (аргументы) в ***** влево на **n** позиций аргументов
- Уменьшает значение **#** на **n**
- Если **n** не указано, оно принимается равным 1

Пример:

```
$ cat color5
orig_args=$*
echo There are $# command line arguments
echo They are $*
echo Shifting two arguments
shift 2
echo There are $# command line arguments
echo They are $*
echo Shifting two arguments
shift 2; final_args=$*
echo Original arguments are: $orig_args
echo Final arguments are: $final_args
```

Пояснения

Команда **shift** выполняет переназначение аргументов командной строки позиционным параметрам, позволяя тем самым перемещаться по аргументам командной строки. После выполнения команды **shift n** все параметры в списке ***** перемещаются влево на **n** позиций аргументов, и значение переменной **#** уменьшается на **n**. Если **n** не указано, оно предполагается равным 1. Команда **shift** не влияет на позиционный параметр **0**.

После выполнения сдвига аргументы, которые были выдвинуты за пределы командной строки, теряются. Если эти переменные необходимо использовать после сдвига, *перед* выполнением команды **shift** их необходимо запомнить.

Команду **shift** полезно использовать для:

- доступа к позиционным параметрам в группах, например, последовательности координат **x** и **y**
- удаления опций из командной строки, в предположении, что опции предшествуют аргументам

Пример

Иже показаны результаты выполнения shell процедуры, текст которой приведен на слайде:

```
$ color5 red green yellow blue orange black
```

```
There are 6 command line arguments
```

```
They are red green yellow blue orange black
```

```
Shifting two arguments
```

```
There are 4 command line arguments
```

```
They are yellow blue orange black
```

```
Shifting two arguments
```

```
Original arguments are: red green yellow blue orange black
```

```
Final arguments are: orange black
```

```
$
```

16–9. СЛАЙД: Команда **read**

Команда **read**

Синтаксис:

```
read переменная [ переменная ... ]
```

Пример:

```
$ cat color6
echo This program prompts for user input
echo "Please enter your favorite two colors -> \c"
read color_a color_b
echo The colors you entered are: $color_b $color_a
$ chmod +x color6
$ color6
This program prompts for user input
Please enter your favorite two colors -> red blue
The colors you entered are: blue red
$ color6
This program prompts for user input
Please enter your favorite two colors -> red blue tan
The colors you entered are: blue tan red
```

Пояснения

Аргументы командной строки позволяют пользователю передавать данные в программу в момент ее вызова, и пользователь должен знать правила передачи аргументов *перед* использованием команды. Возможны ситуации, когда нужно просто запустить на исполнение программу без аргументов, а затем уже сама программа *во время* своего исполнения предложит пользователю ввести необходимые данные. Команда **read** используется для приема данных, вводимых с клавиатуры терминала во время выполнения программы.

Обычно пользователю с помощью команды **echo** выводится приглашение, которое сообщает ему, что программа ожидает ввода данных и какие именно данные нужно ввести. Следовательно, каждой команде **read** должна предшествовать команда **echo**.

В команде **read** задается список имен переменных, которым в качестве их значений будут присваиваться слова из строки, введенной пользователем в ответ на приглашение (слова в строке разделяются символами пробела и/или знака табуляции). Если в команде **read** указано переменных больше, чем вводится слов в строке, последним оставшимся переменным будет присвоено пустое (NULL) значение. Если пользователь ввел слов больше, чем указано переменных в команде, последней переменной будет присвоена строка, содержащая все оставшиеся слова.

После ввода значения этих переменных можно получить точно так же, как и значения других переменных shell'a.

ПРИМЕЧАНИЕ: Не следует путать позиционные параметры с переменными, чьи значения вводятся с помощью команды **read**. Позиционные параметры указываются в командной строке в момент вызова программы на исполнение. Команда **read** присваивает переменным значения, вводимые с клавиатуры в ответ на приглашение программы во время ее выполнения.

Команда **echo** и управляющие (escape) символы

Есть ряд специальных escape символов, которые команда **echo** интерпретирует как символы управления строкой (управляющие символы). Каждому управляющему символу должен предшествовать символ «обратный слеш» (\), и управляющие символы должны быть заключены в двойные кавычки ("). Эти управляющие символы интерпретируются командой **echo**, но не shell'ом.

Символ	Действие при выводе
\a	Символ «звонок» (эквивалентен Ctrl + g)
\b	Забой (возврат на один символ)
\c	Подавление перехода на новую строку
\f	Перевод страницы
\n	Перевод строки
\r	Возврат каретки
\t	Символ табуляции
\\	Обратный слеш
\nnn	Символ, ASCII-код которого равен <i>nnn</i> , где <i>nnn</i> – от одной до трех восьмеричных цифр, начинающихся с нуля

16–10. СЛАЙД: Команда **read** (продолжение)

Команда **read** (продолжение)

Усовершенствованный пример программы установки выводит пользователю приглашение ввести имена файлов, которые требуется установить:

```
$ cat > my_install3
echo $0 will install files into your bin directory
echo "Enter the names of the files -> \c"
read filenames
chmod +x $filenames
mv $filenames $HOME/bin
echo Installation is complete
Ctrl + d
$ chmod +x my_install3

$ my_install3
my_install3 will install files into your bin directory
Enter the names of the files -> f1 f2
Installation is complete
```

Пояснения

Данная версия программы установки выводит пользователю приглашение ввести имена файлов, которые после ввода указываются в качестве аргументов команды **chmod** и затем перемещаются в каталог **\$HOME/bin**. По сравнению с программой **install2**, в которой пользователь должен указывать имена файлов в командной строке, эта программа предоставляет пользователю несколько больше указаний относительно ожидаемого ввода. Теперь пользователь не должен знать специальные правила вызова программы. Эта программа позволяет пользователю точно знать, что его ожидает. Все имена файлов при вводе присваиваются переменной *filenames*.

16–11. СЛАЙД: Дополнительные средства

Дополнительные средства

- Документирование shell процедур включением комментариев, начинающихся знаком номера (#)
- ***sh shell_процедура аргументы***
shell_процедура не должна обязательно иметь установленное право исполнения
shell_процедура должна обязательно иметь установленное право чтения
- ***sh -x shell_процедура аргументы***
Каждая строка ***shell_процедуры*** перед исполнением выводится на экран
Полезно использовать для отладки процедуры

Пояснения

Для включения в shell процедуру комментариев используется символ #. Shell игнорирует все, что размещается после символа # и до нажатия клавиши **[Return]**. Комментарии предоставляют другим пользователям (а, возможно, и самому разработчику процедуры) информацию о целях включения в процедуру различных команд.

Shell процедуру можно выполнить и другим способом:

```
sh shell_процедура аргументы
```

Такой способ выполнения создает порожденный процесс – shell и определяет его как командный интерпретатор, который будет работать во время выполнения процедуры. В этом случае программный файл *не должен* обязательно иметь установленное право на исполнение. Такой способ полезен, когда пользователь работает в одном shell'е и хочет выполнить shell процедуру, написанную на языке другого shell'а.

С помощью специального комментария **#!/usr/bin/имя_shell**, указанного в первой строке shell процедуры, можно также указать, какой именно интерпретатор команд должен выполнять shell процедуру. Поэтому, если вы во время сеанса работы в системе используете POSIX shell, а вам надо выполнить shell процедуру, написанную на языке C shell, в первой строке этой процедуры должно быть записано: **#!/usr/bin/csh**.

Хотя в системе нет специальных отладчиков для отладки shell процедур, команда

```
sh -x shell_процедура аргументы
```

будет выводить на экран каждую команду из shell процедуры *перед* ее исполнением. Это позволяет увидеть, как shell выполняет генерацию имен файлов, подстановку значения переменной и подстановку результатов выполнения команды. Эту опцию особенно полезно использовать для нахождения опечаток.

16–12. УПРАЖНЕНИЯ: Введение в shell программирование

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Напишите shell процедуру **whoson1**, которая:
 - с помощью команды **echo** выведет на экран приветствие,
 - определит переменную *MYNAME* и присвоит ей значение вашего имени,
 - выведет на экран значение переменной *MYNAME*,
 - выведет на экран текущие дату и время,
 - выведет информацию о том, какие пользователи работают в данное время в системе.
2. Перейдите в каталог **/tmp**. Вызовите на исполнение программу **color1**. Найдет shell программу **color1**?
3. Вернитесь в *HOME* каталог. Создайте в нем каталог **bin**. Переместите программу **color1** в каталог **bin**. Добавьте к переменной среды *PATH* ваш каталог **bin**, чтобы shell смог найти программу **color1**. Проверьте, что теперь переменная *PATH* обеспечит правильную работу, перейдя в каталог **/tmp** и вызвав на исполнение программу **color1**. Перед вызовом программы **color1** не забудьте определить переменную *color*.
4. Вернитесь в *HOME* каталог. Присвойте в интерактивном режиме переменной *date_var* результаты работы команды **date**. Напишите shell процедуру **date_tst**, которая выводит значение этой переменной. Установите процедуру **date_tst** в ваш каталог **bin**.

5. Модифицируйте процедуру **date_tst** так, чтобы значение переменной *date_var* присваивалось во время ее выполнения. Необходимо ли теперь экспортировать в среду переменную *date_var*? Нужно ли изменить права доступа к файлу **date_tst**?

6. Напишите shell процедуру **whoson2**, которая:

- с помощью команды **echo** выводит пользователю персональное приветствие в виде *welcome имя_пользователя*, так что если пользователь войдет в систему под именем **stud3**, он увидит приветствие *welcome stud3*; если же пользователь войдет в систему под именем **stud2**, он увидит приветствие *welcome stud2*. (ПОДСКАЗКА: имя пользователя можно получить, используя переменную среды или подстановку результата выполнения команды),
- выводит системные дату и время,
- выводит информацию о том, какие пользователи работают в данное время в системе,
- выводит сообщение пользователю, содержащее идентификацию его сеанса связи с системой,
- выводит сообщение перед завершением процедуры.

Переместите процедуру в каталог **bin**, чтобы вы могли выполнять ее независимо от того, в какой точке файловой системы вы находитесь.

7. Если shell процедура вызывается на исполнение следующим образом:

```
$ myshellprog abc def -d -4 +900 xyz
```

что будет выведено на экран, если в эту shell процедуру включены следующие команды:

```
echo $#  
echo $3  
echo $7  
echo $*  
echo $0
```

8. Пусть shell процедура, вызванная на исполнение так, как указано в предыдущем упражнении, в первой своей строке содержит команду **shift 2**; что тогда будет выведено на экран следующими командами:

```
echo $#  
echo $3  
echo $7  
echo $*  
echo $0
```

9. Каким будет результат работы приведенной ниже shell процедуры, если в ответ на приглашение пользователь введет такую строку:

James A. Smith, Jr.

Shell процедура:

```
echo "Please type in your first, middle, and last names"  
read first middle last  
echo "$last, $first $middle"
```

10. Напишите shell процедуру **search1**, предлагающую пользователю ввести строку, которую следует искать в файлах, находящихся в текущем каталоге. На экран следует выводить имена файлов, в которых встречается введенная строка.

11. Напишите shell процедуру **backwards**, которая принимает не более 10 аргументов и выводит аргументы в обратном порядке.

12. Напишите shell процедуру **myecho**, которая:

- выводит на экран количество аргументов, переданных ей из командной строки,
- выводит на экран первые три аргумента, каждый в отдельной строке,
- выводит все остальные аргументы в одной строке.

Выполните процедуру, передав ей 12 аргументов.

Каким должен быть список аргументов этой процедуры, чтобы в результате ее выполнения получилось следующее:

```
I cannot  
seem to  
find my KEYS.
```

13. Напишите shell процедуру **my_vi**, аргумент командной строки которой определяет имя файла, предназначенного для редактирования. Процедура **my_vi** должна сделать резервную копию указанного файла, а затем начать сеанс редактирования этого файла с помощью редактора **vi**. При создании резервной копии файла используйте расширение имени файла **.bak**. Используйте имена файлов длиной не более 10 символов.

14. Создайте shell процедуру **my_recover**, являющуюся дополнением к процедуре **my_vi**; новая процедура должна восстанавливать из резервной копии файл, заданный аргументом командной строки. В командной строке следует указывать имя файла без расширения **.bak**. Например, чтобы восстановить файл **tst1** из резервной копии **tst1.bak**, нужно выполнить команду **my_recover tst1**.

15. Напишите shell процедуру **info**, которая выводит пользователю приглашение ввести:

- имя,
- название улицы (часть адреса),
- город, государство, zip-код (почтовый индекс).

Процедура должна запомнить введенные данные в переменных и вывести их в информативном виде, с необходимыми комментариями.

16. Напишите shell процедуру **home**, которая предлагает пользователю ввести *входное имя* любого пользователя, зарегистрированного в системе, и выводит имя *HOME* каталога этого пользователя. Помните, что имя *HOME* каталога пользователя указано в шестом поле файла паролей **/etc/passwd**. Процедура должна вывести *входные имена* из файла **/etc/passwd** в четыре столбца, так чтобы пользователь знал существующие в системе входные имена пользователей.

17. Напишите shell процедуру **alpha**, которая выводит на экран первый и последний аргументы командной строки. ПОДСКАЗКА: используйте команду **cut**.

18. Напишите shell процедуру **copy**, которая предоставляет пользователю удобный интерфейс для копирования файлов с помощью команды **cp**. Процедура должна вывести пользователю приглашение ввести имена файлов, которые нужно копировать, а затем приглашение ввести имя приемника копируемых файлов. Если копируется несколько файлов, в качестве приемника необходимо указать имя каталога; если же копируется один файл, в качестве приемника может быть указано имя файла. По окончании копирования процедура должна выдать звуковой сигнал.

Модуль 17 — Shell программирование: ветвление

Цели:

Изучив этот модуль, вы сможете:

- Описывать использование кодов завершения для ветвления по условию
- Использовать команду **test** для анализа кода завершения
- Использовать инструкции **if** и **case** для организации ветвления в shell процедурах

17–1. СЛАЙД: Коды завершения

Коды завершения

Переменная `shell`'а `?` хранит код завершения последней выполнявшейся команды

`0` команда выполнена без ошибок (истина)
не 0 команда завершена в связи с ошибкой (ложь)

Примеры:

```
$ true
$ echo $?
0
$ ls
$ echo $?
0
$ false
$ echo $?
1
$ cp
Usage: cp f1 f2
       cp [-r] f1 ... fn d1
$ echo $?
1
$ echo $?
0
```

Пояснения

Все команды операционной системы UNIX по окончании своей работы возвращают код завершения. Обычно этот код завершения используется, чтобы определить, как завершилась команда – успешно (возвращается `0`) или в ходе ее исполнения были обнаружены какие-либо ошибки (возвращается ненулевое значение). Код завершения, отличный от нуля, часто указывает на характер ошибки. Например, синтаксические ошибки обычно устанавливают код завершения, равный `1`. Команда **true** всегда возвращает `0`, а команда **false** всегда возвращает `1`.

В большинстве случаев в программах решения принимаются на основе анализа значения кода завершения. В `shell`'е есть специальная переменная `?`, которая хранит значение кода завершения последней выполнявшейся команды.

Всегда можно увидеть значение кода завершения *предыдущей* команды, выполнив

```
echo $?
```

Когда выполняется ветвление по условию (т.е. меньше чем, больше чем, равно), код завершения устанавливается в зависимости от того, истинно (возвращается `0`) или ложно (возвращается не нуль) указанное условие. Проверка условий будет представлена на нескольких следующих слайдах.

17–2. СЛАЙД: Команда `test`

Команда `test`

Синтаксис:

`test` *выражение* или [*выражение*]

Команда `test` оценивает истинность выражения и в зависимости от результата устанавливает код завершения

Значение выражения	Код завершения
истина	0
ложь	не 0 (обычно 1)

Команда `test` может оценивать истинность выражений, в которых в качестве операндов могут быть указаны

- целые
- строки
- файлы

Пояснения

Команда `test` используется для вычисления выражений и формирования кода завершения. С помощью аргументов в команде записываются логические выражения, которые и вычисляются этой командой. Команда `test` ничего не пишет в стандартный вывод. Чтобы увидеть результат выполнения команды `test`, нужно вывести на экран значение кода завершения. Код завершения будет установлен в 0, если результатом вычисления выражения будет значение *истина*, или в 1, если результатом вычисления выражения будет *ложь*.

Сначала команда `test` представляется независимо от других команд, так что можно увидеть ее код завершения. Однако наиболее часто эта команда используется в инструкциях `if` и `while` для организации управления порядком выполнения команд.

Команда `test` может также использоваться в виде [*выражение*]. Такой способ ее использования улучшает читабельность программ, особенно когда команда используется для анализа числовых или строковых данных.

ПРИМЕЧАНИЕ: Вокруг скобок [и] обязательно должны быть пробелы.

17-3. СЛАЙД: Команда **test**: сравнение чисел

Команда **test**: сравнение чисел

Синтаксис:

[**число отношение число**] Сравнивает числа в соответствии с указанным отношением

Отношения:

-lt меньше
-le меньше или равно
-gt больше
-ge больше или равно
-eq равно
-ne не равно

Примеры (предполагается, что X=3):

\$ ["\$X" -lt 7]	\$ ["\$X" -gt 7]
\$ echo \$?	\$ echo \$?
0	1

Пояснения

Команда **test** позволяет оценивать истинность отношения между двумя целыми числами. В этом случае наиболее часто команда используется в виде [...]. Код завершения команды **test** позволяет определить, истинно отношение (возвращается 0) или ложно (возвращается 1).

В команде используются следующие операции отношения между числами:

-lt	меньше
-le	меньше или равно
-gt	больше
-ge	больше или равно
-eq	равно
-ne	не равно

Когда анализируется значение переменной shell'a, необходимо учитывать ситуацию, когда переменная имеет пустое значение. Рассмотрим следующий пример:

```
$ [ $XX -eq 3 ]  
sh: test: argument expected
```

Если переменной *XX* не было ранее присвоено значение, тогда ее значением будет NULL. После того как shell выполнит подстановку переменной, команда, которую shell попытается выполнить, будет иметь вид:

```
[ -eq 3 ]
```

В результате подстановки синтаксически правильная команда **test** не получена, что и привело к появлению сообщения о синтаксической ошибке. Чтобы исключить такую ситуацию, проще всего заключить подстановку переменной в кавычки:

```
[ "$XX" -eq 3 ]
```

Теперь, после того как shell выполнит подстановку переменной, команда, которую shell попытается выполнить, будет иметь вид:

```
[ "" -eq 3 ]
```

В результате гарантируется, что переменная имеет, по крайней мере, *пустое* (NULL) значение и предоставит команде **test** приемлемый аргумент.

ПРИМЕЧАНИЕ: Как правило, выражение **\$переменная** заключается в двойные кавычки, что гарантирует, что shell выполнит требуемую подстановку переменной.

17–4. СЛАЙД: Команда **test**: сравнение строк

Команда **test**: сравнение строк

Синтаксис:

<code>[строка1 = строка2]</code>	Определяет, совпадают ли строки
<code>[строка1 != строка2]</code>	Определяет, отличаются ли строки

Примеры:

<code>\$ X=abc</code>	<code>\$ X=abc</code>
<code>\$ ["\$X" = "abc"]</code>	<code>\$ ["\$X" != "abc"]</code>
<code>\$ echo \$?</code>	<code>\$ echo \$?</code>
0	1

Пояснения

Команду **test** можно также использовать для сравнения (равны или не равны) двух строк. Наиболее часто для сравнения строк используется следующий вид команды: `[ ... ]`. Опять же следует помнить, что вокруг скобок `[` и `]` должны быть пробелы; кроме того, пробелы должны окружать и операции сравнения.

Команда допускает следующие операции сравнения строк:

<code>строка1 = строка2</code>	истина, если строка1 и строка2 идентичны
<code>строка1 != строка2</code>	истина, если строка1 и строка2 отличаются друг от друга
<code>-z строка</code>	истина, если длина строки равна нулю
<code>-n строка</code>	истина, если длина строки не равна нулю
<code>строка</code>	истина, если длина строки не равна нулю

Двойные кавычки здесь также защищают от ошибок, если в значении переменной имеются пробелы. Например:

<code>\$ X="Yes we will"</code>	
<code>\$ [\$X = yes]</code>	<i>неправильный синтаксис – вызывает сообщение об ошибке</i>

Интерпретируется shell'ом так: [**Yes we will = yes**]

```
$ [ "$X" = yes ]
```

корректный синтаксис

Интерпретируется shell'ом так: [**"Yes we will" = yes**]. Такая команда будет выполняться корректно, так как строка заключена в кавычки.

Сравнение чисел и сравнение строк

Shell трактует все аргументы как числа, если выполняется сравнение чисел, и как строки, если выполняется сравнение строк. Это хорошо иллюстрируется следующим примером:

```
$ X=03
$ Y=3
$ [ "$X" -eq "$Y" ]
```

сравнение чисел 03 и 3

```
$ echo $?
```



```
0
```

истина – числа имеют одинаковые значения

```
$ [ "$X" = "$Y" ]
```

сравнение строки "03" со строкой "3"

```
$ echo $?
```



```
1
```

ложь – эти две строки не идентичны

17–5. СЛАЙД: Команда **test**: анализ файлов

Команда **test**: анализ файлов

Синтаксис:

test -опция имя_файла

Проверяет, соответствуют ли
характеристики файла **имя_файла**
указанной опции

Примеры:

```
$ test -f funfile
$ echo $?
0
$ test -d funfile
$ echo $?
1
```

Пояснения

Полезным свойством shell'а является его способность проверять характеристики файла, такие как тип файла и права доступа. Например:

```
test -f имя_файла
```

вернет истину (0), если файл существует и является обычным (не каталог или файл устройства);

```
test -s имя_файла
```

вернет истину (0), если файл существует и его размер больше 0.

Существуют и другие возможности анализа файлов. Некоторые из них приведены ниже:

- | | |
|-------------------|---|
| -r файл | истина, если файл существует и доступен для чтения |
| -w файл | истина, если файл существует и доступен для записи |
| -x файл | истина, если файл существует и доступен для исполнения |
| -d каталог | истина, если каталог существует и действительно является каталогом |

Примеры на слайде могут быть записаны и так:

```
$ [ -f funfile ]
```

```
$ [ -d funfile ]
```

Описание дополнительных возможностей анализа файлов можно найти в *Справочном руководстве HP-UX*.

17–6. СЛАЙД: Команда **test**: другие операции

Команда **test**: другие операции

Синтаксис:

-o OR (ИЛИ)
-a AND (И)
! NOT (НЕ)
\ (\) ГРУППИРОВАНИЕ*

Примеры:

```
$ [ "$ANS" = y -o "$ANS" = Y ]  
$ [ "$NUM" -gt 10 -a "$NUM" -lt 20 ]  
$ test -s file -a -r file
```

* ПРИМЕЧАНИЕ: круглые скобки () должны быть экранированы с помощью символа «обратный слеш»

Пояснения

Для записи сложных условий можно использовать булевские операции.

Выражение 1	Операция	Выражение 2	Результат
истина	-o	истина	истина (0)
истина	-o	ложь	истина (0)
ложь	-o	истина	истина (0)
ложь	-o	ложь	ложь (1)
истина	-a	истина	истина (0)
истина	-a	ложь	ложь (1)
ложь	-a	истина	ложь (1)
ложь	-a	ложь	ложь (1)

Примеры

```
$ [ "$ANS" = y -o "$ANS" = Y ]  
$ [ "$NUM" -gt 10 -a "$NUM" -lt 20 ]  
$ test -s file -a -r file -a -x file
```

Операция NOT (!) используется вместе с другими операциями и наиболее часто применяется для анализа файлов. Обязательно *должен* быть пробел между операцией отрицания и любыми другими операциями и аргументами. Например, команду

```
test ! -d file
```

можно использовать вместо команды

```
test -f file -o -c file -o -b file ...
```

Для группирования операций в выражении можно использовать круглые скобки. Здесь следует иметь в виду, что круглые скобки входят в состав специальных символов shell'a, и в первую очередь они будут интерпретироваться shell'ом. Поэтому круглые скобки, если они используются в выражении, должны быть экранированы.

В следующем примере выполняются следующие проверки: указаны два аргумента командной строки, *И* первый аргумент имеет вид **-m**, *И* второй аргумент задает *каталог ИЛИ файл*, размер которого больше нуля:

```
[ \( $# = 2 \) -a \( "$1" = "-m" \) -a \( -d "$2" -o -s "$2" \) ]
```

17–7. СЛАЙД: Команда **exit**

Команда **exit**

Синтаксис:

```
exit [число]
```

Примеры:

```
$ cat exit_test  
echo exiting program now  
exit 99
```

```
$ exit_test  
exiting program now
```

```
$ echo $?  
99
```

Пояснения

Команда **exit** завершает выполнение shell процедуры и устанавливает код завершения. Обычно при нормальном завершении процедуры код завершения устанавливается равным нулю, а при обнаружении каких-либо ошибок – некоторому не нулевому значению. Если в команде не указан аргумент, код завершения процедуры устанавливается равным коду завершения команды, выполнявшейся непосредственно перед командой **exit**.

17–8. СЛАЙД: Конструкция `if`

Конструкция `if`

Синтаксис: (используется для выбора только из одной ветви)

```
if
    список_A
then
    список_B
fi
```

Пример:

```
if
    test -s funfile
then
    echo funfile exists
fi
echo hello
```

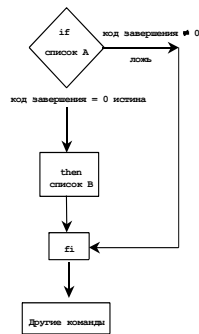
Пояснения

Конструкция `if` осуществляет управление порядком выполнения команд на основе *кода завершения* команды. Если код завершения управляющей команды равен нулю, будет выполнена определенная совокупность команд. Если же код завершения управляющей команды не равен нулю, эта совокупность команд будет пропущена.

На слайде приведен общий формат конструкции `if` и простой пример. Каждый список команд в общем случае состоит из одной или нескольких команд системы UNIX, разделенных символами `Return` или точками с запятой. Решение в конструкции `if` принимается на основе кода завершения *последней* команды в *списке A*, выполняющейся непосредственно перед `then`.

Ниже приводится алгоритм выполнения конструкции `if`:

1. Выполняется *список A*.
2. Если код завершения последней команды из *списка A* равен **0** (ИСТИНА), выполняются команды из *списка B*, после чего обработка продолжается с первого предложения, записанного после `fi`.
3. Если код завершения последней команды из *списка A* не равен **0** (ЛОЖЬ), управление передается на `fi` и обработка продолжается с первого предложения, записанного после `fi`.

Блок-схема конструкции **if**

Для управления порядком выполнения команд часто используется команда **test**, но вместо нее может использоваться *любая* команда, так как все команды системы UNIX генерируют код завершения, что и показано в следующем примере:

```

if
    grep kingkong /etc/passwd > /dev/null
then
    echo found kingkong
fi
  
```

Конструкция **if** используется также для управления порядком выполнения команд в тех ситуациях, когда возможны ошибки, что и показано в следующем примере:

```

if
    [ $# -ne 3 ]
then
    echo Incorrect syntax
    echo Usage: cmd arg1 arg2 arg3
    exit 99
fi
  
```


17–9. СЛАЙД: Конструкция `if-else`

Конструкция `if - else`

Синтаксис: (используется для выбора одной из двух ветвей)

```
if
    список A
then
    список B
else
    список C
fi
```

Пример:

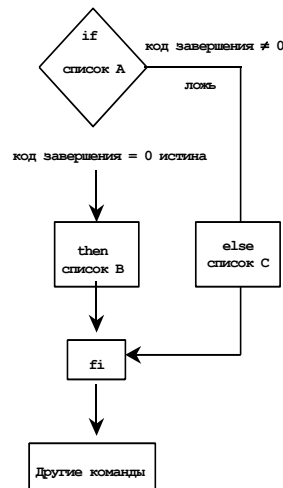
```
if [ "$X" -lt 10 ]
then
    echo X is less than 10
else
    echo X is not less than 10
fi
```

Пояснения

Конструкция **`if-else`** позволяет выполнять одну совокупность команд, если код завершения управляющей команды равен нулю (истина), или другую совокупность команд, если код завершения управляющей команды не равен нулю (ложь).

Ниже приводится алгоритм выполнения данной конструкции **`if`**:

1. Выполняется *список A*.
2. Если код завершения *последней* команды из *списка A* равен **0** (ИСТИНА), выполняются команды из *списка B*, после чего обработка продолжается с первого предложения, записанного после **`fi`**.
3. Если код завершения *последней* команды из *списка A* не равен **0** (ЛОЖЬ), выполняются команды из *списка C*, после чего обработка продолжается с первого предложения, записанного после **`fi`**.

Блок-схема конструкции **if-else**

Следует обратить внимание на то, что *список C* может включать в себя любые команды системы UNIX, в том числе и конструкцию **if**. Пусть пример, приведенный на слайде, нужно изменить так, чтобы проверялись и обрабатывались три ситуации – X меньше 10, больше 10 или равно 10. Эту задачу можно решить следующим образом:

```

if
  [ "$X" -lt 10 ]
then
  echo X is less than 10
else
  if
    [ "$X" -gt 10 ]
  then
    echo X is greater than 10
  else
    echo X is equal to 10
  fi
fi

```

Обратите внимание на то, как стиль написания текста с отступами улучшает его читабельность. Из текста очевидно, с каким **if** связано каждое **fi**. Важно: *каждое if* требует свое **fi**.

17–10. СЛАЙД: Конструкция **case**

Конструкция **case**

Синтаксис: (выбор одной из нескольких ветвей)

```
case слово in
  шаблон1) список A
      ;;
  шаблон2) список B
      ;;
  шаблонN) список N
      ;;
esac
```

Пример:

```
case $ANS in
  yes) echo O.K.
      ;;
  no)  echo no go
      ;;
esac
```

```
case $OPT in
  1) echo option 1 ;;
  2) echo option 2 ;;
  3) echo option 3 ;;
  *) echo no option ;;
esac
```

Пояснения

Конструкцию **if-else** можно использовать для организации выбора одной из нескольких ветвей, но она становится громоздкой, когда количество ветвей становится больше двух или трех. В таких случаях более предпочтительной становится конструкция **case**. Требуемая ветвь выбирается путем последовательных сравнений слова и представленных шаблонов. Сравнения выполняются только как сравнения строк. Когда найден соответствующий шаблон, выполняется ассоциированный с ним список команд. Каждый список команд завершается двумя знаками "точка с запятой" (; ;). По окончании выполнения выбранного списка команд, обработка продолжается с предложений, записанных после **esac**.

Слово обычно определяется как значение переменной shell'a.

Шаблон записывается в таком же виде, как и при генерации имен файлов, хотя здесь не выполняется сопоставление с именами файлов.

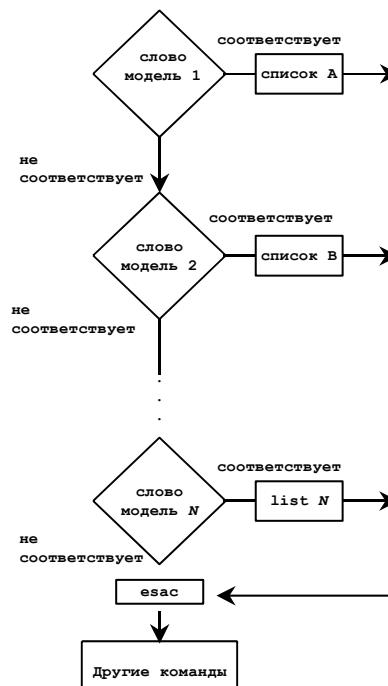
Допускается использовать следующие специальные символы:

- * Соответствует любой строке символов, в том числе и пустой
- ? Соответствует любому одному символу

[...] Соответствует одному любому символу из тех, которые указаны в скобках. Два символа, разделенные знаком минус, соответствуют одному любому символу, находящемуся между указанными двумя.

В задании шаблонов допускается использовать символ |, который означает *OR (ИЛИ)*.

Следует обратить внимание на то, что правая скобка и точки с запятыми являются обязательными.



Блок-схема конструкции **case**

Конструкция **case** обычно используется для организации интерфейсов на основе меню или интерфейсов, позволяющих принимать решение на основе различных опций, введенных пользователем.

17–11. СЛАЙД: Конструкция **case**: примеры задания шаблонов

Конструкция **case**: примеры задания шаблонов

В шаблонах инструкции **case** можно использовать те же специальные символы, что и для генерации имен файлов

```
$ cat menu_with_case
echo "                COMMAND MENU "
echo "      d to display time and date "
echo "      w to display logged-in users "
echo "      l to list contents of current directory "
echo "      Please enter your choice : "
read choice
case $choice in
    [dD]*)    date ;;
    [wW]*)    who ;;
    l*|L*)    ls ;;
    *)        echo Invalid selection ;;
esac
$
```

Пояснения

На данном слайде приведен пример конструкции **case**, модели в которой менее очевидны, чем на предыдущем слайде. С помощью шаблонов можно представить ответы пользователей, не чувствительные к регистру, или же ответы, которые содержат *ту или иную* подстроку.

Часто после всех шаблонов в конструкции **case** записывается шаблон ***)**, в котором пользователю выводится сообщение о том, что он не ввел правильный ответ.

17–12. СЛАЙД: Shell программирование: ветвление (заключение)

Shell программирование: ветвление (заклучение)

Код завершения	Возвращается числовое значение после выполнения каждой команды – echo \$?	
Сравнение чисел	["\$num1" -lt "\$num2"]	
Сравнение строк	["\$string1" = "\$string2"]	
Анализ файлов	test -f имя_файла	
exit n	Завершение программы и формирование кода завершения	
if	case слово in	
 список_A	шаблон1) список_команд	
then	;;	
 список_B	шаблон2) список_команд	
else	;;	
 список_C	*) список_команд	
fi	;;	
	esac	
Решение принимается на основе кода завершения последней команды списка_A	Строка слово сопоставляется с каждым шаблоном	

Пояснения

17–13. УПРАЖНЕНИЯ: Shell программирование: ветвление

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Присвойте переменной *X* значение какой-либо строки. Используя команду **test**, определите, является ли строка *xyz* значением переменной *X*. (ПОДСКАЗКА: нужно вывести на экран код завершения команды **test**)
2. Присвойте переменной *Y* значение какого-либо числа. Используя команду **test**, определите, является ли значение переменной *Y* строго положительным. (ПОДСКАЗКА: нужно вывести на экран код завершения команды **test**)
3. Напишите shell процедуру, включив в нее конструкцию **if**, которая выведет на экран слово *yes*, если переданный в процедуру аргумент имеет значение *abc*, и *no* в противном случае.
4. Напишите shell процедуру, которая выводит пользователю приглашение на ввод числа; запоминает введенное число в переменной *Y*; использует конструкцию **if**, которая выводит текст *Y is positive*, если значение *Y* действительно больше нуля, и *Y is not positive* в противном случае; выводит на экран значение *Y*. (ПОДСКАЗКА: ввод ответа пользователя выполняется с помощью команды **read**)

5. Напишите shell процедуру, которая определяет количество аргументов командной строки и выводит значения аргументов, если в процедуру переданы три аргумента, или сообщение об ошибке в противном случае. (ПОДСКАЗКА: Количество аргументов командной строки определяет специальная переменная shell'a `$#`. Какая специальная переменная shell'a определяет все аргументы командной строки?)

6. Напишите shell процедуру, которая предлагает пользователю ввести ответ и в зависимости от ввода выполняет следующее:
 - Если введено *A*, процедура выводит на экран приветствие *good morning*.
 - Если введено *B* или *b*, процедура выводит на экран приветствие *good afternoon*.
 - Если введено *C* или *quit*, процедура завершает свою работу.
 - Если введено что-то другое, на экран выводится сообщение об ошибке и процедура завершается с кодом 99.

7. Напишите shell процедуру, которая предлагает пользователю ввести некоторое входное имя, после чего проверяет, зарегистрирован ли в системе пользователь с таким именем. Если пользователь с таким именем в системе есть, процедура выводит на экран маршрутное имя *HOME* каталога этого пользователя, в противном случае – сообщение об ошибке.

8. С помощью команды **date** определите время суток: утро (до 12:00), день (между 12:00 и 18:00) или вечер (после 18:00). Напишите shell процедуру **greeting**, которая в зависимости от времени суток выводит на экран соответствующее приветствие: *good morning*, *good afternoon* или *good evening*. (ПОДСКАЗКА: команда **date** использует 24-х часовые часы)

9. Напишите shell процедуру, которая спрашивает у пользователя, хочет ли он увидеть содержимое текущего каталога. Процедура должна сообщить пользователю, что ожидает от него ответа *yes* или *no*, и вывести сообщение об ошибке, если пользователь введет что-то другое. Если пользователь ответит *yes*, процедура должна вывести содержимое текущего каталога. Если пользователь ответит *no*, процедура должна вывести на экран вопрос, содержимое какого каталога он хочет увидеть, ввести ответ пользователя и вывести содержимое указанного каталога. При этом сначала нужно проверить, существует ли такой каталог.
10. Напишите процедуру **mysp**, которая копирует один файл в другой. Процедура должна иметь два аргумента, задающие файл-источник и файл-приемник. В процедуре должны проверяться следующие ситуации:
- файл-источник и файл-приемник не являются одним и тем же файлом;
 - файл-приемник не является каталогом;
 - файл-источник существует;
 - если файл-приемник существует и является обычным файлом, необходимо спросить пользователя, действительно ли он хочет его перезаписать.
11. Напишите shell процедуру **options**, которая работает по следующим правилам, в зависимости от заданных аргументов командной строки:
- Если первый аргумент командной строки **-d**, выполнить команду **date**.
 - Если первый аргумент командной строки **-w**, вывести список всех пользователей, работающих в настоящее время в системе.
 - Если первый аргумент командной строки **-l**, вывести на экран содержимое каталога, имя которого определяет второй аргумент командной строки.
 - Если в командной строке нет аргументов или присутствует более двух аргументов, вывести сообщение об ошибке и установить код завершения **10**.
 - Если аргумент командной строки задает какую-либо другую опцию, вывести на экран соответствующее сообщение об ошибке и установить код завершения **20**.

Модуль 18 — Shell программирование: циклы

Цели:

Изучив этот модуль, вы сможете:

- Использовать конструкцию **while** для организации повторного выполнения участка программы, пока некоторое условие остается истинным
- Использовать конструкцию **until** для организации повторного выполнения участка программы до тех пор, пока некоторое условие не станет истинным
- Использовать итерационную конструкцию **for** для последовательной обработки элементов строки, разделенных пробельными символами

18–1. СЛАЙД: Циклы: введение

Циклы: введение

Назначение:	Повторяющееся выполнение списка команд
Управление:	Основано на анализе <i>кода возврата</i> ключевой команды
Три формы:	<code>while ... do ... done</code> <code>until ... do ... done</code> <code>for ... do ... done</code>

Пояснения

Циклические конструкции позволяют организовать повторяющееся выполнение некоторой совокупности команд; как и в конструкциях ветвления, решение о том, продолжить или прервать выполнение цикла, принимается на основе кода завершения ключевой команды. Часто для управления циклом используется команда **test**.

В отличие от конструкций ветвления, которые начинаются некоторым ключевым словом и заканчиваются им же, только записанным наоборот (**if/fi** и **case/esac**), циклы начинаются соответствующим ключевым словом, за которым следует некоторое условие, а затем тело цикла, ограниченное словами **do/done**.

18–2. СЛАЙД: Команда **let**: вычисление арифметических выражений

Команда **let**: вычисление арифметических выражений

Синтаксис:

let *выражение* или ((*выражение*))

Примеры:

\$ x=10	\$ x=12
\$ y=2	\$ let "x < 10"
\$ let x=x+2	\$ echo \$?
\$ echo \$x	1
12	\$ ((x > 10))
\$ let "x = x / (y + 1)"	\$ echo \$?
\$ echo \$x	0
4	\$ if ((x > 10))
\$ ((x = x + 1))	> then echo x greater
\$ echo \$x	> else echo x not greater
5	> fi
	x greater

Пояснения

Управление циклами часто осуществляется с помощью последовательного увеличения на некоторую постоянную величину некоторой числовой переменной. Команда **let** позволяет использовать в shell процедурах арифметические выражения. Эта команда выполняет арифметические вычисления с числами, представленными в формате "длинное целое" (long integer). Синтаксис команды представлен на слайде; здесь *выражение* представляет собой некоторое арифметическое выражение, в котором используются переменные shell'a и знаки операций, обрабатываемые shell'ом. Использование двойных парных круглых скобок (()) вокруг арифметического выражения подразумевает использование команды **let**. Ниже приведены операции, допускаемые shell'ом для записи выражений, в порядке уменьшения их приоритета.

Операция	Описание
-	унарный минус
!	логическое отрицание
* / %	умножение, деление (нацело), остаток от деления
+ -	сложение, вычитание

<= >= < >	сравнение
== !=	проверка равенства (равно, не равно)
=	присваивание

Для того чтобы изменить порядок вычисления выражения, в записи выражения могут быть использованы круглые скобки, например:

```
let "x=x/(y+1)"
```

Следует обратить внимание на необходимость использования в записи выражения двойных кавычек; они отменяют специальный смысл символов shell'a (в данном примере, круглых скобок). Еще один пример: если в записи выражения используются пробелы для отделения операций и операндов, также нужно использовать двойные кавычки. Вместо команды **let** можно использовать двойные парные круглые скобки **(())**; внутри скобок специальный смысл символов shell'a отменяется:

```
let "x = x + (y / 2)" ИЛИ (( x = x + (y / 2) ))
```

Если используются логические операции и операции отношения (**!**, **<=**, **>=**, **<**, **>**, **==**, **!=**), переменная shell'a **?**, определяющая код завершения, принимает значение *истина* или *ложь* в зависимости от результата вычисления выражения (**0** если *истина*, **1** если *ложь*). Опять же, для того чтобы shell не интерпретировал символы «меньше» и «больше» как символы перенаправления ввода/вывода, необходимо использовать двойные кавычки (или двойные парные круглые скобки).

18–3. СЛАЙД: Конструкция **while**

Конструкция **while**

Цикл повторяется, пока условие истинно

Синтаксис:

```
while
    список_A
do
    список_B
done
```

Пример:

```
$ cat test_while
X=1
while (( X <= 10 ))
do
    echo hello X is $X
    let X=X+1
done

$ test_while
hello X is 1
hello X is 2
.
.
hello X is 10
```

Пояснения

Конструкция **while** – это способ организации цикла, предоставляемый shell'ом, в котором команды, составляющие тело цикла (*список_B*), выполняются, **пока (while)** условие истинно. Условие определяется кодом завершения последней команды из *списка_A*. Часто для управления выполнением цикла используются команды **test** или **let**, но с таким же успехом может быть использована *любая* команда, поскольку она генерирует код завершения.

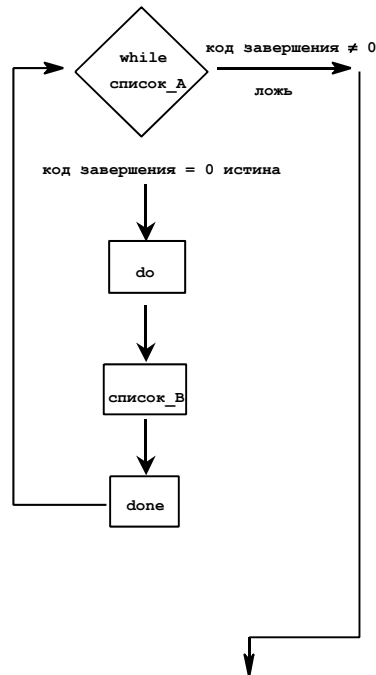
В примере, приведенном на слайде, вместо команды **let** с таким же успехом может быть использована команда **test**:

```
$ X=1
$ while [ $X -le 10 ]
> do
>     echo hello X is $X
>     let X=X+1
> done
```

Конструкция **while** работает следующим образом:

1. Выполняются команды из *списка_A*.

2. Если код завершения *последней* команды из *списка_A* равен **0** (истина), выполняется *список_B*.
3. Выполняется переход к пункту 1.
4. Если код завершения *последней* команды из *списка_A* не равен **0** (ложь), управление передается первой команде, следующей за ключевым словом **done**.



Блок-схема конструкции **while**

ВНИМАНИЕ: Следует обращать внимание на то, чтобы цикл **while** не оказался бесконечным, т.е. таким циклом, в котором управляющая команда *всегда* возвращает код завершения истина.

```
$ cat while_infinite
while
    true
do
    echo hello
done

$ while_infinite
hello
hello
.
.
.
[Ctrl] + [c]
```


18–4. СЛАЙД: Конструкция **while**: примеры

Конструкция **while**: примеры

Пример А:

Повторяется, пока ans равно yes

```
ans=yes
while
[ "$ans" = yes ]
do
echo Enter a name
read name
echo $name >> file.names
echo "Continue?"
echo Enter yes or no
read ans
done
```

Пример В:

*Повторяется, пока в командной строке
есть аргументы*

```
while (( $# != 0 ))
do
if test -d $1
then
echo contents of $1:
ls -F $1
fi
shift
echo There are $# items
echo left on the cmd line.
done
```

Пояснения

На слайде приведены еще два примера использования конструкции **while**. В примере А пользователю выводится приглашение на ввод, и на основании ответа пользователя принимается решение, продолжить или завершить выполнение цикла. В примере В тело цикла выполняется для каждого аргумента командной строки. Если аргумент представляет каталог, выводится его содержимое, в противном случае аргумент просто пропускается. Обратите внимание на использование команды **shift**: она позволяет последовательно получать доступ к каждому аргументу командной строки. Совместное использование команд **shift** и **while** позволяет создавать очень гибкие и мощные циклы. Не имеет значения, сколько аргументов командной строки – 1 или 100 – переданы в процедуру; цикл будет продолжаться, пока не будут обработаны все аргументы.

Следует обратить внимание на то, что если нужно, чтобы тело цикла **while** выполнялось, по крайней мере, один раз, необходимо выполнить некоторые предварительные установки. В примере А тело цикла выполнится обязательно хотя бы один раз, так как переменной *ans* предварительно присвоено значение **yes**. В примере В, если при вызове процедуры в командной строке не будет указан ни один аргумент (**\$#** равно 0), тогда тело цикла не будет выполняться ни разу.

18–5. СЛАЙД: Конструкция `until`

Конструкция `until`

Цикл повторяется до тех пор, пока условие не станет истинным

Синтаксис:

```
until
    список_A
do
    список_B
done
```

Пример:

```
$ cat test_until
X=1
until (( X > 10 ))
do
    echo hello X is $X
    let X=X+1
done

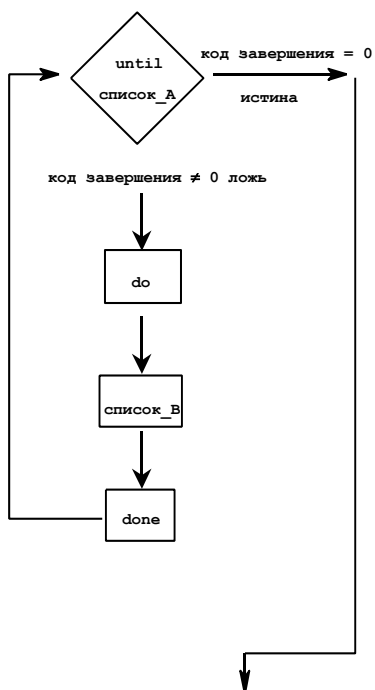
$ test_until
hello X is 1
hello X is 2
.
.
.
hello X is 10
```

Пояснения

Конструкция **`until`** – это другой способ организации цикла, предоставляемый shell’ом, в котором команды, составляющие тело цикла (*список_B*), выполняются **до тех пор, пока** (**`until`**) условие не станет истинным. Как и в цикле **`while`**, условие определяется кодом завершения последней команды из *списка_A*.

Конструкция **`until`** работает следующим образом:

1. Выполняются команды из *списка_A*.
2. Если код завершения *последней* команды из *списка_A* не равен 0 (ложь), выполняется *список_B*.
3. Выполняется переход к пункту 1.
4. Если код завершения *последней* команды из *списка_A* равен 0 (ИСТИНА), управление передается первой команде, следующей за ключевым словом **`done`**.



Блок-схема конструкции `until`

ВНИМАНИЕ: Следует обращать внимание на то, чтобы цикл `until` не оказался бесконечным, т.е. таким циклом, в котором управляющая команда *всегда* возвращает код завершения `false`.

```
$ X=1
$ until
> [ $X -eq 0 ]
> do
> echo hello
> done
hello
hello
.
.
.
[Ctrl] + [c]
```

18–6. СЛАЙД: Конструкция `until`: примеры

Конструкция `until`: примеры

Пример А:

Повторяется до тех пор, пока `ans` не станет равным `no`

```
ans=yes
until
[ "$ans" = no ]
do
  echo Enter a name
  read name
  echo $name >> file.names
  echo "Continue?"
  echo Enter yes or no
  read ans
done
```

Пример В:

Повторяется до тех пор, пока в командной строке не останется аргументов

```
until (( $# == 0 ))
do
  if test -d $1
  then
    echo contents of $1:
    ls -F $1
  fi
  shift
  echo There are $# items
  echo left on the cmd line.
done
```

Пояснения

На слайде с помощью конструкции `until` реализованы те же примеры, которые рассматривались для конструкции `while`. Обратите внимание на то, что для того чтобы правильно использовать логику конструкции `until`, условие проверки в ней изменено на обратное по сравнению с условием, использованным в соответствующей конструкции `while`.

Обратите также внимание на то, что несколько изменилась и реакция на ответ пользователя. Когда использовалась конструкция `while`, выполнение цикла продолжалось *только* в том случае, если пользователь введет ответ **yes**. Это очень жестко задано в условии продолжения цикла. Использование конструкции `until` приводит к тому, что выполнение цикла продолжается, если пользователь введет любой ответ, кроме **no**. Такое условие продолжения цикла уже не является очень жестким. Указанные особенности следует принимать во внимание, отдавая предпочтение той или иной конструкции при создании пользовательского интерфейса.

При использовании конструкции `until` нет необходимости в предварительной установке значения переменной `ans`, так как shell инициализирует переменные значением NULL. Так как NULL не равно **no**, команда `test` вернет значение ложь, и выполнение цикла продолжится. Необходимо только заключить `$ans` в двойные кавычки, чтобы гарантировать синтаксически правильное использование команды `test`.

18–7. СЛАЙД: Конструкция `for`

Конструкция `for`

Для каждого элемента из **списка** повторяется выполнение цикла, при этом **переменной** присваивается значение очередного элемента из **списка**, и так до тех пор, пока **список** не будет исчерпан

Синтаксис:

```
for переменная in список
do
    список_A
done
```

Пример:

```
$ cat test_for
for X in 1 2 3 4 5
do
echo "2 * $X is \c"
let X=X*2
echo $X
done
```

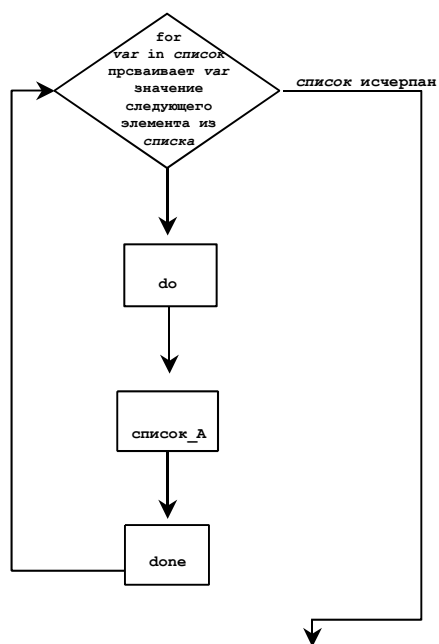
```
$ test_for
2 * 1 is 2
2 * 2 is 4
2 * 3 is 6
2 * 4 is 8
2 * 5 is 10
```

Пояснения

В приведенном на слайде примере ключевыми являются слова **for**, **in**, **do** и **done**. **Переменная** определяет имя переменной shell'a, которой при выполнении **цикла for** присваиваются определенные значения. **Список** представляет собой последовательность строк, разделенных пробелами и/или знаками табуляции; именно эти строки и присваиваются **переменной** во время исполнения каждой итерации цикла.

Конструкция **for** работает следующим образом:

1. Переменной shell'a **переменная** присваивается значение первой строки из **списка**.
2. Выполняется последовательность команд **список_A**.
3. Переменной shell'a **переменная** присваивается значение следующей строки из **списка**.
4. Выполняется последовательность команд **список_A**.
5. Так продолжается до тех пор, пока не будут обработаны все элементы из **списка**.



Блок-схема конструкции `for`

18–8. СЛАЙД: Конструкция **for**: примеры

Конструкция **for**: примеры

Пример A:

```
$ cat example_A
for NAME in $(grep home /etc/passwd | cut -f1 -d:)
do
    mail $NAME < mtg.minutes
    echo mailed mtg.minutes to $NAME
done
```

Пример B:

```
$ cat example_B
for FILE in *
do
    if
        test -d $FILE
    then
        ls -F $FILE
    fi
done
```

Пояснения

Конструкция **for** является очень гибкой конструкцией. Она может организовать цикл для любого списка, который может быть получен любым способом. Например, список можно получить, выполняя подстановку команд, как показано в первом примере на слайде. Используя конвейеры и фильтры, можно получить практически любой список.

Если один и тот же список нужно использовать многократно, его можно сохранить в файле. В дальнейшем для генерации этого списка в цикле **for** можно использовать команду **cat**:

```
$ cat students
user1
user2
user3
user4

$ cat for_students_file_copy
for NAME in $(cat students)
do
    cp test.file /home/$NAME
    chown $NAME /home/$NAME/test.file
    chmod g-w,o-w /home/$NAME/test.file
    echo done $NAME
done
$
```

Обработка аргументов командной строки

Список из *аргументов командной строки* можно подставить в конструкцию **for** двумя способами:

```
for i in $*           или
do
    cp $i $HOME/backups
done
```

```
for i
do
    cp $i $HOME/backups
done
```


18–9. СЛАЙД: Команды **break**, **continue** и **exit**

Команды **break**, **continue** и **exit**

break [<i>n</i>]	Прекращается выполнение <i>n</i> окаймляющих циклов, начиная с текущего (если <i>n</i> не задано, оно принимается равным 1); управление передается команде после <i>n</i> -го done
continue [<i>n</i>]	Прекращается выполнение текущей итерации цикла; управление передается на <i>начало</i> <i>n</i> -го окаймляющего цикла, начиная с текущего (если <i>n</i> не задано, оно принимается равным 1) – начинает выполняться следующая итерация этого цикла
exit [<i>n</i>]	Прекращается выполнение shell процедуры; код завершения устанавливается равным <i>n</i>

Пояснения

Всегда могут возникнуть ситуации, когда необходимо прервать выполнение цикла до его естественного завершения. Для этого используются команды **break** и **continue**, представляющие собой команды безусловного перехода. Обычно они используются, когда при выполнении очередной итерации цикла обнаруживается ошибка, требующая прекращения этой итерации. Команда **exit** используется в тех случаях, когда ситуация не может быть исправлена и необходимо завершить выполнение самой процедуры.

Команда **break** прекращает выполнение текущего цикла и передает управление команде, следующей непосредственно после ключевого слова **done**. Таким образом, можно немедленно прервать выполнение нужного цикла и продолжить выполнение процедуры с команд, расположенных после него.

Логика работы команды **continue** несколько иная. Когда в теле цикла встречается эта команда, она пропускает команды, оставшиеся в теле цикла, и передает управление на заголовок цикла. Таким образом, команда **continue** позволяет завершить выполнение текущей итерации цикла и продолжить выполнение прерванного цикла с его заголовка.

В циклах **while** и **until** выполнение цикла продолжается с выполнения управляющего списка команд (записанного непосредственно после соответствующих ключевых слов). В цикле **for** выполнение цикла продолжается с присваивания переменной следующего значения из списка. После этого выполнение циклов продолжается в соответствии с их логикой работы.

Команда **exit** прерывает выполнение shell процедуры и устанавливает код завершения этой процедуры, равный значению своего аргумента, если этот аргумент указан. Если же выполняется команда **exit** без аргумента, код завершения shell процедуры устанавливается равным коду завершения команды, выполнявшейся непосредственно перед **exit**. В shell'e допускается использование функций, для завершения которых используется команда **return**; она выполняется так же, как и **exit**, только завершает выполнение функции, а не всей процедуры.

ПРИМЕЧАНИЕ: Обычно выполнение цикла должно завершаться нормально, когда появляется соответствующее условие, определенное в заголовке цикла (циклы **while**, **until**) или когда будет исчерпан список (цикл **for**). Рассматриваемые здесь команды должны выполняться только в тех случаях, когда при выполнении цикла обнаруживается какая-либо особая ситуация или ошибка.

Пример

```
while
  cmd1
do
  cmdA
  cmdB
  while
    cmdC
  do
    cmdE
    break 2
    cmdF
  done
  cmdJ
  cmdK
done
cmdX
```

1. Какая команда будет выполняться после выполнения команды **break 2**?
2. Как будет выполняться цикл, если команду **break 2** заменить командой **break**?
3. Как будет выполняться цикл, если команду **break 2** заменить командой **continue 2**?
4. Как будет выполняться цикл, если команду **break 2** заменить командой **continue**?

18–10. СЛАЙД: Команды **break** и **continue**: пример

Команды **break** и **continue**: пример

```
while
  true
do
  echo "Enter file to remove:\c"
  read FILE
  if test ! -f $FILE
  then
    echo $FILE is not a regular file
    continue
  fi
  echo removing $FILE
  rm $FILE
  break
done
```

Пояснения

В приведенном на слайде примере активно используются команды **break** и **continue**. Команды выполняются, так как в цикле **while** в качестве управляющей команды используется **true**, которая всегда в качестве результата возвращает значение *истина*; это значит, что данный цикл является *бесконечным* и будет выполняться всегда, если только какая-нибудь команда в теле цикла не завершит его (что и делает команда **break**). Если введенное имя файла определяет *не* обычный файл, выводится сообщение об ошибке, и выполнение команды **continue** приводит к тому, что на экран снова выводится приглашение пользователю ввести имя файла. Если введенное имя файла определяет *обычный* файл, этот файл удаляется, и команда **break** осуществляет выход из бесконечного цикла.

18–11. СЛАЙД: Shell программирование: циклы (заключение)

Shell программирование: циклы (заключение)

<code>let <i>выражение</i></code>	Вычисление арифметического выражения
<code>((<i>выражение</i>))</code>	Вычисление арифметического выражения
<code>while <i>условие истинно</i> do ... done</code>	Цикл while
<code>until <i>условие не станет истинным</i> do ... done</code>	Цикл until
<code>for <i>переменная in список</i> do ... done</code>	Цикл for
<code>break [n]</code>	Выход из цикла
<code>continue [n]</code>	Завершение текущей итерации цикла
<code>exit [n]</code>	Завершение выполнения процедуры

Пояснения

18–12. УПРАЖНЕНИЯ: Shell программирование: циклы

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Напишите shell процедуру **double_it**, которая предлагает пользователю ввести число и выводит на экран его удвоенное значение.
2. Напишите shell процедуру **sum_them**, которая предлагает пользователю ввести 10 чисел. Процедура должна вычислить сумму всех введенных чисел и вывести на экран полученный результат. (ПОДСКАЗКА: добавляйте к сумме каждое вновь введенное число.)

Если останется время: модифицируйте процедуру **sum_them** таким образом, чтобы количество чисел, сумму которых надо найти, передавалось в процедуру через аргумент командной строки. Например, процедура, вызванная так: **sum_them 6**, должна предложить пользователю ввести шесть чисел и затем вычислить их сумму.

3. Напишите shell процедуру **words_in**, которая предлагает пользователю вводить слова до тех пор, пока не будет введено слово **quit**. Все введенные слова запоминаются. После того как будет введено слово **quit**, введенные слова в порядке, обратном вводу (последнее, предпоследнее, ..., первое) выводятся на экран. Можно ли выполнить это упражнение с помощью цикла **while**? С помощью цикла **until**? Выберите по своему усмотрению какой-нибудь из этих циклов.

Если останется время: выведите все введенные слова в алфавитном порядке.

4. Включите в shell процедуру цикл **for**, который:
 - создает каталоги **Adir**, **Bdir**, **Cdir**, **Ddir**, **Edir**,
 - копирует в каждый каталог файл **funfile**,

- выводит содержимое каждого каталога, чтобы проверить результаты копирования,
- после того как завершается каждая итерация цикла, выводит на экран соответствующее сообщение.

5. Напишите shell процедуру **new_files**, которая допускает задание произвольного количества аргументов командной строки. Эта shell процедура создает новые файлы, имена которых определяются аргументами командной строки (используйте для этого команду **touch**), и выводит на экран сообщение, информирующее пользователя о создании каждого файла.

6. С помощью редактора **vi** создайте файл **mailtest**. В ответ на приглашение shell'a создайте интерактивный цикл **for** для отправки почтового сообщения **mailtest** (с помощью команды **mail**) всем пользователям, работающим в системе. (ПОДСКАЗКА: для генерации списка для цикла **for** используйте команды **who** и **cut** и подстановку команд.)

7. Напишите shell процедуру **my_menu**, которая выводит на экран простое меню, содержащее три альтернативы:

- а. Запуск на исполнение процедуры **double_it** (упражнение 1).
- б. Запуск на исполнение процедуры **sum_them** (упражнение 2).
- с. Выход.

Меню должно заново выводиться на экран каждый раз после выполнения действий, определенных первой и второй альтернативами.

8. Напишите shell процедуру **msg_me**, которая в течение одной минуты каждые 5 секунд выводит на экран сообщение. (ПОДСКАЗКА: посмотрите описание команды **sleep**.) Целесообразно сохранить сообщение, выводимое на экран, в файле, тогда его можно будет легко менять.

9. Напишите shell процедуру **ison**, которая будет *выполняться в фоновом режиме* и каждые 60 секунд будет проверять, работает ли в системе некоторый конкретный пользователь. Имя пользователя передается в процедуру **ison** с помощью аргумента командной строки. Если пользователь работает в системе, на экран следует вывести сообщение, содержащее входное имя пользователя и имя терминала, с которого он вошел в систему. (ПОДСКАЗКА: используйте команду **sleep**.)

Если вы работаете в сети, можете попробовать использовать команду **rwho**.

10. Создайте в своем *HOME* каталоге каталог **.waste**. Напишите shell процедуру **myrm**, которая перемещает все удаляемые вами файлы в каталог **.waste** (он будет служить мусорной корзиной). Это полезный инструмент, позволяющий вам восстанавливать файлы после их удаления. Следует помнить, что система UNIX не имеет функции *отмены удаления*.

Пусть процедура **myrm** позволяет также задавать следующие опции:

- l вывести содержимое корзины
- d очистить корзину

Модуль 19 — Хранение файлов во внешней памяти

Цели:

Изучив этот модуль, вы сможете:

- Использовать команду **tar** для резервного копирования файлов на ленту
- Использовать команды **find** и **cpio** для резервного копирования файлов на ленту
- С помощью команд **tar** и **cpio** восстанавливать файлы, сохраненные на ленте

19–1. СЛАЙД: Сохранение файлов на ленте

Сохранение файлов на ленте

- Для записи файлов на ленту необходимо знать имя *файла устройства* для накопителя на магнитной ленте
- Типичные имена ленточного файла устройства
 - `/dev/rmt/0m` 9-ти дорожечная магнитная лента или DDS лента (старое имя)
 - `/dev/rmt/c0t3d0BEST` 9-ти дорожечная магнитная лента или DDS лента
- Необходимо узнать у системного администратора, какой файл устройства осуществляет доступ к накопителю на магнитной ленте
- Команды, которые выполняют резервное копирование

`tar`
`cpio`

Пояснения

Обычному пользователю системы UNIX в процессе его работы в системе часто требуется сохранять копии файлов на сменном носителе. Достаточно часто для хранения резервных копий файлов используется 9-дорожечная магнитная лента (1/2-дюймовая катушка магнитной ленты), или DDS формат ленты DAT. В этом модуле рассматриваются основные возможности запоминания на ленте и восстановления с ленты резервных копий файлов. Следует иметь в виду, что обычно за резервное копирование всей системы отвечает системный администратор, поэтому все свои действия по созданию резервных копий следует координировать именно с ним.

ПРИМЕЧАНИЕ: Единственный способ восстановить удаленные файлы – это восстановить эти файлы из резервных копий, сохраненных на ленте.

Замечания по поводу магнитных лент формата DDS

С самого начала введения продукта HP DAT в 1990 г. HP поддерживает в качестве внешней среды хранения только картриджи DDS. Аудио ленты DAT не поддерживаются, и использование лент DAT делают недействительными гарантии на драйверы.

Картриджи DDS соответствуют более высоким стандартам по сравнению с обычной лентой DAT. Картриджи DDS и магнитные ленты очень строго тестируются, прежде чем им будет разрешено носить логотип DDS. Не DDS ленты могут работать с DAT драйверами, но при

этом могут иметь место потери данных, замятие ленты, порча магнитных головок и другие повреждения.

Убедитесь в том, что вы действительно используете DDS формат DAT лент. Некоторые ленты, маркированные как *Data Grade*, совсем не обязательно являются лентами формата DDS.

На лентах формата DDS используется следующий логотип:



Логотип лент формата DDS

19–2. СЛАЙД: Команда **tar**

Команда **tar**

Синтаксис: **tar -ключ [f файл_устройства] [файл. . .]**

Примеры:

Создать архив:

```
$ tar -cvf /dev/rmt/0m myfile
```

Прочитать оглавление архива:

```
$ tar -tvf /dev/rmt/0m
```

Извлечь файл из архива:

```
$ tar -xvf /dev/rmt/0m myfile
```

Пояснения

Команда **tar** ведет архив файлов на магнитной ленте. Она сохраняет файлы на магнитной ленте и восстанавливает их. Функции команды определяются ее первым аргументом, называемым **ключом**.

Возможны следующие значения ключа:

- | | |
|----------|--|
| c | Создается (create) новый архив |
| x | Файлы извлекаются (extracted) из архива |
| t | Выводится оглавление (table of contents) архива |
| r | Файлы добавляются в конец архива |
| u | В архив добавляются вновь созданные или модифицированные файлы |

К указанным выше ключам могут быть добавлены модификаторы, уточняющие поведение команды:

- | | |
|----------|--|
| v | Многословный (verbose) режим – на экран выводятся имена файлов, записываемых в архив или извлекаемых из архива |
|----------|--|

f файл Определяется **файл**, в который записывается архив. Примечание: **файл** не обязательно должен быть файлом устройства. Можно создать архивный файл в своем каталоге, и он будет размещен на диске. По умолчанию используется файл **/dev/rmt/0m**.

Система Linux



Версия команды **tar** в системе Linux имеет большее количество различных опций, хотя и работает так же, как и в системе HP-UX. Подробное описание можно найти в соответствующей статье Справочника.

19–3. СЛАЙД: Команда **cpio**

Команда **cpio**

Два режима:

cpio -o[сvх]

Создание архива. Список имен файлов вводится из **stdin**. Архив записывается в **stdout**

cpio -i[cdmtuvх]

Извлечение из архива. Архив читается из **stdin**

Примеры:

Создать архив файлов в текущем каталоге:

```
$ find . | cpio -ocv > /dev/rmt/0m
```

Извлечь из архива все файлы:

```
$ cpio -icdmv < /dev/rmt/0m
```

Пояснения

Данная команда создает архивные копии файлов и каталогов системы HP-UX. Имя **cpio** означает: *копирование входа на выход (copy input to output)*. Команда **cpio** имеет два режима:

- o** *Создать резервную копию.* Имена файлов читаются из стандартного ввода, копия каждого файла пишется в стандартный вывод.
- i** *Восстановить из резервной копии.* Из стандартного ввода читается резервная копия файлов, из которой извлекаются нужные файлы и восстанавливаются на диске.

Когда создается резервная копия файлов, команда **cpio -o** использует стандартный ввод в качестве поставщика имен файлов и стандартный вывод в качестве архива. В соответствии с этим необходимо сообщить команде, что архив должен быть записан в файл магнитной ленты, и каким-либо способом создать список имен файлов, которые нужно поместить в архив. Обычно это достигается с помощью перенаправления стандартного вывода для **cpio** и передачи по конвейеру выхода команды **find** на вход команды **cpio**.

Когда файлы извлекаются из архива, команда **cpio -i** читает архив из стандартного ввода (требуется перенаправление стандартного ввода на специальный файл устройства для магнитной ленты) и восстанавливает файлы на диске. Имена восстановленных файлов будут создаваться в зависимости от того, какие маршрутные имена файлов использовались при создании архива – абсолютные или относительные.

Ниже приведено описание некоторых дополнительных опций, используемых вместе с главными опциями **-o** и **-i**.

-o	-i	Назначение опции
-c	-c	Записать заголовки файлов в формате ASCII. (Если используется вместе с опцией -o , обязательно должен использоваться и вместе с опцией -i)
-	-d	Воссоздать в нужном виде структуру каталога
-	-m	Сохранить текущее время модификации файла (важно для управления версиями файла)
-	-t	Вывести оглавление архива
-	-u	Безусловное восстановление (если файл с таким именем уже существует, указание данной опции приводит к перезаписи этого файла)
-v	-v	Выводить список имен копируемых файлов
-x	-x	Обрабатывать специальные файлы (файлы устройств)

Дополнительные примеры

- Получить оглавление архива:

```
$ cpio -ict < /dev/rmt/0m
```
- Извлечь из архива один файл:

```
$ cpio -icudm "filename" < /dev/rmt/0m
```
- Извлечь из архива все файлы, имена которых удовлетворяют заданному шаблону:

```
$ cpio -icudm '*filename*' < /dev/rmt/0m
```

Замечания по поводу команды **find**

- Обычно для создания списка имен файлов, которые нужно копировать, вместе с командой резервного копирования используется команда **find**. Следует заметить, что команда **find** может создавать список как относительных (**find .**), так и абсолютных (**find /home/user3**) маршрутных имен файлов. Метод, используемый для генерации имен файлов, и определяет, какие имена файлов будут сохранены в резервной копии.

Синтаксис:

```
find список_маршрутных_имен [выражение]
```

В **выражении** можно использовать различные ключевые слова, которые определяют критерий поиска. Подробное описание команды можно найти в статье **find(1)** Справочника.

Система Linux



Версия команды **cpio** в системе Linux имеет большее количество различных опций, хотя и работает так же, как и в системе HP-UX. Подробное описание можно найти в соответствующей статье Справочника.

19–4. УПРАЖНЕНИЯ: Хранение файлов во внешней памяти

Указания

Выполните приведенные ниже упражнения. Запишите команды, которые вы используете для решения задач, указав в них в качестве файла устройства **/dev/rmt/0m**. Если доступен конкретный драйвер устройства, ваш инструктор сможет реально выполнить некоторые из этих команд. Вы можете заменить имя файла устройства **/dev/rmt/0m** в командах резервного копирования именем обычного файла, если вы хотите выполнить эти упражнения, не обращаясь к лентопротяжному устройству:

```
$ tar cf /tmp/архивный_файл каталог
```

1. Используя команду **tar**, создайте резервную копию всех файлов, находящихся в вашем *HOME* каталоге, имена которых начинаются буквами **abc**.
2. Получите оглавление созданного архива.
3. Используя команды **find** и **cpio**, создайте резервную копию всех файлов, находящихся в вашем *HOME* каталоге и во всех его подкаталогах, сохранив структуру каталога.
4. Удалите файл **backup** из вашего текущего каталога. Затем восстановите его из резервной копии с помощью команды **cpio**.

5. Создайте каталог **\$HOME/tree.cp**. В статье **cpio(1)** найдите описание режима прохода (pass mode) команды **cpio**. Используя команду **cpio** в режиме прохода, воссоздайте структуру каталога **\$HOME/tree** в каталоге **\$HOME/tree.cp**.

Приложение А — Краткий справочник команд

Цели:

Приводится список часто используемых команд вместе с объяснением правил их использования.

А–1. Краткий справочник команд

Команды общего назначения

exit	Завершить сеанс работы в системе
man команда	Вывести статью <i>Справочника</i> для команды
laserROM	Начать сеанс связи с документацией HP LaserROM
абсолютное маршрутное имя	Полное определение места размещения файла или каталога в иерархии файловой системы UNIX. <i>ВСЕГДА</i> начинается символом /
относительное маршрутное имя	Определение места размещения файла или каталога, начиная от текущей позиции в иерархии файловой системы UNIX
.	Текущий каталог
..	Родительский каталог
pwd	Вывести место размещения текущего каталога в иерархии файловой системы
cd каталог	Перейти в указанный каталог
cd	Перейти в <i>HOME</i> каталог
mkdir каталог	Создать каталог
rmdir каталог	Удалить каталог
ls файл или каталог	Вывести характеристики файла или содержимое каталога
ls -a	Вывести информацию обо всех файлах, включая и скрытые файлы
ls -F	Вывести имена файлов с указанием их типа / – обозначает каталог * – обозначает исполняемый файл – обозначает обычный файл – обозначает FIFO файл (именованный канал)
ls -l	Вывести полную информацию о файле, в том числе права доступа, права собственности и размер файла rwX rwX rwX владелец группа остальные r – право чтения (значение режима = 4)

w	– право записи (значение режима = 2)
x	– право исполнения (значение режима = 1)
ll	Сокращенное название HP-UX для команды ls -l
lsf	Сокращенное название HP-UX для команды ls -F
lsr	Сокращенное название HP-UX для команды ls -R
lsx	Сокращенное название HP-UX для команды ls -x
cat [<i>файл</i>]	Вывести содержимое файла
more [<i>файл</i>]	Вывести содержимое файла поэкранно space – следующий экран Return – следующая строка q – завершить работу more
tail -n <i>файл</i>	Вывести на экран последние <i>n</i> строк файла
pr <i>файл</i>	Форматировать файл для печати
lp <i>файл</i>	Поставить файл в очередь на печать (HP-UX)
lpr <i>файл</i>	Поставить файл в очередь на печать (Linux)
pr <i>файл</i> lp	Форматировать и напечатать файл (HP-UX)
pr <i>файл</i> lpr	Форматировать и напечатать файл (Linux)
lpstat -t	Вывести информацию о состоянии принтеров и системы печати (HP-UX)
lpq	Вывести информацию о состоянии принтеров и системы печати (Linux)
cancel <i>номер_задания</i>	Удалить задание из очереди к принтеру (HP-UX)
lprm <i>номер_задания</i>	Удалить задание из очереди к принтеру (Linux)
touch <i>файл</i>	Создать новый файл или обновить временные характеристики существующего файла
cp [-i] <i>f1 f2</i>	Копировать <i>f1</i> в <i>f2</i>
cp [-i] <i>f1 f2... каталог</i>	Копировать файлы в <i>каталог</i>
ln [-i] <i>f1 f2</i>	Создать ссылку <i>f2</i> на файл <i>f1</i> <i>f1</i> и <i>f2</i> обеспечивают доступ к одному и тому же пространству на диске

Приложение А
Краткий справочник команд

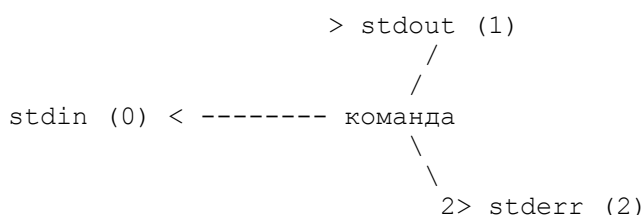
ln -s каталог1 каталог2	Создать символическую ссылку каталог2 на каталог1
mv [-i] f1 f2	Переименовать f1 в f2
mv [-i] f1 f2 ... каталог	Переместить файлы в каталог
mv [-i] каталог1 каталог2	Переименовать каталог1 в каталог2
rm f1 f2 ...	Удалить файлы
rm -i f2 f2 ...	Удалить файлы интерактивно (с подтверждением)
rm -r каталог	Удалить каталог и ВСЕ, что в нем находится
who	Вывести информацию о пользователях, работающих в системе
who am i	Вывести входное имя пользователя и информацию о его терминальной сессии
whoami	Вывести входное имя пользователя
news	(HP-UX) Вывести системные новости (обновляет файл \$HOME/.news_time)
write пользователь	Начать интерактивную связь с пользователем (задается входное имя пользователя)
mesg y	Разрешить вывод сообщений на терминал
mesg n	Запретить вывод сообщений на терминал
mail пользователь	Отправить почтовое сообщение пользователю
mail	Почитать почтовые сообщения ? – получить справку о mail d – удалить предыдущее сообщение s файл – сохранить сообщение в файле q – завершить работу mail
mailx пользователь	Послать сообщение пользователю
mailx	Прочитать почтовые сообщения
elm	Утилита HP для отправления и чтения почтовых сообщений
echo строка	Вывести на экран строку
banner строка	Вывести на экран строку большими буквами

date	Вывести системные время и дату
id	Вывести идентификатор пользователя и статус группы
chmod режим файл	Изменить права доступа к файлу в соответствии с режимом chmod +x file chmod 777 file
umask режим	Снять права, заданные режимом , из прав доступа, устанавливаемых по умолчанию при создании нового файла
chown пользователь файл	Передать пользователю права владения файлом ; см. файл /etc/passwd (в системе Linux по умолчанию доступно только администратору системы)
chgrp группа файл	Изменить задание группы для файла ; см. файл /etc/group (в системе Linux по умолчанию доступно только администратору системы)
su имя_пользователя	Переключить идентификатор пользователя на имя_пользователя
newgrp имя_группы	Переключить идентификатор группы на имя_группы
passwd	Изменить пароль для входа в систему
vi имя_файла	Начать сеанс редактирования vi для указанного файла

Генерация имен файлов

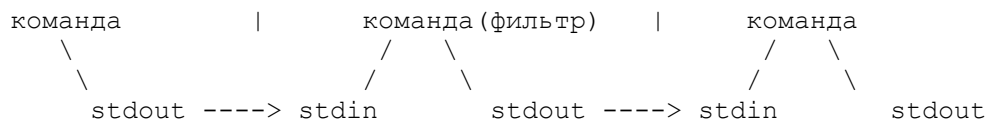
*	Соответствует любой строке символов (в том числе и пустой)
?	Соответствует одному любому символу
[amqp]	Соответствует одному из указанных символов, в данном примере или a , или m , или q , или p
[a-z]	Соответствует любому одному символу из указанного диапазона, в данном примере от a и до z
[!a-z]	Соответствует одному любому символу КРОМЕ тех, которые находятся в указанном диапазоне

Перенаправление ввода/вывода в файл: команда <> файл



команда < файл	Заставить команду читать из файла вместо stdin
команда > файл	Направить stdout команды в файл
команда 2> файл.err	Направить stderr команды в файл
команда > файл 2> файл.err	Направить stdout и stderr в файлы
команда >&2	Направить stdout в stderr Полезно при генерации с помощью echo сообщения об ошибке: echo сообщение об ошибке >&2

Конвейер: cmd <> cmd



команда1 команда2	Перехватить вывод команды1 и направить его на стандартный ввод команды2
----------------------------	---

Переменные shell'a

имя=lisa	Присвоить значение переменной имя
export имя	Экспортировать переменную имя в среду
set	Вывести определения всех переменных в текущем shell'e
env	Вывести только переменные среды
echo enter a name	Вывести приглашение пользователю на ввод
read имя	Прочитать данные из стандартного ввода и присвоить их переменной имя
echo \$имя	Вывести значение (\$) переменной имя
grep \$имя /etc/passwd	Поиск вхождения значения имени в файле /etc/passwd

<code>cmd apr1 apr2 apr3 ... apr9</code>	Командная строка с аргументами
<code>\$0 \$1 \$2 \$3 ... \$9</code>	Переменные для аргументов командной строки
<code>shift n</code>	Сдвинуть аргументы командной строки
<code>echo \$#</code>	Вывести количество аргументов командной строки
<code>echo \$*</code>	Вывести все (текущие) аргументы командной строки
<code>exit #</code>	Завершить процедуру и установить ее код завершения равным <code>#</code>
<code>echo \$?</code>	Вывести код завершения последней исполнявшейся команды

Экранирование

<code>\</code>	Отменить специальный смысл следующего за ним символа
<code>'строка'</code>	Отменить специальный смысл всех символов, заключенных в кавычки
<code>"строка"</code>	Отменить специальный смысл всех символов, заключенных в кавычки, кроме <code>\$</code> , <code>"</code> , <code>\</code> и <code>`</code> (знак тупого ударения)

Подстановка команд

<code>\$ (команда)</code> или <code>`команда`</code>	Выполнить команду и подставить в командную строку ее результат
<pre>banner \$(date) dirs=\$(ls -F grep /) X=\$(expr \$X + 1) for name in \$(who cut -f1 -d" ")</pre>	

Фильтры

<code>cut -с список [файл]</code>	Вырезает и выводит на экран указанные позиции
<code>cut -f список -d символ [файл]</code>	Вырезает и выводит на экран указанные поля <code>-d символ</code> – задает <i>символ</i> разделитель полей
Пример: <code>who cut -c12-18</code> <code>cut -f1,6 -d: /etc/passwd</code>	
<code>grep [-inv] модель [файл]</code>	Ищет в <i>файле</i> строки, соответствующие указанной <i>модели</i> <code>-i</code> – игнорировать в модели регистр записи букв <code>-n</code> – выводить номера строк, в которых найдена модель <code>-v</code> – выводить строки, в которых модель НЕ НАЙДЕНА

Пример:

```
grep user /etc/passwd  
who | grep user3
```

more [*файл*]

Выводить содержимое файла поэкранно

Пример:

```
ps -ef | more  
sort funfile | more
```

pr [-#] [-o #] [-h "*текст*"] [*файл*] Форматировать вывод на экран
-# — задает вывод в # колонок
-o# — смещает строки на # позиций от левого края
-h "*текст*" — замещает заголовок по умолчанию
указанным *текстом*

Пример:

```
pr funfile | more
```

sort [-ndtX] [+*поле*] [*файл*] -n — числовая сортировка
-d — словарная сортировка
-tX — использовать X в качестве разделителя полей
+*поле* — базовое поле, по которому выполняется
сортировка (нумерация полей начинается с 0)

Пример:

```
sort names  
sort -nt: +2 /etc/passwd
```

tee [-a] *файл*

Направить вывод в **stdout** и в *файл*
-a — добавлять вывод в конец *файла*

Пример:

```
ls | tee ls.out
```

wc [-cwl] [*файл*]

Подсчитать количество символов, слов и строк в
файле
-c — подсчитать количество символов
-w — подсчитать количество слов
-l — подсчитать количество строк

Мультизадачный режим

команда > **команда.out** &

Запустить **команду** на выполнение в фоновом
режиме
stdin отсоединяется для заданий, выполняющихся в
фоновом режиме

nohup команда > **команда.out** & Защитить **команду**, выполняющуюся в фоновом
режиме, от завершения при окончании сеанса работы
в системе

nice команда

Запустить **команду** на исполнение с пониженным
приоритетом

jobs	Вывести информацию о заданиях, выполняющихся в текущем сеансе работы
ps -ef	Вывести список всех процессов, существующих в системе
echo \$\$	Вывести идентификатор процесса для текущего процесса shell
Ctrl + z	Приостановить задание, выполняющееся в активном режиме
bg %#	Перевести задание с номером # в фоновый режим
fg %#	Перевести задание с номером # в активный режим
kill PID	Завершить задание с идентификатором процесса, равным PID
kill -s ИМЯ_СИГНАЛА PID	Послать процессу с идентификатором PID сигнал ИМЯ_СИГНАЛА
trap команда #	Установить перехватывание сигнала # и выполнить команду , когда этот сигнал будет получен
stty -a	Вывести информацию об установках и характеристиках терминала
Ctrl + c	Послать сигнал прерывания интерактивному процессу (сигнал 2)
Ctrl + \	Послать сигнал о завершении интерактивному процессу (сигнал 3)

Ветвление

if команды then cmdtrue(s) else cmdfalse(s) fi	<i>if КОД ЗАВЕРШЕНИЯ ПОСЛЕДНЕЙ команды – истина, выполнять команды, записанные после then</i> <i>if КОД ЗАВЕРШЕНИЯ ПОСЛЕДНЕЙ команды – ложь, выполнять команды, записанные после else</i>
case \$vara in pat1) cmdsa ;; pat2) cmdsb ;; *) cmds default ;; esac	<i>Сопоставлять значение переменной vara с шаблонами выполнять команды, указанные после найденного шаблона</i>

Циклы

```
while cmd(s)
do
    cmdtrue(s)
done
```

while КОД ЗАВЕРШЕНИЯ ПОСЛЕДНЕЙ команды – истина, выполнять команды, записанные после do

```
until cmd(s)
do
    cmdfalse(s)
done
```

until КОД ЗАВЕРШЕНИЯ ПОСЛЕДНЕЙ команды не станет равным истине, выполнять команды, записанные после do

```
for vara in a b c d e
do
    cmd(s)
done
```

Присваивать переменной vara значение каждого элемента из списка и для каждого элемента списка выполнять команды

Переменные среды POSIX shell'a

#	Количество аргументов, переданных в shell процедуру
*	Список всех аргументов, переданных в shell процедуру
?	Код завершения последней выполнявшейся команды
\$	PID последнего выполняющегося shell'a
COLUMNS	Определяет ширину экрана для режимов редактирования shell'a
EDITOR	Определяет режим редактирования, используемый для стека команд. Связан с set -o vi
ENV	Процедура, выполняемая при запуске нового shell'a. Обычно устанавливается равным .kshrc
FCEDIT	Определяет редактор, вызываемый из стека команд
IFS	Internal Field Separators (внутренний разделитель полей); обычно это пробел, символ табуляции и символ новой строки. Он разделяет команду и аргументы в командной строке и слова в строке, вводимой с помощью read
HISTFILE	Маршрутное имя файла, используемого для хранения предыстории команд. По умолчанию это .sh_history

HISTSIZE	Количество команд, сохраненных в файле предыстории, к которым может обращаться shell. По умолчанию 128
HOME	Входной каталог пользователя. Аргумент по умолчанию команды cd .
LINES	Определяет количество строк окна для вывода.
PATH	Список каталогов, в которых выполняется поиск исполняемых программ.
PS1	Первичное приглашение. По умолчанию \$
PS2	Вторичное приглашение. По умолчанию >
PWD	Текущий рабочий каталог; устанавливается последней выполнявшейся командой cd
OLDPWD	Предыдущий рабочий каталог; устанавливается равным каталогу, в котором выполнялась последняя команда cd . Доступен с помощью cd -
SHELL	Маршрутное имя программы, соответствующей текущему shell'у
TERM	Модель используемого терминала
TMOUТ	Если значение этой переменной больше 0, shell завершится, если в течение указанного времени не будет выполнена никакая команда (т.е. не будет нажата клавиша Return)
TZ	Определяет зону времени, которая используется для вывода даты и времени
VISUAL	Определяет режим редактирования, который используется для стека команд. Связан с set -o vi

Система Linux



Переменные, перечисленные выше, могут быть использованы в среде bash shell системы Linux и в среде Korn shell систем Linux и HP-UX. Некоторые из переменных, перечисленных выше, могут в среде входного shell'а пользователя не иметь значений, устанавливаемых по умолчанию, если только системный администратор не предусмотрит необходимые установки при входе пользователя в систему.

Ответы и решения

2–20. Упражнения: Вход в систему и простейшие действия

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы. Для выполнения некоторых упражнений вам может понадобиться *Справочник HP-UX*.

1. Войдите в систему, используя входное имя и пароль, выданные вам преподавателем. Возникли ли у вас какие-либо трудности?

Ответ:

Некоторые проблемы могут возникнуть, если во время ввода своего входного имени или пароля вы допустили ошибку и попытались ее исправить с помощью клавиши `Backspace`. Следует помнить, что для стирания символов во время входа в систему следует использовать клавишу `#`.

2. Выйдите из системы, используя клавиши `CTRL` + `d` или команду `exit`. Заметили ли вы что-нибудь? Вновь войдите в систему.

Ответ:

3. Какие из приведенных ниже команд синтаксически правильны? Попробуйте выполнить их, чтобы увидеть, что будет выведено на экран – результаты выполнения команды или сообщения об ошибках.

```
$ echo
$ echo hello
$ echohello
$ echo HELLO WORLD
```

4. Только для системы HP-UX

Попробуйте выполнить следующие команды, чтобы увидеть, что именно будет выведено на экран:

```
$ banner
$ banner hello
$ BANNER hello
```

Ответ:

```
$ echo          правильно
$ echo hello     правильно
$ echohello      неправильно
$ echo HELLO WORLD  правильно
```

В команде `echo` можно указать нуль, один или более аргументов. Так как аргументами команды являются просто строки символов, а команда просто отображает свои

аргументы на экране, не имеет значения, в верхнем или нижнем регистре вводятся символы.

Для того чтобы отделить команду от ее аргументов, shell требует наличие пробельных символов (пробела или символа табуляции). Команда, представленная в третьей строке, не будет работать, так как shell вместо того, чтобы выполнить команду **echo**, которой передается аргумент **hello**, пытается выполнить команду **echohello**.

Система HP-UX:

\$ banner	<i>неправильно</i>
\$ banner HELLO	<i>правильно</i>
\$ BANNER hello	<i>неправильно</i>

В команде **banner**, в отличие от команды **echo**, необходимо указать, по крайней мере, один аргумент. Поэтому команда, представленная во второй строке, правильная, так как команда **banner** не обращает внимания на то, в каком регистре – верхнем или нижнем представлен ее аргумент. В третьем примере shell будет искать команду **BANNER**, которая не является корректной командой shell'a. Следует помнить, что shell чувствителен к регистру и поэтому **banner** – не то же самое, что **BANNER**.

Система Linux



В системе Linux версия команды **banner** используется только при выводе на принтер, а не на экран терминала. По этой причине при выполнении данных упражнений использование команды **banner** невозможно.

5. Установите для себя пароль (или измените пароль, если он уже установлен). Не забудьте правила формирования пароля.

Ответ:

```
$ passwd
Changing password for user3
Old password:
New password:
Re-enter new password:
$
```

6. Используя варианты команды **who**, ответьте на следующие вопросы. Какие команды вы использовали?

Кто работает в системе?

С какого терминала вы вошли в систему?

Кто вы для системы (ваше входное имя)?

Ответы и решения

Ответ:

```
$ who
$ who am i
$ whoami
```

7. Могут ли другие пользователи послать сообщение на ваш терминал? С помощью какой команды вы сможете узнать это?

Ответ:

```
$ mesg
```

8. Проверьте, вошел ли в систему ваш коллега, и пошлите сообщение на его терминал. Организуйте диалог с ним. Пошутите друг с другом.

Что случится, если вы попытаетесь послать сообщение своему коллеге, когда он еще не вошел в систему? Что произойдет, если ваш коллега запретил вывод чужих сообщений на свой терминал?

Ответ:

```
$ who                                Подтверждение того, что коллега вошел в систему
$ write входное_имя_коллеги
текст сообщения
текст сообщения
Ctrl + d                             Завершение разговора
```

В системах HP-UX или Linux в случае, если ваш коллега еще не вошел в систему, вы получите сообщение:

```
входное_имя_коллеги is not logged on.
```

Если ваш партнер запретил вывод чужих сообщений на свой экран, вы получите сообщение:

Система HP-UX:

```
Permission denied.
```

Система Linux:



```
write: входное_имя_коллеги has messages disabled
```

9. Прочитайте системные новости. Какую команду вы использовали? Сможете ли вы повторно вывести новости, которые вы уже прочитали?

Ответ (система HP-UX):

```
$ news
this is a news message
$ news
no new news
$ news -a
this is a news message
```

Ответ (система Linux):



В этой системе нет команды **news**.

10. Выполните команду **date** с соответствующими аргументами, так чтобы дата была выведена в формате *mm-dd-yy*. ПОДСКАЗКА: посмотрите примеры, приведенные в статье **date (1)** Справочника HP-UX.

Ответ:

```
$ date +%m-%d-%y
```

11. Используя команду **man**, найдите описание команды **cp**. Что это за команда? Какое минимальное количество аргументов она требует?

Ответ:

Команда **cp** используется для копирования одного или нескольких файлов. Она требует, по крайней мере, два аргумента: имя исходного файла и имя файла-приемника.

12. Используя команду **man**, найдите описание команды **ls**. Что это за команда? Какое минимальное количество аргументов она требует?

Ответ (система HP UX):

Команда **ls** используется для вывода содержимого каталогов. Она не требует обязательных аргументов. Следует обратить внимание на то, что в этой команде доступно много опций. Каждая опция расширяет возможности команды **ls** и каждая опция задается одной буквой.

Система Linux



В системе Linux команда **ls** выполняет те же функции. Она используется для вывода содержимого каталогов и не требует аргументов, хотя при необходимости можно указать произвольное количество аргументов.

13. Выполните команду **ll /usr/bin**. Вы увидите несколько экранов, для просмотра которых потребуется использовать прокрутку.

Только для HP UX:

С помощью клавиши "стрелка вверх" переместите курсор в начало экрана. Выполните команду **clear**. Осталось что-нибудь на экране? Нажмите вместе с клавишей **[Shift]** клавишу "стрелка вниз", чтобы выполнить прокрутку вниз. Появится ли на экране фрагмент результатов команды **ls -l**?

Ответы и решения

Ответ (система HP-UX):

Команда **ll /usr/bin** выведет результат, который занимает несколько экранов. Выполнение команды **clear** после перемещения курсора в начало экрана очищает только последний экран результатов команды. Прокрутка вниз позволит вывести на экран предыдущие фрагменты вывода.

Система Linux



В системе Linux нет команды **ll**. Нужно использовать команду **ls**, в которой явно указана соответствующая опция: **ls -l**.

Кроме того, характеристики клавиш перемещения курсора в средах терминальных окон систем HP-UX/CDE и Linux/KDE отличаются друг от друга. Использование клавиш перемещения курсора так, как это сделано в данном примере, не будет работать в терминальном окне системы Linux.

14. Выйдите из системы. Вновь войдите в систему при включенном режиме CAPS Lock. Как можно исправить эту ситуацию в текущем сеансе работы, т.е. без того чтобы выйти из системы и потом вновь войти в нее? (ПОДСКАЗКА: посмотрите статью **stty** Справочника.)

Ответ (HP-UX):

Обратите внимание на то, что если вы нажмете клавишу **Caps Lock**, это не даст никакого результата. Чтобы отменить действие включенного режима Caps Lock, необходимо использовать команду **stty**:

```
$ STTY -LCASE
```

после чего нажать на клавиатуре клавишу **Caps Lock**. Теперь можно вводить буквы и верхнего, и нижнего регистров. Такой интерфейс предназначен для терминалов, которые поддерживают буквы только верхнего регистра, чтобы они могли правильно интерпретировать команды, которые обычно вводятся в нижнем регистре.

Ответ (Linux):



Linux не разрешает пользователям при входе в систему вводить входное имя заглавными буквами.

3–19. Упражнения: Использование CDE

Цель: Освоить использование основных элементов среды CDE рабочего стола.

Указания

1. Используя менеджер файлов, перейдите в папку **class**. Выберите файл **cde_intro** и скопируйте его в файл **cde_intro2**.

Ответ:

1. Установите курсор мыши на каталог **class** и дважды щелкните.
2. Установите курсор на файл **cde_intro**.
3. Из меню **Selected** выберите **Copy To**. Появится всплывающее диалоговое окно, в котором будет выведена подсказка, предлагающая ввести имя файла-приемника. Введите **cde_intro2**.
4. Нажмите .
2. Переместите файл **cde_intro2** в папку **cde_dir**.

Ответ:

1. Установите курсор на файл **cde_intro2**.
2. Из меню **Selected** выберите **Move To**. Появится всплывающее диалоговое окно, в котором будет выведена подсказка, предлагающая ввести имя папки-приемника. Введите **cde_dir**.
3. Нажмите .
3. Перейдите в папку **cde_dir**. Измените права доступа к файлу **cde_intro2**, так чтобы его можно было только читать.

Ответ:

1. Установите курсор на папке **cde_dir** и дважды щелкните мышью, чтобы перейти в этот каталог.
2. Высветите файл **cde_intro2**.
3. Появится всплывающее диалоговое меню, в котором будут выведены текущие значения прав собственности и прав доступа, а также размер файла и время последней модификации файла.
4. Щелчком мыши *снимите* право записи.
5. Нажмите .

Ответы и решения

4. Вернитесь во входную папку. Используйте менеджер файлов, для того чтобы найти все файлы, которые содержат строку *graphical environment*. Выполняйте поиск в папке **class**.

Ответ:

1. Откройте меню **File** и выберите **Find...**
 2. Введите текст *graphical environment* в поле **File Contents**.
 3. В поле **Search Folder:** введите после имени вашего входного каталога имя **class**.
 4. Щелкните по .
5. С помощью менеджера файлов найдите все файлы, имена которых начинаются символами *data*.

Ответ:

1. Откройте меню **File** и выберите **Find...**
 2. В поле **File or Folder Name:** введите **data***.
 3. В поле **Search Folder:** введите после имени вашего входного каталога имя **class**.
 4. Щелкните по .
6. Можно выполнить поиск по шаблону, если использовать вопросительный знак (?) для определения одного (любого) символа. С помощью менеджера файлов найдите все файлы, имена которых начинаются символами *data*, за которыми следует один (любой) символ.

Ответ:

1. Откройте меню **File** и выберите **Find...**
 2. В поле **File or Folder Name:** введите **data?**.
 3. В поле **Search Folder:** введите после имени вашего входного каталога имя **class**.
 4. Щелкните по .
7. С помощью менеджера файлов из папки **cde_dir** удалите файл **cde_intro2**.

Ответ:

1. Откройте папку **cde_dir**.
2. Высветите файл **cde_intro2**.

3. Выберите элемент **Put in Trash** из меню **Selected** *ИЛИ* нажмите и удерживайте левую кнопку мыши во время перетаскивания иконки файла в корзину. Как только иконка файла окажется на корзине, отпустите кнопку мыши.
8. Восстановите файл **cde_intro2** из корзины.

Ответ:

1. Дважды щелкните на иконке корзины на передней панели, чтобы открыть окно корзины.
2. Выберите для восстановления файл **cde_intro2**.
3. Откройте меню **File** и выберите **Put Back**. Файл будет возвращен в его исходную позицию в файловой системе.
9. С помощью менеджера файлов удалите безвозвратно из папки **cde_dir** файл **cde_intro2**.

Ответ:

1. Откройте папку **cde_dir**.
2. Высветите файл **cde_intro2**.
3. Выберите элемент **Put in Trash** из меню **Selected** *ИЛИ* нажмите и удерживайте левую кнопку мыши во время перетаскивания иконки файла в корзину. Как только иконка файла окажется на корзине, отпустите кнопку мыши.
4. Дважды щелкните на иконке корзины на передней панели, чтобы открыть окно корзины.
5. Выберите файл **cde_intro2**.
6. Откройте меню **File** и выберите **Shred**. Файл будет удален безвозвратно.
10. Используя текстовый редактор, откройте для редактирования файл **\$HOME/class/cde_intro**. Скопируйте первый абзац текста и вставьте его в конец документа.

Ответ:

1. Установите курсор мыши на начало первого абзаца. Нажмите и удерживайте левую кнопку мыши во время обводки курсором мыши текста, который нужно скопировать. Отпустите кнопку мыши.
2. Из меню **Edit** выберите **Copy**. Копия текста запоминается в буфере.
3. Установите курсор в конец файла.
4. Из меню **Edit** выберите **Paste**.
11. Замените все вхождения символов CDE, кроме первого и третьего, текстом *Common Desktop Environment*.

Ответы и решения

Ответ:

1. Из меню **Edit** выберите **Find/Change**
 2. В поле **Find** введите **CDE**.
 3. В поле **Change** введите **Common Desktop Environment**.
 4. Чтобы начать поиск, нажмите или щелкните по .
 5. Если найден текст, соответствующий шаблону поиска, курсор будет установлен на начало этого текста. Чтобы заменить его, щелкните по . Если найденный текст изменять не надо, но следует продолжить поиск, щелкните по . Если нужно заменить все фрагменты текста, удовлетворяющие шаблону поиска, нужно щелкнуть по .
 6. Когда все изменения сделаны, щелкните по .
12. Исправьте в файле все слова с опечатками.

Ответ:

1. Из меню **Edit** выберите **Check Spelling**. Откроется диалоговое окно **Spell**.
 2. В поле **Change To** введите правильное слово.
 3. Щелкните по , чтобы внести единичное исправление, или по , чтобы внести все исправления. Если нужно просто найти слова с опечатками, а исправлять их не нужно, щелкните по .
 4. Когда все исправления внесены, щелкните по .
13. Используя всплывающее меню переключателя рабочих областей, добавьте еще одну область и назовите ее **CDE Class**.

Ответ:

1. Установите курсор мыши на ту часть переключателя рабочих областей, которая не занята другими управляющими элементами и кнопками переключателя, и нажмите правую кнопку мыши. Выберите **Add Workspace**.
 2. Установите курсор мыши на рабочую область с именем **New** и нажмите правую кнопку мыши. Выберите **Rename**.
14. Используя меню дополнительных панелей и всплывающие меню, измените иконку для личных приложений с текстового редактора на терминал.

Ответ:

1. Щелкните по треугольнику (стрелке вверх), который находится над управляющим элементом текстового редактора на передней панели.
2. Установите курсор на иконку терминала и нажмите правую кнопку мыши.

3. Выберите **Copy to Main Panel**.
4. Чтобы закрыть меню дополнительной панели, щелкните по треугольнику (стрелка вниз) над управляющим элементом терминала на передней панели.
15. Используйте **Man Page Viewer** из менеджера приложений, чтобы получить статью *Справочника HP-UX* для команды **ls**.

Ответ:

1. Откройте менеджер приложений.
2. Дважды щелкните по иконке **Desktop_Apps group**, чтобы вывести ее содержимое.
3. Прокручивайте окно вниз, пока не увидите иконку активизации **Man Page Viewer**.
4. Для запуска приложения дважды щелкните по иконке активизации.
5. Чтобы получить нужный текст, введите в диалоговом окне текст **ls**.
16. Выберите вашего коллегу, которому можно будет отправить почтовое сообщение. Если вы работаете в разных системах, вы должны знать входное имя вашего коллеги и хост имя системы, в которой он работает. Эта информация необходима для задания адреса email вашего коллеги в формате *имя_пользователя@хост_имя*.

Пошлите вашему коллеге сообщение и попросите его прислать сообщение вам. (Если вы испытываете какие-либо затруднения в поиске коллеги, вы можете послать сообщение самому себе.)

Ответ:

1. Из меню **Compose** выберите **New Message**.
2. В поле **To**: введите email адрес вашего коллеги, а в поле **Subject** – тему.
3. После того, как вы ввели адрес, нажмите **Return**, чтобы перейти в область текста и ввести само сообщение.
4. Щелкните по кнопке **Send**.
17. Когда ваш коллега пришлет вам сообщение, ответьте на него и отошлите исходное сообщение третьему коллеге.

Ответ:

Для того чтобы ответить на сообщение:

1. Выберите сообщение, на которое надо ответить.
2. Из меню **Compose** выберите **Reply to Sender**.
3. Введите ответ.

Ответы и решения

- Щелкните по **Send**.

Чтобы перенаправить сообщение:

- Выберите сообщение, которое надо перенаправить.
 - Из меню **Compose** выберите **Forward Message**.
 - В поле **To**: введите email адрес получателя сообщения.
 - При желании, введите дополнительно свой комментарий.
 - Щелкните по **Send**.
18. Используя текстовый редактор, создайте шаблон, который вы будете использовать для ежемесячной отправки вашему руководителю отчета о положении дел. Сохраните этот шаблон в файле **status**. В почтовом сервисе создайте шаблон с именем **monthly**, содержащий этот вновь созданный файл. Используйте этот шаблон для отправления отчета о положении дел вашему коллеге.

Ответ:

Ранее вы создавали файлы с помощью текстового редактора; чтобы создать шаблон, сделайте следующее:

- В почтовом сервисе щелкните по **Mail Options** из линейки меню **Mailbox**.
- Щелкните по кнопке **Category** и выберите **Templates**. Появится диалоговое окно **Template**.
- В поле **Menu Label** введите название **monthly**.
- В поле **File/Path**: введите имя файла **status**.
- Чтобы включить шаблон в список шаблонов, щелкните по **Add**.

Чтобы использовать шаблон:

- Из меню **Compose** выберите **New Message**.
 - Из меню **Format** выберите **Templates**.
 - Из доступного списка выберите имя нужного шаблона.
19. Используйте функции календаря для создания пяти или шести назначений встреч в текущем месяце. Вы можете назначить единичную встречу или регулярные встречи, например, ежемесячные.

Ответ:

- Откройте календарь в режиме вывода текущего месяца. Щелкните по тому дню, когда вы хотите назначить встречу.
- Из меню **Edit** выберите **Appointment...**, чтобы активизировать **Calendar Appointment Editor**.

3. Выберите время начала (**Start**) и окончания (**End**) встречи.
 4. Определите, что это за встреча.
 5. Если вы хотите определить эту встречу как регулярную, щелкните по .
20. Установите настройки календаря так, чтобы вы могли по сети видеть ваш календарь и календарь вашего коллеги, добавив оба эти календаря в список просмотра (**Browse List**).

Ответ:

1. Из меню **Browse** выберите **Menu Editor**.
 2. В поле **User Name** введите *имя_календаря@хост_имя*.
 3. Щелкните по .
 4. Щелкните по .
 5. Выполните указанные действия для своего календаря и для календаря своего коллеги.
21. Вы хотите запланировать важную встречу с вашим коллегой, но сначала хотите проверить его календарь, возможна ли эта встреча. Просмотрите календари таким образом, чтобы можно было видеть одновременно и свой календарь, и календарь вашего коллеги.

Ответ:

1. Из меню **Browse** выберите **Compare Calendars**.
 2. Выберите имя календаря, который вы хотите просмотреть.
Календари при выводе накладываются один поверх другого. Занятое время выводится затемненным, незанятое – незатемненным.
22. Предоставьте вашему коллеге права доступа **Insert** и **Change** к вашему календарю, так чтобы он мог планировать встречи с вами, когда это необходимо.

Ответ:

1. Из меню **File** выберите **Options...**
2. Из меню **Category** выберите **Access List**, чтобы открыть диалоговое окно **Access List and Permissions** (Список доступа и права доступа).
3. В поле **User Name** введите *имя_календаря@хост_имя* для того календаря, которому вы хотите предоставить права доступа.
4. Выберите права **View**, **Insert** и **Change**.
5. Щелкните по , чтобы добавить календарь в список доступа с теми правами, которые вы выбрали.

Ответы и решения

6. Щелкните по **Apply**.
 7. Выполните для себя эти операции, чтобы вы могли перекрывать свой календарь и календарь своего коллеги.
23. Запланируйте встречу с вашим коллегой и отправьте ему по почте напоминание о ней.

Ответ:

1. Просмотрите свое расписание и расписание своего коллеги.
2. Щелкните по незатененной области, доступной в обоих календарях.
3. Щелкните по **Schedule**. Откроется окно **Calendar Group Appointment Editor**. Буква **Y** в колонке **Access** означает, что вы имеете право **Insert**, чтобы изменить планы встреч. Буква **N** означает, что у вас такого права нет. Если у вас нет права **Insert**, напомните вашему коллеге, чтобы он предоставил вам это право.

4-14. УПРАЖНЕНИЯ: Файловая система

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Каково имя вашего *HOME* каталога?

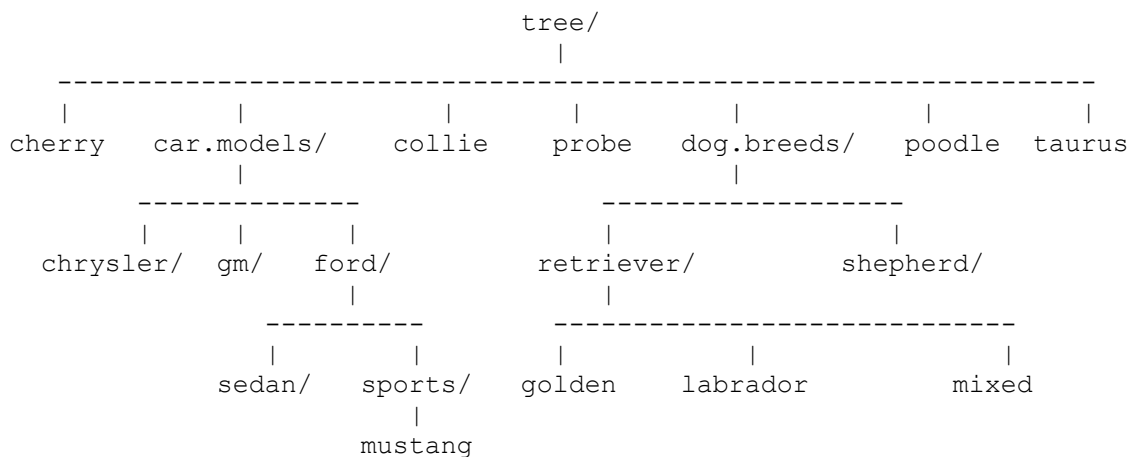
Ответ:

Войдите в систему и затем выполните команду **pwd**. На экран будет выведено что-то вроде следующего: **/home/ВАШЕ_ВХОДНОЕ_ИМЯ**.

2. Находясь в *HOME* каталоге и используя команду **ls**, определите полную древовидную структуру каталогов, начиная с каталога **tree**. Представьте графически структуру дерева каталогов, изобразив каталоги кружками. В случае необходимости, используйте отдельную страницу.

Ответ:

Упражнение требует исполнения многочисленных команд **ls** (**lsf**). Или, в качестве другого способа, можно использовать в команде рекурсивную опцию **-R**. Структура каталогов может иметь следующий вид:



3. Каково абсолютное маршрутное имя файла **labrador** в дереве каталога, полученного в предыдущем упражнении? Каково его относительное маршрутное имя по отношению к вашему *HOME* каталогу?

Ответ:

Абсолютное маршрутное имя:

/home/ВАШЕ_ВХОДНОЕ_ИМЯ/tree/dog.breeds/retriever/labrador

Относительное маршрутное имя:

tree/dog.breeds/retriever/labrador

Ответы и решения

4. Перейдите из *HOME* каталога в каталог **retriever**. Используя относительное маршрутное имя, перейдите в каталог **shepherd**. Используя опять же относительное маршрутное имя, перейдите в каталог **car.models**. Наконец, вернитесь в *HOME* каталог. Какие команды вы использовали? Каким образом вы узнавали, что находитесь в нужном каталоге?

Ответ:

```
$ cd
$ cd tree/dog.breeds/retriever
$ cd ../shepherd
$ cd ../../car.models
$ cd
```

Чтобы проверить правильность перехода в каталог, следует использовать команду

```
$ pwd
```

5. Создайте каталог **junk** в своем *HOME* каталоге. Сделайте этот каталог текущим. Какие команды вы использовали? Каково абсолютное маршрутное имя нового каталога?

Ответ:

```
$ cd
$ mkdir junk
$ cd junk
$ pwd
/home/ваше_входное_имя/junk
```

6. Находясь в *HOME* каталоге, с помощью одной команды создайте следующие каталоги:

```
junk/dirA/dir1
junk/dirA
junk/dirA/dir2
junk/dirA/dir1/dirc
```

Возникли ли у вас какие-нибудь проблемы? Если возникли, прежде чем выполнять упражнение снова, удалите каталоги, созданные в результате неправильных действий. Какую команду вы использовали?

Ответ:

```
$ mkdir junk/dirA junk/dirA/dir1 junk/dirA/dir2 junk/dirA/dir1/dirc
```

или

```
$ mkdir -p junk/dirA/dir1/dirc junk/dirA/dir2
```

Если имена каталогов вводились в том порядке, в каком они представлены в условиях данного упражнения, возникала ошибка при выполнении команды, так как команда обрабатывает свои аргументы в порядке слева направо.

7. Находясь в *HOME* каталоге, выведите содержимое каталога **dirA**, находящегося в каталоге **junk**. Используйте и относительное, и абсолютное маршрутные имена. Какие команды вы использовали?

Ответ:

```
$ ls junk/dirA
$ ls /home/ваше_входное_имя/junk/dirA
```

8. Находясь в *HOME* каталоге и используя только команду **rmdir**, удалите все подкаталоги, находящиеся в каталоге **junk**. Как можно выполнить это упражнение, используя единственную команду **rmdir**?

Ответ:

```
$ rmdir junk/dirA/dir1/dirC
$ rmdir junk/dirA/dir1
$ rmdir junk/dirA/dir2
$ rmdir junk/dirA

$ rmdir junk/dirA/dir1/dirC junk/dirA/dir1 junk/dirA/dir2 junk/dirA
```

9. Вернитесь в *HOME* каталог. С помощью единственной команды выведите подробные характеристики файлов **cp** и **vi** (из каталога **/usr/bin**). Попробуйте использовать и абсолютное, и относительное маршрутное имя.

Ответ:

```
$ cd
$ pwd
/home/ВАШЕ_ВХОДНОЕ_ИМЯ
$ ls -l /bin/cp /bin/vi          Абсолютные маршрутные имена
$ ls -l ../../bin/cp ../../bin/vi  Относительные маршрутные имена
```

5-16. УПРАЖНЕНИЯ: Работа с файлами и каталогами

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

Работа с файлами

1. Находясь в *HOME* каталоге, используйте команду **cat** для вывода на экран содержимого файла **funfile**. Что произошло? Какая альтернативная команда позволяет управлять выводом на экран содержимого файла?

Ответ:

```
$ cat funfile
```

Это слишком большой файл, и его содержимое не умещается на одном экране. Команда **more** позволяет управлять выводом информации на экран. Например:

```
$ more funfile
```

2. Используйте команду **more** для вывода на экран содержимого каталога **tree**. Что произошло? Какую команду нужно использовать для вывода на экран содержимого каталога?

Ответ:

```
$ more tree
***** tree is a directory *****
```

Команде **more** известно, что **tree** – это каталог, а не обычный текстовый файл, поэтому его содержимое нельзя вывести на экран в читабельном виде. Для вывода на экран содержимого каталога следует использовать команду **ls**. Например:

```
$ ls tree
```

3. Используйте команду **more** для вывода на экран файла **/usr/bin/ls**. Что произошло? Выведите на экран содержимое файла **/usr/bin/ls** с помощью команды **cat**. Что произошло?

Ответ:

```
$ more /usr/bin/ls
***** /usr/bin/ls: Not a text file *****
```

Ответ (система Linux):

```
$ more /bin/ls
```

Команда **more** знает, что **/usr/bin/ls** – это файл, содержащий скомпилированную программу, а не обычный текстовый файл, поэтому его содержимое не может быть выведено на экран в читабельном виде.

```
$ cat /usr/bin/ls
```

В результате выполнения команды **cat** на экране появится непонятная информация ("мусор"). Такой результат является следствием того, что команда **cat** выводит на экран двоичную (скомпилированную) программу. Это может привести к изменению установок терминала.

Чтобы восстановить терминал или окно терминала в **HP**, нужно:

- Нажать клавишу **Break**.
- Одновременно нажать **Shift** + **Ctrl** + **Reset**.
- Нажать **Return**, чтобы получить приглашение shell'a.
- В ответ на приглашение ввести команду:

```
$ tset -e -k
```

-e: установить стирание символа на ^H,
-k: установить удаление строки на ^X

```
$ tabs
```

Чтобы восстановить терминал или окно терминала в **Linux**, нужно:



В ответ на приглашение ввести команду:

```
$ tset -e -k
```

-e: установить стирание символа на Backspace,
-k: установить удаление строки на ^U

4. Вернитесь в *HOME* каталог. Скопируйте файл **names** в файл **names.cp**. Выведите содержимое обоих файлов, чтобы убедиться в их идентичности.

Ответ:

```
$ cp names names.cp
$ cat names names.cp
```

5. Если файл **names** будет изменен, повлияет это на файл **names.cp**? Модифицируйте файл **names**, скопировав в него файл **funfile**. Что произошло с файлами **names** и **names.cp**?

Ответ:

Файлы **names** и **names.cp** – два разных объекта. Содержимое файла **names** будет заменено содержимым файла **funfile**. На файл **names.cp** это никак не повлияет.

```
$ cp funfile names
$ more names names.cp
```

В файле **names** теперь то же содержимое, что и в файле **funfile**, тогда как в файле **names.cp** сохранилось то, что было ранее в файле **names**.

6. Как можно восстановить исходное содержимое файла **names**? Выполните команду для восстановления файла **names**.

Ответ:

Чтобы восстановить исходное содержимое файла **names**, нужно скопировать в него файл **names.cp** или переименовать файл **names.cp** в **names**.

```
$ cp names.cp names
или
$ mv names.cp names
```

Ответы и решения

7. Создайте еще одну копию файла **names** с именем **names.new**. Измените имя **names.new** на **names.orig**.

Ответ:

```
$ cp names names.new
$ mv names.new names.orig
```

8. Как создать два файла (с именами **names.2nd** и **names.3rd**), которые ссылаются на содержимое файла **names**?

Ответ:

```
$ ln names names.2nd
$ ln names names.3rd      или  $ln names.2nd names.3rd
```

9. Если вы изменили содержимое файла **names**, повлияет ли это на содержимое файлов **names.2nd** и **names.3rd**? Скопируйте файл **funfile** в файл **names** и выведите характеристики всех файлов, имена которых начинаются с **names**, в длинном формате. Изменилось что-нибудь для файла **names.orig**? **names.2nd**? **names.3rd**?

Ответ:

Файлы **names**, **names.2nd** и **names.3rd** ссылаются на одни и те же данные, размещенные на диске. Если изменяется какой-то один из этих трех файлов, изменятся все три файла. Из выведенных характеристик файлов видно, что у этих трех файлов количество ссылок равно трем, так как существуют три имени, ссылающиеся на одни и те же данные. Файл **names.orig** представляет собой отдельный объект, на что указывает то, что количество ссылок этого файла осталось равным единице.

```
$ cp funfile names
$ ls -l names.orig names names.2nd names.3rd
-rw-r--r-- 1 user3 class 37 Jul 24 11:06 names.orig
-rw-r--r-- 3 user3 class 125 Jul 24 11:08 names
-rw-r--r-- 3 user3 class 125 Jul 24 11:10 names.2nd
-rw-r--r-- 3 user3 class 125 Jul 24 11:12 names.3rd
```

Если выполнить команду **ls -i** для всех файлов, имена которых начинаются с **names**, будут выведены номера описателей (**inode**) для этих файлов. В описателе файла хранятся все характеристики файла, такие как права доступа, количество ссылок на файл, права собственности для файла и другие. Файлы, связанные друг с другом, имеют один и тот же номер описателя файла.

```
$ ls -i names.orig names names.2nd names.3rd
102 names.orig
322 names
322 names.2nd
322 names.3rd
```

10. Удалите файл **names**. Что случится с файлами **names.2nd** и **names.3rd**?

Ответ:

```
$ rm names
```

Файлы **names.2nd** и **names.3rd** не будут удалены, но количество ссылок у этих файлов уменьшится на единицу, что можно увидеть, выполнив команду **ls -l**:

```
$ ls -l names.orig names names.2nd names.3rd
names not found
-rw-r--r-- 1 user3 class 37 Jul 24 11:06 names.orig
-rw-r--r-- 2 user3 class 125 Jul 24 11:10 names.2nd
-rw-r--r-- 2 user3 class 125 Jul 24 11:12 names.3rd
```

11. Используйте интерактивную опцию команды **rm** для удаления файлов **names.2nd** и **names.3rd**.

Ответ:

```
$ rm -i names.2nd names.3rd
names.2nd? y
names.3rd? y
$
```

12. Снова скопируйте файл **names.orig** в файл **names**.

Ответ:

```
$ cp names.orig names
```

Работа с каталогами

1. Создайте в своем *HOME* каталоге каталог **fruit**. Используя только одну команду, переместите перечисленные ниже файлы из *HOME* каталога в каталог **fruit**:

```
lime
grape
orange
```

Ответ:

```
$ cd
$ mkdir fruit
$ mv lime grape orange fruit
```

2. Переместите указанные ниже файлы из *HOME* каталога в каталог **fruit**. В каталоге **fruit** перемещенные файлы должны получить другие имена, также указанные ниже:

Источник Приемник

```
apple    APPLE
peach    Peach
```

Ответ:

```
$ cd
$ mv apple fruit/APPLE
$ mv peach fruit/Peach
$
```

3. Проанализируйте структуру каталога **tree**, находящегося в вашем *HOME* каталоге. Каталог **tree** требует некоторой реорганизации.

Переместите файлы **collie** и **poodle**, чтобы они оказались в каталоге **dog.breeds**.

Переместите файл **probe** в каталог **sports**.

Ответы и решения

Переместите файл **taurus** в каталог **sedan**.

В каталоге **tree** создайте новый каталог **horses**.

Скопируйте файл **mustang** в созданный каталог **horses**.

Переместите файл **cherry** в каталог **fruit**, созданный в предыдущем упражнении.

ПОДСКАЗКА: Все эти изменения нужно выполнять, находясь в некотором каталоге, но в каком именно каталоге, на ваш взгляд, вы должны находиться?

Ответ:

```
$ cd
$ cd tree
$ pwd
/home/ваше_входное_имя/tree
$ mv collie poodle dog.breeds
$ mv probe car.models/ford/sports
$ mv taurus car.models/ford/sedan
$ mkdir horses
$ cp car.models/ford/sports/mustang horses
$ mv cherry ../fruit
```

4. Переместите каталог **fruit** из *HOME* каталога в каталог **tree**.

Ответ:

```
$ cd
$ mv fruit tree
```

В каталоге **tree** создается каталог **fruit**.

5. Сделайте каталог **fruit** текущим рабочим каталогом. Переместите файлы **banana** и **lemon** в каталог **fruit**. ПОДСКАЗКА: Помните, что имя "точка-точка" (..) представляет родительский каталог, а имя "точка" (.) представляет текущий каталог.

Ответ:

```
$ cd
$ cd tree/fruit
$ mv ../../banana ../../lemon .
```

"Охотник за мусором" (Дополнительно)

1. В *HOME* каталоге в файле **scaveng.README** вы найдете первый ключ к разгадке для охотника за мусором. Охотник будет использовать файлы и каталоги, размещенные в каталоге **scavenger**, который находится в *HOME* каталоге. В каталоге **scavenger** размещены подкаталоги **north**, **south**, **east** и **west**. В каждом из этих подкаталогов, в свою очередь, находятся подкаталоги **1_mile**, **2_mile**, **3_mile**. Ключи к разгадке (подсказки) в каждом из этих подкаталогов позволяют найти зашифрованное секретное слово. Например, если подсказка выглядит как "иди на восток на три мили", это значит, что нужно выбрать подкаталог **east/3_mile**, в котором нужно прочитать содержимое файла **README**. В этом файле находится следующая подсказка. Движение по системе подсказок продолжается до тех пор, пока не будет найдено зашифрованное секретное слово. Удачи!

Ответ:

```
$ cd
$ cd scavenger
$ more scaveng.README
north, 1 mile
$ cd north/1_mile
$ more README
east, 2 miles
$ cd ../../east/2_mile
$ more README
You are on the right track!
south, 3 miles
$ cd ../../south/3_mile
$ more README
You have to keep going south, 2 miles
$ cd ../2_mile
$ more README
You are almost there west 1 mile
$ cd ../../west/1_mile
$ more README
CONGRATS You have found the end of the trail.
The code word is _____
```

Печать файлов – HP-UX

1. Выведите текущее состояние принтеров в системе спулинга lp spooler и найдите имя принтера по умолчанию.

Ответ:

```
$ lpstat -t
scheduler is running
system default destination: rw
device for rw: /dev/rw
rw accepting requests since Jul 1 10:56:20 1994
printer rw is idle.  enabled since Jul 4 14:32:52 1994
fence priority : 0
```

2. Отправьте файл **funfile** на строчный принтер. Запишите идентификатор запроса, который будет выведен на терминал.

Ответ:

```
$ lp funfile
request id is rw-58 (1 file)
```

3. Проверьте, что ваш запрос поставлен в очередь к принтеру.

Ответ:

```
$ lpstat
rw-58 ralph 3967 Jul 4 16:57:25 1994
rw-59 ralph 1331 Jul 6 13:01:19 1994
```

4. Как можно определить, какие файлы других пользователей печатаются? Попробуйте узнать это.

Ответы и решения

Ответ:

Это можно узнать, выполнив команду **lpstat -t**.

- Используйте команду **cancel**, чтобы удалить свой запрос из очереди к принтеру. Убедитесь в том, что запрос удален из очереди.

Ответ:

```
$ cancel rw-58 rw-59
request "rw-58" canceled
request "rw-59" canceled
$ lpstat
$
```

Печать файлов – Linux



- Выведите текущее состояние очереди к принтеру в системе спулинга принтера.

Ответ:

```
$ lpq
Rank   Owner    Job   Files                Total Size
1st    user1    46    /etc/passwd         1904 bytes
2nd    alun     47    names                15 bytes
```

- Отправьте файл **funfile** на строчный принтер.

Ответ:

```
$ lpr funfile
```

- Убедитесь в том, что ваш запрос поставлен в очередь к принтеру.

Ответ:

```
$ lpq
Rank   Owner    Job   Files                Total Size
1st    user1    46    /etc/passwd         1904 bytes
2nd    alun     47    names                15 bytes
3rd    user1    48    funfile              816 bytes
```

- Как можно определить, какие файлы других пользователей печатаются? Попробуйте сделать это.

Ответ:

Это можно узнать, выполнив команду **lpq** и просмотрев имена файлов в колонке **Files**.

- Используйте команду **lprm**, чтобы удалить свой запрос из очереди к принтеру. Убедитесь в том, что запрос удален из очереди.

Ответ:

```
$ lprm 46 48
    dfA045ABQzluw dequeued
    cfA045ABQzluw dequeued
    dfA046ABQzluw dequeued
    cfA046ABQzluw dequeued
$
$ lpq
    Rank    Owner      Job   Files      Total Size
    1st     alun       47    names      15 bytes
$
```

6-14. УПРАЖНЕНИЯ: Доступ к файлам

Указания

Упражнения разделены на четыре раздела. Выполните команды, необходимые для решения представленных упражнений, и ответьте на поставленные вопросы. Возможно, будет недостаточно времени для выполнения всех упражнений.

Права доступа к файлам

1. Найдите в своем *HOME* каталоге файл **mod5.1**. Кто имеет права доступа к этому файлу? Какие? Можете ли вы вывести на экран содержимое файла **mod5.1**?

Ответ:

```
$ ls -l
-rw-r--r-- 1 ваше_входное_имя class      20 Jan 24 13:13 mod5.1
```

Пользователь с именем *ваше_входное_имя* имеет право чтения и записи.

Члены группы **class** имеют право чтения.

Все остальные пользователи также имеют право чтения.

```
$ cat mod5.1
```

Эта команда выполнится успешно, так как вы имеете право чтения этого файла.

2. Установите для файла **mod5.1** следующие права доступа: **-w-----**. Сможете ли вы теперь вывести на экран содержимое файла **mod5.1**?

Ответ:

```
$ chmod a-rwx,u=w mod5.1
$ cat mod5.1
```

Теперь вы не имеете право чтения файла **mod5.1**, поэтому команда **cat** выдаст сообщение об ошибке.

3. Установите для файла **mod5.1** следующие права доступа: **rw-----**. Сможете ли вы теперь вывести на экран содержимое файла **mod5.1**? Сможет ли другой пользователь – член вашей группы вывести на экран содержимое файла **mod5.1**?

Ответ:

```
$ chmod u=rw mod5.1
```

Вы можете вывести содержимое файла **mod5.1**.

Другой пользователь не сможет вывести содержимое файла **mod5.1**.

4. Какие права доступа к файлу **mod5.1** нужно установить, чтобы другой пользователь – член вашей группы *смог* прочитать содержимое файла?

Ответ:

```
$ chmod g+r mod5.1
```

Разрешается чтение файла всем членам группы **class**, а ваш коллега также является членом группы **class**. Следовательно, вы устанавливаете для группы право чтения файла.

5. Создайте копию файла **mod5.1** с именем **mod5.2**. Удалите для файла **mod5.2** право записи. Сможете ли вы удалить этот файл? Что нужно сделать, чтобы защитить этот файл от удаления?

Ответ:

```
$ cp mod5.1 mod5.2
$ chmod -w mod5.2
$ rm mod5.2
mod5.2: 444 mode ? (y/n)
```

Файл **mod5.2** будет удален!

Вам также нужно удалить право записи для вашего *HOME* каталога. Если вы удалите право записи для своего *HOME* каталога и затем попытаетесь удалить файл, вы получите сообщение "**permission denied**".

6. Кто является владельцем файла **root_file**, находящегося в вашем *HOME* каталоге? С какой группой ассоциирован этот файл? Кто имеет право изменить для этого файла владельца или группу? Какими правами доступа к этому файлу обладаете вы?

Ответ:

Владельцем файла является **root**. Группа файла – **other**. Только суперпользователь (администратор системы) может изменить для этого файла владельца или группу. Вы имеете только право чтения этого файла.

7. Если вы хотите внести изменения в содержимое файла **root_file**, что вы для этого сделаете?

Ответ:

Так как вы имеете право чтения файла, вы можете сделать копию файла **root_file**. Владелец копии будете вы, и поэтому вы сможете изменить содержимое копии файла.

```
$ cp root_file my_root_file
$ ls -l my_root_file
-rw-r--r-- 1 user3 class 3967 Jan 24 13:13 my_root_file
```

8. Выполните команду **mesg y**. Теперь выполните команду **tty** и посмотрите, какой специальный файл устройства ассоциирован с вашим терминалом. Каковы права доступа к этому файлу? Кто является владельцем этого файла? Выполните команду **mesg n**. Какие теперь права доступа к этому файлу? Что в действительности делает команда **mesg**?

Ответ:

```
$ who am I
user3 tty03 Jul 9 11:10
$ mesg y
$ ll /dev/tty03
crw--w--w- 1 user3 class      0 Jan 24 13:13 /dev/tty03
$ mesg n
crw----- 1 user3 class      0 Jan 24 13:13 /dev/tty03
```

Ответы и решения

Вы являетесь владельцем специального файла устройства, ассоциированного с вашим управляющим терминалом, с которого вы вошли в систему. На самом деле команда **mesg** просто выполняет команду **chmod** для файла устройства, ассоциированного с вашим терминалом, чтобы предоставить другим или удалить право записи на ваш терминал.

Права доступа к каталогам

1. В своем *HOME* каталоге создайте новый каталог **mod5.dir**. Скопируйте в каталог **mod5.dir** файл **mod5.1**. Выведите содержимое нового каталога. Какие права доступа установлены для каталога **mod5.dir**? (ПОДСКАЗКА: **ls -ld mod5.dir**)

Ответ:

```
$ cd
$ mkdir mod5.dir
$ cp mod5.1 mod5.dir
$ ls mod5.dir
mod5.1
$ ls -ld mod5.dir
drwxrwxrwx 3 YOUR_LOGNAME class      1024 Jul 24 13:13 mod5.dir
$
```

2. Установите для каталога **mod5.dir** следующие права доступа: **rw-----**. Сможете ли вы перейти в каталог **mod5.dir**? Сможете ли вы вывести содержимое каталога **mod5.dir**? Сможете ли вы получить доступ к файлу **mod5.1**, находящемуся в каталоге **mod5.dir**?

Ответ:

```
$ chmod a-rwx,u+rw mod5.dir
$ cd mod5.dir
sh: mod5.dir: Permission denied
$ ls mod5.dir
mod5.1
$ ls -l mod5.dir/
mod5.dir/mod5.1 not found
total 0
$ cat mod5.dir/mod5.1
cat: cannot open mod5.dir/mod5.1: Permission denied
$
```

3. Установите для каталога **mod5.dir** следующие права доступа: **-wx-----**. Сможете ли вы вывести содержимое каталога **mod5.dir**? Сможете ли вы вывести содержимое файла **mod5.1**, находящегося в каталоге **mod5.dir**? Сможете ли вы перейти в каталог **mod5.dir**?

Ответ:

```
$ chmod u=wx mod5.dir
$ ls mod5.dir
mod5.dir unreadable
$ cat mod5.dir/mod5.1
This is the contents of mod5.1
$ cd mod5.dir
```

команда cd выполнена успешно!

```
$ pwd
/home/user3/mod5.dir
$ ls
. unreadable
```

4. Сможет ли другой пользователь скопировать файл в ваш *HOME* каталог? Как вывести права доступа, установленные для вашего *HOME* каталога?

Ответ:

```
$ cd
$ ls -ld .
drwxr-xr-x 3 ваше_входное_имя class 1024 Jul 24 13:13 .
```

Другие пользователи могут выводить содержимое вашего *HOME* каталога и переходить в ваш *HOME* каталог, но они не могут модифицировать его содержимое. Поэтому другие пользователи не смогут копировать файлы в ваш *HOME* каталог.

5. Скопируйте файл **mod5.1** из своего *HOME* каталога в каталог **/usr/bin**. Возникли ли какие-либо проблемы? Какие права доступа установлены для каталога **/usr/bin**?

Ответ:

```
$ ls -ld /usr/bin
dr-xr-xr-x 3 bin other 1024 Jul 24 13:13 /usr/bin
```

Для каталога **/usr/bin** право записи для других (others) пользователей не установлено, поэтому команда копирования не будет выполнена.

6. Сможете ли вы скопировать файл **/usr/bin/date** в свой *HOME* каталог?

Ответ:

```
$ cd
$ ls -l /usr/bin/date
-r-xr-xr-x 1 bin bin 16384 Nov 15 13:13 /usr/bin/date
$ cp /usr/bin/date .
```

Так как файл **/usr/bin/date** имеет право чтения для других (others) пользователей, вы сможете скопировать этот файл.

Изменение для файла владельца и группы



Пользователям системы Linux не разрешается использовать команду **chown**. Поэтому упражнения со второго по четвертый пользователям системы Linux выполнять не следует.

1. В вашем *HOME* каталоге найдите файл, имя которого совпадает с вашим входным именем. Какие права доступа к этому файлу вы имеете? К какой группе принадлежит ваш коллега? Какие права доступа к этому файлу имеет ваш коллега?

Ответ:

```
$ ls -l ваше_входное_имя
```

Ответы и решения

```
-rw----- 1 ваше_входное_имя class      3967 Jan 24 13:13 ваше_входное_имя
```

Вы имеете права чтения и записи.

Ваш коллега принадлежит к вашей же группе **class**.

Ваш коллега не имеет никаких прав доступа к этому файлу.

2. Продолжая работать с этим файлом, передайте право владения этим файлом вашему коллеге. Есть ли у вас теперь какие-либо права доступа к этому файлу? Попробуйте сделать копию этого файла. Сможете ли вы вернуть себе право владения этим файлом?

Ответ (HP-UX):

```
$ ls -l ваше_входное_имя
```

```
-rw----- 1 ваше_входное_имя class      3967 Jan 24 13:13 ваше_входное_имя
```

Сначала вы имеете право чтения и записи.

```
$ chown входное_имя_коллеги ваше_входное_имя
```

```
$ ls -l ваше_входное_имя
```

```
-rw----- 1 вх_имя_коллеги class 3967 Jan 24 13:13 ваше_входное_имя
```

- Теперь вы не имеете никаких прав доступа к этому файлу.
- Чтобы скопировать этот файл, вам надо иметь, по крайней мере, право чтения, поэтому с текущими правами доступа вы не сможете скопировать этот файл.
- Вы являетесь членом группы **class**, но для данной группы никакие права доступа не установлены.

Единственный способ вернуть себе право владения этим файлом – ваш коллега должен выполнить команду **chown**, чтобы вернуть вам право владения файлом. (У вас хорошие отношения с вашим коллегой?) Вы можете выполнить команду **su**, чтобы продолжить сеанс работы от имени вашего коллеги (если ваш коллега сообщил вам свой пароль для входа в систему), и в этом новом статусе выполнить команду **chown**, чтобы вернуть себе право владения этим файлом.

3. Создайте копию файла **mod5.1** с именем **mod5.3**. Удалите для файла **mod5.3** все права доступа. Сможете ли вы передать права владения этим файлом своему коллеге?

Ответ (HP-UX):

```
$ cp mod5.1 mod5.3
```

```
$ chmod a-rwx mod5.3
```

```
$ chown partner mod5.3
```

Вы можете изменить права владения файлом, так как права доступа к файлу ассоциируются с доступом к содержимому файла, а не с идентификаторами пользователя – владельца файла и группы, назначенными для этого файла.

4. Создайте копию файла **mod5.1** с именем **mod5.4**. Установите для этого файла следующие права доступа: **rw-r-----**. Измените группу, ассоциированную с этим файлом, на **class2**. Передайте права владения этим файлом пользователю **root**. Сможете ли вы вывести содержимое файла **mod5.4**?

Ответ (HP-UX):

```
$ cp mod5.1 mod5.4
```

```
$ chmod a-rwx,u+w,ug+r mod5.4
```

```
$ chgrp class2 mod5.4
```

```
$ chown root mod5.4
$ cat mod5.4cat: cannot open mod5.4: Permission denied
```

Так как вы не являетесь членом группы **class2**, вы не можете получить доступ к файлу **mod5.4**.

5. Перейдите в группу **class2**. Сможете ли вы теперь вывести содержимое файла **mod5.4**? Восстановите свою исходную группу. (ПОДСКАЗКА: используйте команду **id**, чтобы вывести свои идентификаторы пользователя и группы.)

Ответ:

```
$ newgrp class2
$ cat mod5.4
This is the contents of mod5.1
$ newgrp
```

После того, как вы изменили свою действующую группу на **class2**, вы имеете право доступа к файлу **mod5.4**.

Права доступа для вновь создаваемых файлов

1. Какие права доступа устанавливаются для вновь создаваемых файлов? ПОДСКАЗКА: создайте новый файл с помощью редактора или команды **touch** или же скопируйте какой-либо существующий файл. Проверьте, какие права доступа установлены для вновь созданного файла. Что можно сказать относительно прав доступа, устанавливаемых при создании нового каталога? Какова текущая маска, используемая для установки прав доступа при создании новых файлов?

Ответ:

```
$ touch new_file
$ ls -l new_file
-rw-rw-rw- 1 ваше_входное_имя class      0 Jul 24 13:13 new_file
$ mkdir new_dir
$ ls -ld new_dir
drw-r---r-- 3 ваше_входное_имя class 1024 Jul 24 13:13 new_dir
$ umask
000
```

2. Как нужно модифицировать назначаемые по умолчанию права доступа при создании новых файлов, чтобы исключить возможность модификации файлов и каталогов другими пользователями – членами группы и всеми остальными пользователями системы? Проверьте результаты, создав еще один новый файл и еще один новый каталог.

Ответ:

```
$ umask a-rwx,u=rwx,g=r,o=r
$ touch new_file2
$ ls -l new_file2
-rw-r--r-- 1 YOUR_USER_NAME class      0 Jul 24 13:13
new_file2
$ mkdir new_dir2
$ ls -ld new_dir2
drwxr--r-- 3 YOUR_USER_NAME class 1024 Jul 24 13:13 new_dir2
```

7–21. УПРАЖНЕНИЯ: Основы shell

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Создайте псевдоним команды с именем **h**, который позволит выполнить команду **history**.

Ответ:

```
$ alias h=history
```

2. Проверьте команды, включенные в файл **.shrc** (или **.kshrc**) в вашем входном каталоге. Добавьте к списку команд псевдоним **h**.

Ответ:

```
vi .shrc
```

добавить новую строку

```
alias h=history
```

3. В командной строке определите псевдоним команды с именем **go**, который позволит перейти в каталог **tree** и выполнить в нем команду **ls -F**. Теперь выполните из командной строки команду **go**. Что произошло? Выполните команду **pwd**, чтобы увидеть, где вы находитесь. Вернитесь в свой входной каталог. (ПОДСКАЗКА: в одной командной строке можно ввести несколько команд, если разделять их точкой с запятой)

Ответ:

```
$ alias go="cd /home/user3/tree; ls -F"
$ go
car.models/ dog.breeds/ fruit/ horses/
$ pwd
/home/user5/tree
$ cd
```

4. Завершите сеанс работы, а затем вновь войдите в систему и проверьте свои псевдонимы. Почему нужно было выходить из системы?

Ответ:

Необходимо заново прочитать файлы **.profile** и **.shrc**. Самый простой способ сделать это – выйти из системы и затем вновь войти в нее.

Система Linux:



В системе Linux нужно прочитать файл **.bash_profile**.

5. Убедитесь в том, что вы находитесь в вашем **HOME** каталоге. Что произойдет, если вы введете с клавиатуры текст **more f** Esc Esc? Каким образом можно увидеть в командной строке имя файла **funfile**?

Ответ:

Ввод указанного текста с клавиатуры приведет к тому, что слова **more f** появятся в командной строке, а затем в ответ на нажатие следующих клавиш shell издаст звуковой сигнал, так как в текущем каталоге есть несколько файлов, имена которых начинаются с буквы **f**. Если напечатать букву **u** и затем снова нажать **[Esc]** **[Esc]**, будет завершен ввод имени файла **funfile**.

В Linux Korn shell'e необходимо предварительно установить в качестве редактора командной строки редактор **vi**:

```
$ set -o vi
```

Для завершения ввода имени файла следует использовать клавиши **[Esc]** **[]** вместо **[Esc]** **[Esc]**.

6. Находясь в *HOME* каталоге, скопируйте файл **frankenstein** в каталог **tree/car.models/ford/sports**. Используйте завершение ввода имени файла **frankenstein**, а также любого имени в маршрутном имени каталога.

Ответ (HP-UX):

```
$ cp fr [Esc] [Esc] tree/ca [Esc] [Esc] ford/sports
$ cp frankenstein tree/car.models/ford/sports
```

Ответ (Linux – Korn shell)



```
$ cp fr [Esc] [ ] tree/ca [Esc] [ ] ford/sports
$ cp frankenstein tree/car.models/ford/sports
```

7. Введите в командной строке следующие неправильные команды; *НЕ НАЖИМАЙТЕ* **[Return]**:

```
cd /user/spol/ko/interface
```

Используя редактирование командной строки, исправьте текст так, чтобы можно было прочитать следующее:

```
cd /usr/spool/lp/interface
```

(*Не* вводите заново правильный текст).

Ответ:

```
$ cd /user/spol/ko/interface [Esc]
```

Чтобы внести необходимые изменения, используйте клавиши **[Backspace]** и пробела для позиционирования курсора и команды **x**, **a**, **cw** редактора **vi** для редактирования. Не забывайте использовать клавишу **[Esc]** каждый раз, когда нужно выйти из режима ввода.

8. Выполните команду **ls -F**.

Вызовите повторно эту команду на экран и замените **ls -F** на **ls -l**, используя необходимые команды редактора **vi**. Выполните полученную команду.

Ответы и решения

Ответ:

```
$ ls -F
$ [Esc] k
```

С помощью команды **l** переместите курсор под букву **F**; с помощью команды **r** замените ее буквой **l** и нажмите **Return**: **[l] [l] [l] [l] [r] l [Return]**

9. Используя стек команд, повторно вызовите на экран последнюю команду копирования и замените в ней имя **frankenstein** на **funfile**.

Ответ:

```
[Esc] k
$ cp frankenstein tree/car.models/ford/sports
[l] [l] [l] или [w]
[c] [w] funfile [Return]
$ cp funfile tree/car.models/ford/sports/
```

10. Повторно вызовите на экран последнюю команду копирования и модифицируйте ее таким образом, чтобы можно было вывести на экран содержимое каталога **sports**.

Ответ:

```
[Esc] k
$ cp funfile tree/car.models/ford/sports
[c] [w] ls [ESC]      замена слова cp на ls
[w]
[d] [w]              удаление слова funfile
[Return]
$ ls tree/car.models/ford/sports
```

11. Вызовите повторно на экран последнюю выполнявшуюся команду и модифицируйте ее таким образом, чтобы можно было перейти в каталог **sedan** (ПОДСКАЗКА: маршрутное имя этого каталога – **tree/car.models/ford/sedan**). Используйте команду **pwd**, чтобы убедиться в изменении рабочего каталога.

Ответ:

```
[ESC] k
$ ls tree/car.models/ford/sports
[c] [w] cd [ESC]      замена слова ls на cd
[w] [w] . . .        повторять, пока курсор не переместится под начало слова sports
[c] [w] sedan [ESC]  замена слова sports на sedan
[Return]
$ cd tree/car.models/ford/sedan
$ pwd
tree/car.models/ford/sedan
```

12. Вернитесь в свой **HOME** каталог и с помощью команды **history** или созданного псевдонима **h** выведите на экран содержимое стек команд; затем с помощью команды **r** выполните повторно команду перехода в каталог **sedan**. Также с помощью команды **r** выведите на экран имя текущего рабочего каталога.

Ответ:

```
$ cd
$ history
$ r 'cd t'   или   r номер_команды
$ r p
```


8–13. УПРАЖНЕНИЯ: Среда shell

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Используя подстановку команд, присвойте переменной *today* значение, соответствующее текущей дате.

Ответ:

```
$ date
Fri Apr 2 11:57:21 EST 1993
$ today=$(date)
echo $today
Fri Apr 2 11:57:21 EST 1993
```

2. Как легче всего вывести содержимое входного каталога другого пользователя?

Ответ:

Предположим, что входное имя пользователя *mike*; тогда можно вывести содержимое его входного каталога следующим образом:

```
$ ls ~mike
```

3. Присвойте переменной *shell*'а *MYNAME* значение вашего имени. Как можно увидеть значение этой переменной?

Ответ:

```
$ MYNAME=stud3
$ echo $MYNAME
stud3
```

4. Создайте порожденный процесс *shell*, выполнив команду **sh**. Проверьте значение переменной *MYNAME*. Что произошло? Завершите порожденный *shell* (используйте **Ctrl**+**d** или **exit**). Теперь родительский процесс знает что-нибудь о переменной *MYNAME*?

Ответ:

Переменная *MYNAME* размещена в области локальных данных родительского процесса *shell*. Когда создается порожденный процесс *shell*, он наследует от родительского процесса только его среду.

Когда порожденный процесс завершится, родительский процесс проснется и вспомнит все, что он знал. Это можно проверить, выполнив команду

```
$ echo $MYNAME
```

5. Выполните в родительском *shell*'е команду, которая позволит порожденному *shell*'у увидеть значение переменной *MYNAME*. Как можно увидеть все переменные, унаследованные порожденным *shell*'ом от родительского?

Ответ:

```
$ export MYNAME
```

Ответы и решения

```
$ env
```

6. Создайте еще один порожденный shell. Проверьте значение переменной *MYNAME*. Теперь измените значение этой переменной, присвоив ей значение имени вашего коллеги. Где теперь размещена переменная *MYNAME* – в среде или в области локальных данных? Выведите список значений переменных среды. Какое значение имеет переменная *MYNAME*?

Ответ:

```
$ MYNAME=stud2
$ env
```

Переменная *MYNAME* остается в среде порожденного процесса shell.

7. Теперь удалите переменную *MYNAME* из порожденного shell'a. После этого будет ли переменная *MYNAME* существовать в области локальных данных порожденного shell'a или в его среде? Почему?

Ответ:

```
unset MYNAME
```

MYNAME теперь не существует в порожденном shell'e – ни в области локальных данных, ни в среде, так как команда **unset** отовсюду удаляет эту переменную.

8. Завершите порожденный shell и вернитесь в свой входной shell. Продолжает ли существовать в нем переменная *MYNAME*? Почему? Какую команду вы использовали для проверки?

Ответ:

```
$ [Ctrl] + [d] [Return]
```

Удаление переменной из порожденного shell'a не влияет на переменные родительского shell'a. Поэтому переменная *MYNAME* продолжает оставаться в среде родительского shell'a. Чтобы проверить это, можно вывести на экран переменные среды родительского shell'a.

```
$ env
```

9. Модифицируйте первичное приглашение shell'a так, чтобы выводился текст: **good_day\$**. Что произойдет с этим приглашением, если вы завершите сеанс работы в системе, а затем вновь войдете в систему?

Ответ:

```
$ PS1=good_day
good_day$
```

Когда вы завершите сеанс работы в системе и затем вновь войдете в систему, первичное приглашение shell'a будет выглядеть в соответствии с правилами по умолчанию (**\$**).

10. Измените первичное приглашение shell'a так, чтобы выводилось ваше входное имя. Например, если вы зарегистрированы в системе под именем **stud3**, первичное приглашение должно выглядеть так: **stud3\$**. (ПОДСКАЗКА: есть ли переменная среды, значением которой является входное имя пользователя?)

Ответ:

```
$ PS1=$LOGNAME      или   $ PS1=$(whoami)
stud3                stud3
```

11. Присвойте переменной *dir* значение **/usr/bin/ls**. Как можно использовать эту переменную, чтобы выполнить команду **ls**? Допускает ли такое использование переменной *dir* указание имени каталога или файла в качестве аргумента?

Ответ:

```
$ dir=/usr/bin/ls
$ $dir
```

\$dir будет замещено значением **/usr/bin/ls**, после чего будет выполняться команда **/usr/bin/ls**. Эта команда допускает задание аргументов. Попробуйте выполнить:

```
$ $dir $HOME /tmp /var
```

9–8. УПРАЖНЕНИЯ: Генерация имен файлов

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Перейдите в свой *HOME* каталог. Выполните команду **ls *** и объясните ее результаты.

Ответ:

Результаты работы команды **ls** зависят от того, как заданы ее аргументы. Если аргументом команды является имя файла, **ls** выводит это имя. Если же аргументом является имя каталога, **ls** выводит содержимое этого каталога. Поэтому, если среди аргументов команды встречаются и имена файлов, и имена каталогов, что имеет место при использовании символа генерации имен файлов звездочка (*), **ls** выведет список имен всех файлов, находящихся в текущем рабочем каталоге, а также содержимое всех подкаталогов текущего каталога.

2. Если при выполнении команды **echo ???XX** получен результат: **???XX**, что это означает?

Ответ:

Это означает, что в текущем рабочем каталоге нет файла, имя которого удовлетворяет заданному шаблону **???XX**. Поэтому символы генерации имен файлов рассматриваются как обычные алфавитно-цифровые символы, и команда **echo** выводит их.

3. Перейдите в свой *HOME* каталог. Какую команду здесь нужно выполнить, чтобы получить следующие результаты:
 - a. Вывести все имена файлов, заканчивающиеся символами **.c**.
 - b. Вывести все имена файлов, которые заканчиваются символами **.c** и начинаются словом **mod**.
 - c. Вывести все имена файлов, которые в любом месте содержат слово **file**.
 - d. Вывести все имена файлов, которые заканчиваются или символами **.c**, или **.f**, или **.p**.

Ответ:

- a. **ls *.c**

```
libtest.c  math.c  mod1.c  mod2.c  mod3.c  myprog.c  part1.c
part2.c  xdbtest.c
```

- b. **ls mod*.c** или **ls mod[0-9].c** или **ls mod?.c**

```
mod1.c  mod2.c  mod3.c
```

- c. **ls *file***

```
file.1 file.2 herfile myfile my_root_file ourfile root_file
yourfile filegen/: Abc Abcd abc abcdemf e35f efg fe3f fe3fg
```

d. `ls *.*[cfp]`

```
libtest.c math.c math.f math.p mod1.c mod2.c mod3.c myprog.c
myprog.f myprog.p xdbtest.c part1.c part2.c
```

4. Создайте каталог **c_source**. Используя символы генерации имен файлов, переместите из текущего каталога в каталог **c_source** все файлы, имена которых оканчиваются на **.c**.

Ответ:

```
$ mkdir c_source
$ mv *.c c_source
```

5. Создайте в своем *HOME* каталоге каталог **dir_1**. Что произойдет, если вы выполните команду: **cd dir***?

Ответ:

```
$ cd dir*      Расширяется в: cd dir_1
```

Вы действительно перейдете в каталог **dir_1**.

6. Вернитесь в свой *HOME* каталог и создайте в нем каталоги **dir_2**, **dir_3** и **dir_4**. Теперь снова попробуйте выполнить команду **cd dir***. Объясните, что случилось.

Ответ:

```
$ cd
$ mkdir dir_2 dir_3 dir_4
$ cd dir*      Расширяется в: cd dir_1 dir_2 dir_3 dir_4
```

Такое использование команды является неправильным: команда **cd** требует задания только одного аргумента, поэтому она завершится с ошибкой.

7. С помощью команды **touch** (синтаксис: **touch имя_файла**) создайте файлы, для которых будут выполняться перечисленные ниже требования:

Шаблону **?XX** соответствует имя только ОДНОГО файла.

Шаблону **?..XX** соответствуют имена только ДВУХ файлов.

Шаблону ***XX** соответствуют имена только ТРЕХ файлов.

Шаблону **XX.??** соответствует имя только ОДНОГО файла.

Шаблону **XX.*** соответствуют имена только ДВУХ файлов.

Используйте команду **echo**, чтобы проверить полученные результаты.

Ответ:

```
$ touch AXX
$ touch A..XX
$ touch B..XX
$ touch XX.AB
$ touch XX.A
$ echo ?XX
```

Ответы и решения

```
AXX
$ echo ?.XX
A.XX B.XX
$ echo *XX
A.XX AXX B.XX
$ echo XX.??
XX.AB
$ echo XX.*
XX.A XX.AB
```

8. Используйте команду **rm** так, чтобы за один раз удалить все файлы, созданные в предыдущем упражнении (ПОДСКАЗКА: для большей уверенности можно использовать команду в виде **rm -i**)

Ответ:

```
$ rm *XX*
```

10–7. УПРАЖНЕНИЯ: Экранирование

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Введите команду **echo** таким образом, чтобы она вывела на экран следующий текст:

```
$1 million dollars    ...    and that's a bargain !
```

Ответ:

```
$ echo "\$1 million dollars    ...    and that's a bargain !"
$ echo '$1 million dollars    ...    and that\'s a bargain !
```

Чтобы вывести на экран указанный текст, можно по-разному использовать команду **echo**. Следует учитывать, что команда **echo** использует символы пробела как разделители слов – аргументов команды. Есть опции, которые позволяют управлять таким поведением команды. В действительности, необходимо экранировать только символы **\$** и **'**.

2. Присвойте переменной *long_string* следующее значение:

```
$1 million dollars    ...    and that's a bargain !
```

Выведите значение переменной *long_string*, чтобы убедиться в правильности присваивания.

Ответ:

```
$ long_string="\$1 million dollars    ...    and that's a bargain !"
$ echo "$long_string"
```

Эта задача имеет несколько решений. Необходимо экранировать все пробелы в строке, чтобы присваивание выполнилось успешно. Посмотрите, что произойдет, если в команде **echo** вы не используете двойные кавычки для вывода **\$long_string**.

3. Что произойдет при выполнении следующих команд?

```
$ banner good day
$ banner 'good day'
```

Сколько аргументов указано в каждой приведенной выше командной строке?

Ответ:

```
$ banner good day                                два аргумента в командной строке
GOOD
DAY
$ banner "good day"                              один аргумент в командной строке
GOOD DAY
```

Одиночные кавычки экранируют пробел, отменяя его специальный смысл как символа – разделителя аргументов, поэтому во второй команде указан только один аргумент.

Ответы и решения

Ответ (Linux):



```
$ banner good day
ksh: banner: not found
$ banner "good day"
ksh: banner: not found
```

два аргумента в командной строке

один аргумент в командной строке

Команда `banner` в системе Linux не может быть использована для вывода на экран. По умолчанию, команда `banner (/usr/games/banner)` не доступна конечным пользователям.

4. Установите в качестве первичного приглашения строку: *Way to go ваше_входное_имя* \$

Ответ:

```
$ PS1="Way to go $LOGNAME $ "
Way to go stud3 $
```

5. Как можно вывести на экран следующее сообщение?

Exercises #1, #2, and #3 are now complete.

Ответ:

```
echo Exercises \#1, \#2, and \#3 are now complete.
```

Символ `#` является специальным – он начинает комментарий, поэтому его надо экранировать.

6. Предположим, что переменная *abc* не определена. Что произойдет при выполнении следующей команды?

```
echo '$abc'
```

Что произойдет при выполнении следующей команды?

```
echo "$abc"
```

Ответ:

```
$ echo '$abc'
```

```
$abc
```

Одиночные кавычки (`'`) *не* допускают выполнение подстановки переменной, поэтому на экран будет выведен весь текст *\$abc*

```
$ echo "$abc"
```

```
sh: abc: parameter not set.
```

Когда символ `$` используется внутри двойных кавычек (`"`), shell пытается выполнить подстановку указанной переменной. Так как этой переменной не было присвоено

никакое значение, при попытке получить значение переменной shell генерирует сообщение об ошибке.

Примечание: при обращении к переменной, значение которой не определено, POSIX/Korn shell генерируют сообщение об ошибке в том случае, если установлена опция **set -u**. Чтобы увидеть, какие опции установлены для вашего shell'a, следует выполнить команду **set -o**. Если опция **-u** не установлена, обращение к не установленной переменной не будет интерпретироваться как ошибка, а в качестве значения такой переменной передается NULL. Можно отменить эту опцию, выполнив команду **set +u**. Более подробное описание можно найти в статье *Справочника sh-posix(1)*.

7. Используя команду **touch**, создайте файл, имя которого состоит:

- из слов White Space,
- из четырех пробелов,
- из трех пробелов.

Как будут выглядеть такие имена файлов при выводе содержимого каталога? В каком виде следует вывести содержимое каталога, чтобы можно было определить, сколько пробелов – 3 или 4 – использовано в имени файла?

Ответ:

```
$ touch "White Space"
$ touch "    "
$ touch "  "
```

Когда используется команда **ls**, имена файлов, которые начинаются пробелом, будут выводиться первыми. Можно определить, сколько пробелов включено в имя файла, если сделать этот файл исполняемым (добавить право **x**) и затем выполнить команду **ls -F**; теперь можно увидеть, сколько пробелов появится перед звездочкой (*).

Следует обратить внимание на то, что, используя механизм экранирования, можно создавать файлы, имена которых содержат любой специальный символ. В этом случае всегда, когда нужно обратиться к такому файлу, при записи его имени необходимо использовать символы экранирования.

11–11. УПРАЖНЕНИЯ: Перенаправление ввода/вывода

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Выполните перенаправление результатов команды **date** в файл **date.out**, находящийся в вашем *HOME* каталоге.

Ответ:

```
$ cd
$ date > date.out
```

2. Добавьте в файл **date.out** результаты команды **ls**. Посмотрите содержимое файла **date.out**. Что вы заметили?

Ответ:

```
$ ls >> date.out
$ more date.out
```

Результатом команды **ls** является список файлов, находящихся в текущем рабочем каталоге. Каждое имя файла размещается в отдельной строке. Shell выполняет вывод результатов команды **ls** в несколько колонок, только если выводит на терминал. Можно получить другие результаты, если использовать опцию **-C** команды **ls**.

3. Используя перенаправление ввода, пошлите электронной почтой файл **date.out** вашему коллеге.

Ответ:

```
$ mail входное_имя_коллеги < date.out
```

4. Используя команду **cat** и перенаправление вывода, создайте в своем *HOME* каталоге два коротких файла – **f1** и **f2**.

Ответ:

```
$ cat > f1
This is the file f1
[Ctrl] + [d]
$ cat > f2
This is the file f2
Ctrl + d
```

5. Используя команду **cat**, посмотрите содержимое созданных в предыдущем упражнении файлов. С помощью этой же команды создайте новый файл **f.join**, содержащий информацию из обоих этих файлов – **f1** и **f2**. Видите ли вы что-нибудь на экране?

Ответ:

```
$ cat f1 f
This is the file f1
This is the file f2
$ cat f1 f2 > f.join
```

содержимое обоих файлов записывается в f.join

На экране ничего не будет видно – все, что выводится в стандартный вывод, записывается в файл.

6. Используя команду **cat**, выведите на экран содержимое файлов **f1**, **f2** и **f.new**. ПРИМЕЧАНИЕ: файл **f.new** *НЕ ДОЛЖЕН* существовать. Что вы увидите на экране? Можно ли определить, какие сообщения выводились в стандартный вывод, а какие – в стандартный протокол?

Ответ:

```
$ cat f1 f2 f.new
This is the file f1
This is the file f2
cat: Cannot open f.new
```

Из результатов, выведенных на экран, не очевидно, что для вывода использовались два выходных потока, так как оба потока назначены на экран терминала.

7. Используя команду **cat**, снова выведите на экран содержимое файлов **f1**, **f2** и **f.new**. ПРИМЕЧАНИЕ: файл **f.new** опять *НЕ ДОЛЖЕН* существовать. На этот раз перехватите сообщения об ошибках, генерируемые командой, и направьте их в файл **f.error**. Что вы увидите на экране? Создан ли новый файл? Проверьте его содержимое.

Ответ:

```
$ cat f1 f2 f.new 2> f.error
This is the file f1
This is the file f2
$ cat f.error
cat: Cannot open f.new
```

8. Используйте опять команду **cat** для вывода содержимого файлов **f1**, **f2** и **f.new**. ПРИМЕЧАНИЕ: файл **f.new** и на этот раз *НЕ ДОЛЖЕН* существовать. Теперь перехватите сообщения, выводимые в стандартный вывод, и направьте их в файл **f.good**, и *В ЭТОЙ ЖЕ КОМАНДЕ* направьте вывод диагностических сообщений в файл **f.bad**. Что вы увидите на экране? Созданы ли какие-либо новые файлы? Проверьте их содержимое.

Ответ:

```
$ cat f1 f2 f.new > f.good 2> f.bad
$ cat f.good
This is the file f1
This is the file f2
$ cat f.bad
cat: Cannot open f.new
```

Будут созданы файлы **f.good** и **f.bad**. На экране ничего нельзя будет увидеть, так как оба выходных потока перенаправлены в файлы.

9. Выполните команду **cp** без аргументов. Что произошло? Попробуйте перенаправить стандартный вывод команды в файл **cp.error**. Что произошло? Что нужно сделать, чтобы перенаправить в файл сообщения об ошибках? Выводит ли команда **cp** какие-либо результаты в стандартный вывод?

Ответы и решения

Ответ:

```
$ cp
Usage: cp f1 f2
cp [-r] f1 ... fn d1
$ cp 2> cp.error
```

Команда **cp** не выводит какие-либо результаты в стандартный вывод. Обычно она в случае успешного выполнения завершается молча.

10. Выведите на экран содержимое файла **/etc/passwd**, упорядочив его по входному имени пользователя.

Ответ:

```
$ sort -d /etc/passwd
```

11. Выведите содержимое файла **/etc/passwd**, упорядоченное по третьему полю. Что получилось? Снова выполните эту команду, указав в ней числовое упорядочение по третьему полю. Есть ли отличия в результатах выполнения этих двух команд?

Ответ:

```
$ sort -t: -k 3 /etc/passwd    лексикографическая сортировка
```

Обратите внимание на то, что числовые значения в третьем поле вовсе не упорядочены. Это произошло потому, что для поля, содержащего числовые значения, была выполнена лексикографическая сортировка.

```
$ sort -nt: -k 3 /etc/passwd    числовая сортировка
```

Результаты выполнения этой команды существенно лучше, так как теперь значения, содержащиеся в третьем поле, рассматриваются как числа и упорядочиваются по числовым значениям.

12. Выберите из файла **/etc/passwd** все строки, в которых встречается строка **stud**. Сохраните результаты в файле **grepped**. С помощью фильтра определите, сколько строк файла **/etc/passwd** содержат строку **stud**.

Ответ:

```
$ grep stud /etc/passwd > grepped
$ wc -l grepped
16 grepped
```

(В той системе, в которой работаете вы, результаты выполнения фильтра могут быть другими.)

13. Используя перенаправление вывода и фильтры, определите, сколько пользователей в данный момент работает в системе.

Ответ:

```
$ who > whoson
$ wc -l whoson
```

14. Сколько пользователей зарегистрировано в вашей системе? Какие команды вы использовали для ответа на поставленный вопрос? (ПОДСКАЗКА: Каждому

пользователю, зарегистрированному в системе, соответствует одна строка в файле **/etc/passwd**)

Ответ:

```
$ wc -l /etc/passwd
```

15. Упорядочите файл **names** из вашего входного каталога и сохраните результат в файле **names.sort**. Снова отсортируйте этот файл, но теперь в обратном порядке, и сохраните результат в файле **names.rev**. Какие команды вы использовали? Обратитесь к статье *Справочника* **sort(1)** и найдите опцию, которая позволит команде сохранить результаты своей работы в исходном файле **names**.

Ответ:

```
$ sort names > names.sort
$ sort -r names > names.rev
$ sort names -o names
```

16. Направьте вывод команды **banner** на терминал вашего коллеги. ПОДСКАЗКА: Вы знаете, какой файл устройства ассоциирован с терминалом вашего коллеги? Если вы получите сообщение **Permission denied**, что это значит?

Ответ:

Прежде всего, необходимо определить файл устройства, ассоциированный с терминалом вашего коллеги:

```
$ who > whoson
$ grep входное_имя_коллеги whoson
входное_имя_коллеги tty03 Jul 16 8:02
```

С помощью команды **tty** проверьте назначения терминалов. Это позволит определить, какой файл устройства ассоциирован с терминалом, за которым работает ваш коллега.

```
$ banner good morning > /dev/tty03
```

Если будет получено сообщение **Permission denied**, это значит, что ваш коллега с помощью команды **mesg n** отменил всем, кроме себя, право записи на свой терминал.

Система Linux

Ответ (Linux – Korn shell)



Так как в системе Linux команда **banner** не доступна, используйте в данном упражнении команду **echo** и направьте ее вывод на терминал вашего коллеги. Например:

```
$ echo "XXXXXXXXX Good Morning XXXXXXXXX" > /dev/tty03
```

12–13. УПРАЖНЕНИЯ: Конвейеры

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Создайте конвейер, который подсчитывает количество пользователей, в настоящее время работающих в системе.

Ответ:

```
$ who | wc -l
```

2. Создайте конвейер, подсчитывающий количество строк в файле **/etc/passwd**, которые удовлетворяют модели поиска **home**. Затем подсчитайте количество строк, которые не удовлетворяют модели поиска.

Ответ:

```
$ grep home /etc/passwd | wc -l      количество строк, содержащих home
$ grep -v home /etc/passwd | wc -l   количество строк, не содержащих home
```

3. Модифицируйте конвейер, созданный в предыдущем упражнении, так, чтобы все строки файла **/etc/passwd**, удовлетворяющие модели поиска **home**, были сохранены в файле **all.users**, прежде чем будет подсчитано их количество.

Ответ:

```
$ grep home /etc/passwd | tee all.users | wc -l
```

4. Создайте конвейер, который упорядочит содержимое файла **names** из вашего **HOME** каталога и выведет результаты в три колонки без заголовка и хвостовика.

Ответ:

```
$ sort names | pr -3 -t
```

5. Создайте псевдоним **whoson**, который выводит упорядоченный по алфавиту список всех пользователей, работающих в данный момент в системе.

Ответ:

```
$ alias whoson="who | sort"
```

6. Создайте конвейер, который позволит получить список, состоящий только из имен пользователей, в настоящее время работающих в системе.

Ответ:

```
$ who | cut -c1-8
```

или

```
$ who | cut -f1 -d" "
```

7. Создайте конвейер, который позволит получить список имен файлов из вашего рабочего каталога и их прав доступа.

Ответ:

```
$ ll | cut -c2-10,58-
```

8. Создайте конвейер, который для каждого файла из вашего *HOME* каталога запишет в файл **listing.out** имя владельца файла, размер файла и имя файла и одновременно выведет на экран общее количество файлов.

Ответ:

```
$ ll | cut -c16-24,34-44,58- | tee listing.out | wc -l
```

9. Создайте конвейер, который для каждого пользователя системы выберет имя пользователя, его числовой идентификатор и имя *HOME* каталога. Выведите полученный список, упорядочив его по имени пользователя. Измените конвейер так, чтобы список был упорядочен по числовым идентификаторам пользователей. ПОДСКАЗКА: необходимую информацию можно получить из файла **/etc/passwd**.

Ответ:

```
$ cut -f1,3,6 -d: /etc/passwd | sort алфавитная сортировка
$ cut -f1,3,6 -d: /etc/passwd | sort -n -t: -k 2 числовая сортировка
```

13–11. УПРАЖНЕНИЯ

Указания

Узнайте у вашего преподавателя, какие упражнения вы сможете выполнить на занятии. Также узнайте имена компьютеров, с которыми вы сможете установить связь.

Система Linux



ПРИМЕЧАНИЕ: Пользователи системы Linux не смогут выполнить упражнения 3 и 4 из-за применяемой в системе по умолчанию политики безопасности. Поэтому пользователям системы Linux не следует пытаться выполнять эти упражнения.

1. Используя команду **hostname**, определите сетевое имя локального компьютера, на котором вы работаете. С какими компьютерами в сети вы можете установить связь?

Ответ:

Команда **hostname** выводит на экран сетевое имя локального компьютера. В файле **/etc/hosts** можно найти имена всех компьютеров, с которыми можно установить связь.

2. Используя команду **telnet**, установите соединение с другим компьютером. Используя команду **hostname**, проверьте, что вы установили связь именно с тем компьютером, с каким хотели. После этого завершите сеанс работы с удаленным компьютером.

Ответ:

```
$ telnet fred
Trying...
Connected to fred
Escape character is '^]'.

HP-UX fred 10.00 B 9000/715

login: введите свое входное имя
Password: введите пароль
.
.
.
$ hostname
fred
$ exit
```

3. С помощью команды **ftp** перешлите один из своих файлов в свой *HOME* каталог на удаленном компьютере, затем с помощью команды **rsh** скопируйте другой файл на удаленный компьютер. Обратите внимание на различия.

Ответ (HP-UX):

Работая в **ftp**, для пересылки файла нужно будет использовать команду **put**, по аналогии с примерами, приведенными в пояснениях.

4. Используя команду **remsh**, выведите содержимое своего *HOME* каталога на удаленной машине, чтобы проверить, что копирование файлов в предыдущем упражнении было выполнено.

Ответ (HP-UX):

```
$ remsh имя_компьютера ls
```

Команда **ls** выведет содержимое *HOME* каталога, находящегося на указанном компьютере.

14–11. УПРАЖНЕНИЯ: Ввод и удаление текста, перемещение курсора

Указания

Используя **vi**, начните сеанс редактирования файла **tst** из вашего *HOME* каталога. Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Запустите **vi** для файла **tst**.

Ответ:

```
$ vi tst
```

2. Вставьте слово *only* между словами *will be*.

Ответ:

Переместите курсор на вторую строку: нажмите **j** или **Return** или **2G**.

а) Переместите курсор под последнюю букву *l* в слове *will*: нажмите **ee** или **2e** или несколько раз **l**. Добавьте текст после буквы *l*: нажмите **a**; введите **only**, или

б) Переместите курсор под первую букву *b* в слове *be*: нажмите **ww** или **2w** или несколько раз **l**. Вставьте текст перед буквой *b*: нажмите **i**; введите **only**.

3. Добавьте слова *many, many* в конце строки *"It will be used for"*.

Ответ:

Вернитесь в режим команд, нажав **Esc**.

Переместите курсор в конец строки, введя **\$**.

Добавьте текст: нажмите **a**; введите текст **many, many**.

4. Добавьте новую строку в конец файла. Введите в ней свое имя. По окончании ввода *HE НАЖИМАЙТЕ Esc*!

Ответ:

Вернитесь в режим команд, нажав **Esc**.

Переместите курсор в последнюю строку: **G**.

Откройте после последней строки новую: **o**.

Введите свое имя.

5. Используя клавишу **Backspace**, удалите свое имя и введите имя своего коллеги.

Ответ:

Backspace для всех символов первого имени.

Введите имя коллеги.

6. Откройте новую строку в начале файла (ПОДСКАЗКА: **o**).

Ответ:

Вернитесь в командный режим, нажав **Esc**.

1GO

7. Напечатайте **12345**.

Ответ:

12345

8. Нажмите два раза **Backspace**. Исчезли с экрана какие-либо цифры?

Ответ:

Backspace Backspace

Курсор находится под цифрой 4. Никакие цифры с экрана не исчезли.

9. Введите **1234**. Что произошло с теми цифрами, для которых нажималась клавиша **Backspace**?

Ответ:

Цифры 4 и 5 будут заменены другими цифрами.

10. Нажмите **Backspace** три раза.

Ответ:

Backspace Backspace Backspace

Курсор теперь находится под второй цифрой 2.

11. Нажмите **Esc**. Что произошло с символами, для которых нажималась клавиша **Backspace**? Где теперь находится курсор?

Ответ:

Вторая группа цифр **234** исчезнет, а курсор вернется на одну позицию назад и будет находиться под второй цифрой **1**.

12. Напечатайте четыре буквы **a**. Сколько букв **a** появилось на экране? Почему?

Ответ:

На экране появятся три буквы **a**, так как первая буква **a** будет воспринята как команда **append** редактора **vi**.

Ответы и решения

13. Нажмите **Backspace** 5 раз. Что произошло? Почему?

Ответ:

Клавишу **Backspace** можно нажать только три раза, так как во время сеанса ввода были введены только три буквы.

14. Нажмите **Esc**. Что произошло?

Ответ:

Все введенные буквы **a** исчезли. Вы вернулись в режим команд.

15. Завершите сеанс работы **vi**, сохранив все изменения в файле **tst**.

Ответ:

Введите **:wq** или **ZZ**

14–21. УПРАЖНЕНИЯ: Модификация текста

Указания

Используйте редактор **vi** для выполнения следующих упражнений:

1. Начните сеанс **vi** редактирования файла **vi.tst**; выполните те изменения, которые указаны в тексте этого файла. Ниже приводится текст файла **vi.tst** (в скобках дается перевод предлагаемых действий).

Enter your name here -> (Введите здесь свое имя)

(Замените в следующих предложениях названия цвета, цветка и книги своими)

Change the following to your favorite color -> lavender

Change the following to your favorite flower -> rose

Change the following to your favorite book -> A Tale of Two Cities

(Исправьте опечатки в следующих двух строках)

Correct the typos in the next two lines:

Corect teh typooos im thiss line.

Ther awe mroe mistakkes in thsi linne.

(Приведенные выше две строки должны читаться так)

The above two lines should read:

Correct the typos in this line.

There are more mistakes in this line.

(удалите в следующей строке все вхождения слова “jog”)

Delete every occurrence of the word "jog" in the next line:

walk jog run walk jog run walk jog run walk jog run

(В предыдущей строке замените каждое слово “walk” словом “WALK”)

Change every occurrence of the word "walk" to "WALK" in the above line.

```
line1
line2
line3
line4
line5
line6
line7
line8
```

(Выполните следующие упражнения с представленными выше строками – от 1-й до 8-й)

Complete the following exercises on line1 through line8 above:

(Удалите строки, содержащие от line1 до line5, и вставьте их после строки, содержащей line8)

1. Move the lines containing line1 through line5 and paste them after the line containing line8.

(Скопируйте строки, содержащие от line2 до line4, и вставьте их перед строкой, содержащей line6, а также после строки, содержащей line3)

2. Copy the lines containing line2 through line4 and paste them before the line containing line6, and also after the line containing line3.

Ответы и решения

(Завершите сеанс редактирования файла `vi.tst`, сохранив в нем все сделанные изменения)
Quit your edit session on "`vi.tst`" saving the changes that you have made.

2. Начните новый сеанс редактирования файла **funfile** из вашего *HOME* каталога и замените в файле все вхождения слова *bug* словом *FEATURE*.

Ответ:

```
:1,$s/bug/FEATURE/g
```

3. Запишите первые 40 строк файла **funfile** в файл **new.40**.

Ответ:

```
:1,40w new.40
```

4. Перейдите к последней строке файла **funfile**.

Ответ:

```
G
```

5. Найдите и выполните команду, которая перемещает курсор в нижнюю половину экрана. Замените половину экрана следующей строкой:

```
This file is silly.
```

Ответ:

```
cMThis file is silly.
```

Esc

6. Не выходя из **vi**, запишите новую версию файла в файл **funfile.123**.

Ответ:

```
:w funfile.123
```

7. Не выходя из **vi**, загрузите в буфер редактора файл **new.40**, перезаписав прежнее состояние буфера.

Ответ:

```
:e new.40
```

8. Используя опцию *number* редактора **ex**, включите режим нумерации строк.

Ответ:

```
:set number
```

9. Найдите в файле **new.40** вхождение слова *FEATURE*.

Ответ:

```
/FEATURE
```

10. Замените найденное слово *FEATURE* словом *BUG* и сохраните файл в файле **new.new.40**.

Ответ:

```
cwBUG Esc
:w new.new.40
```

11. Скопируйте файл **funfile** в **funfile.new**. В файле **funfile.new** найдите все вхождения строк *System* или *system* и с помощью **/**, **cw**, **n** и **.** замените все эти слова, кроме одного, словом *XXXXX*.

Ответ:

```
1G
/[Ss]ystem
cwXXXXX Esc
n
.
n
n
.
n
.
n
.
```

12. Сохраните результаты редактирования и выйдите из редактора **vi**.

Ответ:

```
:wq
```

или

```
ZZ
```

15–7. УПРАЖНЕНИЯ: Управление процессами

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Найдите в своем *HOME* каталоге файл **infinite**. Запустите эту программу на исполнение в оперативном режиме и посмотрите, что произойдет. Нажмите **Ctrl** + **c** для завершения выполнения программы.

```
$ infinite
hello world
hello world
hello world
Ctrl + c
$
```

2. Запустите программу **infinite** на выполнение в фоновом режиме и перенаправьте ее вывод в файл **infin.out**.

```
$ infinite > infin.out &
```

Выполните команду **ps -f**. Обратите внимание на значения идентификаторов PID и PPID процесса **infinite**. Теперь выйдите из системы, потом войдите снова и выполните **ps -ef | grep user_id**, где *user_id* – входное имя пользователя. Где процесс **infinite**? Удалите файл **infin.out** перед тем, как выполнять следующее упражнение.

Ответ:

Значение PID (ID процесса), соответствующее интерактивному shell'у (**-sh** для HP-UX или **ksh** для Linux), будет равно значению PPID (ID родительского процесса) процесса **infinite**. При выходе из системы HP-UX родительский процесс завершается, а также завершаются все порожденные процессы (включая **infinite**). Если же выйти из системы Linux, фоновый процесс будет продолжать работать, как если бы он был запущен командой **nohup**.

3. Команда **nohup** защищает процесс от завершения в случае, когда завершается его родительский процесс. Вновь запустите команду **infinite** на выполнение в фоновом режиме, но защитите ее от завершения с помощью команды **nohup**.

```
$ nohup infinite > infin.out &
```

Теперь выйдите из системы и затем опять войдите в нее. Снова выполните команду **ps -ef | grep user_id**. Процесс **infinite** еще выполняется? Какой процесс теперь является для него родительским?

Ответ:

Когда родительский процесс (интерактивный shell) завершается, порожденный процесс (**infinite**) становится осиротевшим. Осиротевшие процессы усыновляются процессом с идентификатором 1 (**init**). При повторном входе в систему можно увидеть, что процесс **infinite** все еще выполняется.

4. Используйте команду **kill** для завершения выполнения программы **infinite**.

Ответ:

\$ kill PID *Значение PID можно узнать, выполнив команду ps*

5. Запустите программу **infinite** на выполнение в *оперативном* режиме и перенаправьте ее вывод в файл **infin.out**. Приостановите выполнение программы, нажав клавиши **Ctrl** + **z**. На экране вы увидите сообщение о том, что процесс приостановлен. Переведите процесс **infinite** в фоновый режим, обратите внимание на сообщение. Завершите выполнение программы **infinite** с помощью команды **kill**.

Ответ:

```
$ infinite > infin.out
[Ctrl] + [z]
[1] + Stopped infinite > infin.out
$ bg %1
[1] infinite > infin.out &
$ kill %1
[1] + Terminated infinite > infin.out
```

16–12. УПРАЖНЕНИЯ: Введение в shell программирование

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Напишите shell процедуру **whoson1**, которая:

- с помощью команды **echo** выведет на экран приветствие,
- определит переменную *MYNAME* и присвоит ей значение вашего имени,
- выведет на экран значение переменной *MYNAME*,
- выведет на экран текущие дату и время,
- выведет информацию о том, какие пользователи работают в данное время в системе.

Ответ:

```
$ vi whoson1
echo Welcome to whoson1
MYNAME=ваше_имя
echo $MYNAME
date
who
$ chmod +x whoson1
$ whoson1
```

2. Перейдите в каталог **/tmp**. Вызовите на исполнение программу **color1**. Найдет shell программу **color1**?

Ответ:

```
$ cd /tmp
$ color1
sh: color1: not found
```

Программа **color1** не найдена, так как она не находится ни в одном из каталогов, перечисленных в переменной среды *PATH*.

3. Вернитесь в *HOME* каталог. Создайте в нем каталог **bin**. Переместите программу **color1** в каталог **bin**. Добавьте к переменной среды *PATH* ваш каталог **bin**, чтобы shell смог найти программу **color1**. Проверьте, что теперь переменная *PATH* обеспечит правильную работу, перейдя в каталог **/tmp** и вызвав на исполнение программу **color1**. Перед вызовом программы **color1** не забудьте определить переменную *color*.

Ответ:

```
$ cd
$ mkdir bin
$ mv color1 bin
$ PATH=$PATH:$HOME/bin
$ cd /tmp
$ color=lavender
$ export color
$ color1
You are now running program: color1
the value of the variable color is: lavender
$
```

4. Вернитесь в *HOME* каталог. Присвойте в интерактивном режиме переменной *date_var* результаты работы команды **date**. Напишите shell процедуру **date_tst**, которая выводит значение этой переменной. Установите процедуру **date_tst** в ваш каталог **bin**.

Ответ:

```
$ date_var=$(date)
$ export date_var
$ cd $HOME/bin
$ vi date_tst
echo the value of date_var is $date_var

$ chmod +x date_tst
$ date_tst
```

5. Модифицируйте процедуру **date_tst** так, чтобы значение переменной *date_var* присваивалось во время ее выполнения. Необходимо ли теперь экспортировать в среду переменную *date_var*? Нужно ли изменить права доступа к файлу **date_tst**?

Ответ:

```
$ vi $HOME/bin/date_tst
date_var=$(date)
echo the value of date_var is $date_var

$ date_tst
```

Теперь переменную *date_var* не нужно экспортировать в среду, так как она определяется и используется в одном и том же процессе. Права доступа к файлу **date_tst** изменять не нужно, так как в предыдущем упражнении этому файлу было добавлено право на исполнение. Редактирование файла не влияет на его права доступа.

6. Напишите shell процедуру **whoson2**, которая:

- с помощью команды **echo** выводит пользователю персональное приветствие в виде *welcome имя_пользователя*, так что если пользователь войдет в систему под именем **stud3**, он увидит приветствие *welcome stud3*; если же пользователь войдет в систему под именем **stud2**, он увидит приветствие *welcome stud2*. (ПОДСКАЗКА: имя пользователя можно получить, используя переменную среды или подстановку результата выполнения команды),
- выводит системные дату и время,
- выводит информацию о том, какие пользователи работают в данное время в системе,
- выводит сообщение пользователю, содержащее идентификацию его сеанса связи с системой,
- выводит сообщение перед завершением процедуры.

Переместите процедуру в каталог **bin**, чтобы вы могли выполнять ее независимо от того, в какой точке файловой системы вы находитесь.

Ответ:

```
$ vi $HOME/bin/whoson2
echo welcome $LOGNAME      или      echo welcome $(whoami)

date
who
```

Ответы и решения

```
echo your terminal session identification information is
who am I
echo thank you for using whoson2
$ chmod +x $HOME/bin/whoson2
$ whoson2
```

7. Если shell процедура вызывается на исполнение следующим образом:

```
$ myshellprog abc def -d -4 +900 xyz
```

что будет выведено на экран, если в эту shell процедуру включены следующие команды:

```
echo $#
echo $3
echo $7
echo $*
echo $0
```

Ответ:

```
6
-d
пустая строка
abc def -d -4 +900 xyz
myshellprog
```

8. Пусть shell процедура, вызванная на исполнение так, как указано в предыдущем упражнении, в первой своей строке содержит команду **shift 2**; что тогда будет выведено на экран следующими командами:

```
echo $#
echo $3
echo $7
echo $*
echo $0
```

Ответ:

```
4
+900
пустая строка
-d -4 +900 xyz
myshellprog
```

9. Каким будет результат работы приведенной ниже shell процедуры, если в ответ на приглашение пользователь введет такую строку:

```
James A. Smith, Jr.
```

Shell процедура:

```
echo "Please type in your first, middle, and last names"
read first middle last
echo "$last, $first $middle"
```

Ответ:

```
Please type in your first, middle, and last names
James A. Smith, Jr
Smith, Jr., James A.
```

Обратите внимание на то, что текст *Smith, Jr.* будет введен в последнюю переменную.

10. Напишите shell процедуру **search1**, предлагающую пользователю ввести строку, которую следует искать в файлах, находящихся в текущем каталоге. На экран следует выводить имена файлов, в которых встречается введенная строка.

Ответ:

Ниже приводится текст shell процедуры **search1**:

```
echo "Please enter a string to search for: \c"
read string
echo The following files contain the string $string:
grep -l $string *
```

11. Напишите shell процедуру **backwards**, которая принимает не более 10 аргументов и выводит аргументы в обратном порядке.

Ответ:

```
#!/usr/bin/sh
# backwards: reverses command line arguments
# usage: reverse a b c d e f g h i
#
echo ${10} $9 $8 $7 $6 $5 $4 $3 $2 $1
```

12. Напишите shell процедуру **myecho**, которая:

- выводит на экран количество аргументов, переданных ей из командной строки,
- выводит на экран первые три аргумента, каждый в отдельной строке,
- выводит все остальные аргументы в одной строке.

Выполните процедуру, передав ей 12 аргументов.

Каким должен быть список аргументов этой процедуры, чтобы в результате ее выполнения получилось следующее:

```
I cannot
seem to
find my KEYS.
```

Ответ:

```
#!/usr/bin/sh
# myecho: Display the number of command line arguments,
#         print the first three arguments on separate lines
#         and print the remaining arguments on one line
#         usage: myecho a b c
#
echo "The number of command line arguments is $#."
#
echo $1;echo $2;echo $3
shift 3
echo $*
$ myecho a b c d e f g h i j k l
The number of command line arguments is 12
a
b
c
d e f g h i j k l
$ myecho "I cannot" "seem to" "find my KEYS."
```

Ответы и решения

13. Напишите shell процедуру **my_vi**, аргумент командной строки которой определяет имя файла, предназначенного для редактирования. Процедура **my_vi** должна сделать резервную копию указанного файла, а затем начать сеанс редактирования этого файла с помощью редактора **vi**. При создании резервной копии файла используйте расширение имени файла **.bak**. Используйте имена файлов длиной не более 10 символов.

Ответ:

```
#!/usr/bin/sh
# my_vi: Create a backup file prior to starting a vi session
# usage: my_vi filename
#
echo Copying $1 to ${1}.bak
cp $1 ${1}.bak
vi $1
echo Edit of $1 is complete
echo You may recover your original file from ${1}.bak
```

14. Создайте shell процедуру **my_recover**, являющуюся дополнением к процедуре **my_vi**; новая процедура должна восстанавливать из резервной копии файл, заданный аргументом командной строки. В командной строке следует указывать имя файла без расширения **.bak**. Например, чтобы восстановить файл **tst1** из резервной копии **tst1.bak**, нужно выполнить команду **my_recover tst1**.

Ответ:

```
#!/usr/bin/sh
# my_recover: Recover a file from backup
# usage: my_recover filename
echo Restoring $1 from ${1}.bak
cp ${1}.bak $1
echo $1 is recovered
```

15. Напишите shell процедуру **info**, которая выводит пользователю приглашение ввести:

- имя,
- название улицы (часть адреса),
- город, государство, zip-код (почтовый индекс).

Процедура должна запомнить введенные данные в переменных и вывести их в информативном виде, с необходимыми комментариями.

Ответ:

```
#!/usr/bin/sh
# info: Prompt user for mailing address
#
echo "Input your name: \c"
read name
echo "Input your street address: \c"
read address
echo "Input your City, State, and Zip Code: \c"
read where
echo;echo
echo "Your name is   $name"
echo "You live at    $address"
echo "               $where"
```

16. Напишите shell процедуру **home**, которая предлагает пользователю ввести *входное имя* любого пользователя, зарегистрированного в системе, и выводит имя *HOME* каталога этого пользователя. Помните, что имя *HOME* каталога пользователя указано в шестом поле файла паролей **/etc/passwd**. Процедура должна вывести *входные имена* из файла **/etc/passwd** в четыре столбца, так чтобы пользователь знал существующие в системе входные имена пользователей.

Ответ:

```
#!/usr/bin/sh
# home: Return the value of a user's HOME directory
# usage: home
echo Select a user identifier from the following list:
cut -f1 -d: /etc/passwd | pr -4 -t
echo "Input user identifier: \c"
read user
home=$(grep $user: /etc/passwd | cut -f6 -d:)
echo;echo "user: $user HOME directory: $home"
```

17. Напишите shell процедуру **alpha**, которая выводит на экран первый и последний аргументы командной строки. ПОДСКАЗКА: используйте команду **cut**.

Ответ:

```
#!/usr/bin/sh
# alpha: Displays the first and last command line arguments
#
last=$(echo $* | cut -f$# -d" ")
echo "The first command line argument is $1."
echo "The last command line argument is $last."
```

18. Напишите shell процедуру **copy**, которая предоставляет пользователю удобный интерфейс для копирования файлов с помощью команды **cp**. Процедура должна вывести пользователю приглашение ввести имена файлов, которые нужно копировать, а затем приглашение ввести имя приемника копируемых файлов. Если копируется несколько файлов, в качестве приемника необходимо указать имя каталога; если же копируется один файл, в качестве приемника может быть указано имя файла. По окончании копирования процедура должна выдать звуковой сигнал.

Ответ:

```
#!/usr/bin/sh
# file_copy: User interface for copying files
# usage: copy
#
echo Please enter the names of the file(s) you want to copy:
echo "-> \c"
read filenames
echo Please enter the destination.
echo NOTE!
echo If you entered more than one file, the destination must be
a directory.
echo "Enter destination here -> \c"
read dest
echo Copying files now ...
cp $filenames $dest
echo Done copying files "\a"
```

17–13. УПРАЖНЕНИЯ: Shell программирование: ветвление

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Присвойте переменной *X* значение какой-либо строки. Используя команду **test**, определите, является ли строка *xyz* значением переменной *X*. (ПОДСКАЗКА: нужно вывести на экран код завершения команды **test**)

Ответ:

```
$ X=xyz
$ test "$X" = "xyz"
$ echo $?
0
```

2. Присвойте переменной *Y* значение какого-либо числа. Используя команду **test**, определите, является ли значение переменной *Y* строго положительным. (ПОДСКАЗКА: нужно вывести на экран код завершения команды **test**)

Ответ:

```
$ Y=100
$ test "$Y" -gt 0      or      [ "$Y" -gt 0 ]
$ echo $?
0
```

```
$ Y=-100
$ test "$Y" -gt 0      or      [ "$Y" -gt 0 ]
$ echo $?
1
```

3. Напишите shell процедуру, включив в нее конструкцию **if**, которая выведет на экран слово *yes*, если переданный в процедуру аргумент имеет значение *abc*, и *no* в противном случае.

Ответ:

```
if
[ "$1" = "abc" ]
then
    echo yes
else
    echo no
fi
```

4. Напишите shell процедуру, которая выводит пользователю приглашение на ввод числа; запоминает введенное число в переменной *Y*; использует конструкцию **if**, которая выводит текст *Y is positive*, если значение *Y* действительно больше нуля, и *Y is not positive* в противном случае; выводит на экран значение *Y*. (ПОДСКАЗКА: ввод ответа пользователя выполняется с помощью команды **read**)

Ответ:

```
echo "please enter a number: \c"
read Y
if
    [ "$Y" -gt 0 ]
then
    echo Y is positive
    echo The value of Y is $Y
else
    echo Y is not positive
    echo The value of Y is $Y
fi
```

5. Напишите shell процедуру, которая определяет количество аргументов командной строки и выводит значения аргументов, если в процедуру переданы три аргумента, или сообщение об ошибке в противном случае. (ПОДСКАЗКА: Количество аргументов командной строки определяет специальная переменная shell'a \$#. Какая специальная переменная shell'a определяет все аргументы командной строки?)

Ответ:

```
if
    [ "$#" -ne 3 ]
then
    echo "there are not exactly three command line arguments" >&2
else
    echo $*
fi
```

6. Напишите shell процедуру, которая предлагает пользователю ввести ответ и в зависимости от ввода выполняет следующее:
- Если введено *A*, процедура выводит на экран приветствие *good morning*.
 - Если введено *B* или *b*, процедура выводит на экран приветствие *good afternoon*.
 - Если введено *C* или *quit*, процедура завершает свою работу.
 - Если введено что-то другое, на экран выводится сообщение об ошибке и процедура завершается с кодом 99.

Ответ:

```
echo "Please input A, B, b, or C: \c"
read input
case $input in
    A) echo good morning
        ;;
    [Bb]) echo good afternoon
        ;;
    C|quit) exit
        ;;
    *) echo You entered an illegal option.
        exit 99
        ;;
esac
```

Ответы и решения

7. Напишите shell процедуру, которая предлагает пользователю ввести некоторое входное имя, после чего проверяет, зарегистрирован ли в системе пользователь с таким именем. Если пользователь с таким именем в системе есть, процедура выводит на экран маршрутное имя *HOME* каталога этого пользователя, в противном случае – сообщение об ошибке.

Ответ:

```
echo "Input a user login name -> \c"
read user
if
    grep $user /etc/passwd > /dev/null
then
    home=$(grep $user /etc/passwd | cut -f6 -d:)
    echo The HOME directory for $user is $home
else
    echo;$echo "$user is not here!!!"
fi
```

8. С помощью команды **date** определите время суток: утро (до 12:00), день (между 12:00 и 18:00) или вечер (после 18:00). Напишите shell процедуру **greeting**, которая в зависимости от времени суток выводит на экран соответствующее приветствие: *good morning*, *good afternoon* или *good evening*. (ПОДСКАЗКА: команда **date** использует 24-х часовые часы)

Ответ:

```
time=$(date | cut -c12-20)
hour=$(echo $time | cut -f1 -d:)

if [ $hour -lt 12 ]
then
    echo good morning
else
    if [ $hour -ge 12 -a $hour -lt 18 ]
    then
        echo good afternoon
    else
        echo good evening
    fi
fi
```

9. Напишите shell процедуру, которая спрашивает у пользователя, хочет ли он увидеть содержимое текущего каталога. Процедура должна сообщить пользователю, что ожидает от него ответа *yes* или *no*, и вывести сообщение об ошибке, если пользователь введет что-то другое. Если пользователь ответит *yes*, процедура должна вывести содержимое текущего каталога. Если пользователь ответит *no*, процедура должна вывести на экран вопрос, содержимое какого каталога он хочет увидеть, ввести ответ пользователя и вывести содержимое указанного каталога. При этом сначала нужно проверить, существует ли такой каталог.

Ответ:

```
echo Would you like to see the contents of your current
directory?
echo Please enter yes or no.
echo "----> \c"
read ans1
case $ans1 in
    yes) ls
        ;;
    no) echo What directory would you like to see?
        read ans2
        if test -d $ans2
        then
            ls $ans2
        else
            echo directory $ans2 does not exist
        fi
        ;;
    *) echo You have not entered a proper response.
        echo Please try again.
        ;;
esac
```

10. Напишите процедуру **mycp**, которая копирует один файл в другой. Процедура должна иметь два аргумента, задающие файл-источник и файл-приемник. В процедуре должны проверяться следующие ситуации:

- файл-источник и файл-приемник не являются одним и тем же файлом;
- файл-приемник не является каталогом;
- файл-источник существует;
- если файл-приемник существует и является обычным файлом, необходимо спросить пользователя, действительно ли он хочет его перезаписать.

Ответ:

```
#!/usr/bin/sh
# mycp file1 file2
if [ "$#" -ne 2 ]
then
    echo "Usage: $0 file1 file2" >&2
    exit 1
fi
if [ "$1" = "$2" ]
then
    echo "$1 = $2"
    echo "No copy performed" >&2
    exit 2
fi
if test -d $2
then
    echo "Your target is a directory." >&2
    echo "No copy performed!" >&2
    exit 4
fi
```

Ответы и решения

```
if test -f $2
then
    echo "Your target file already exists."
    echo "Do you want to overwrite it? [y/n]: \c"
    read ans
    if [ "$ans" != "y" -a "$ans" != "Y" ]
    then
        echo "No copy performed!" >&2
        exit 3
    fi
fi

if test -f $1
then
    cp $1 $2
    echo "Copy complete"
else
    echo "Source file does not exist"
    echo "No copy performed"
    exit 4
fi
```

11. Напишите shell процедуру **options**, которая работает по следующим правилам, в зависимости от заданных аргументов командной строки:

- Если первый аргумент командной строки **-d**, выполнить команду **date**.
- Если первый аргумент командной строки **-w**, вывести список всех пользователей, работающих в настоящее время в системе.
- Если первый аргумент командной строки **-l**, вывести на экран содержимое каталога, имя которого определяет второй аргумент командной строки.
- Если в командной строке нет аргументов или присутствует более двух аргументов, вывести сообщение об ошибке и установить код завершения **10**.
- Если аргумент командной строки задает какую-либо другую опцию, вывести на экран соответствующее сообщение об ошибке и установить код завершения **20**.

Ответ:

```
if
[ "$#" -lt 1 -o "$#" -gt 2 ]
then
    echo "usage: $0 -d" >&2
    echo " $0 -l directory" >&2
    echo " $0 -w" >&2
    exit 10
fi
case $1 in
    -d) date
        ;;
    -w) who
        ;;
    -l) if test -d $2
        then
            echo the contents of directory $2 are:
            ls -F $2
        else
```

```
        echo directory $2 does not exist
    fi
    ;;
*) echo "bad option" >&2
   echo "usage: $0 -d" >&2
   echo " $0 -l directory" >&2
   echo " $0 -w" >&2
   exit 20
   ;;
esac
```

18–12. УПРАЖНЕНИЯ: Shell программирование: циклы

Указания

Выполните приведенные ниже упражнения и ответьте на поставленные вопросы.

1. Напишите shell процедуру **double_it**, которая предлагает пользователю ввести число и выводит на экран его удвоенное значение.

Ответ:

```
#!/bin/sh
# double_it: предлагает пользователю ввести число и затем выводит на экран
#             его удвоенное значение
#
echo "Input an integer value: \c"
read num
echo "Two times the number you entered is \c"
let num=num*2
echo $num
```

2. Напишите shell процедуру **sum_them**, которая предлагает пользователю ввести 10 чисел. Процедура должна вычислить сумму всех введенных чисел и вывести на экран полученный результат. (ПОДСКАЗКА: добавляйте к сумме каждое вновь введенное число.)

Если останется время: модифицируйте процедуру **sum_them** таким образом, чтобы количество чисел, сумму которых надо найти, передавалось в процедуру через аргумент командной строки. Например, процедура, вызванная так: **sum_them 6**, должна предложить пользователю ввести шесть чисел и затем вычислить их сумму.

Ответ:

```
#!/bin/sh
# sum_them: ввод и суммирование 10 чисел
#
sum=0
count=1
echo You will be prompted to enter 10 numbers.
echo Their sum will be displayed after all 10 numbers have been
entered.
while
    [ $count -le 10 ]
do
    echo "Please enter a number ($count): \c"
    read num
    let sum=sum+num
    let count=count+1
done
echo The sum of the 10 numbers you entered is: $sum
```

Решение дополнительной задачи, в которой аргумент командной строки определяет количество вводимых чисел:

```
#!/bin/sh
# sum_them2:      Количество чисел, которые надо ввести и сложить,
#                пользователь передает через аргумент командной строки
#
#
if
    [ $# -ne 1 ]
then
    echo Usage: $0 number >&2
    exit 99
fi

count=1
echo You will be prompted to enter $1 numbers.
echo Their sum will be displayed after all $1 numbers have been
entered.
while
    [ $count -le $1 ]
do
    echo "Please enter a number ($count): \c"
    read num
    let sum=sum+num
    let count=count+1
done
echo The sum of the $1 numbers you entered is: $sum
```

3. Напишите shell процедуру **words_in**, которая предлагает пользователю вводить слова до тех пор, пока не будет введено слово **quit**. Все введенные слова запоминаются. После того как будет введено слово **quit**, введенные слова в порядке, обратном вводу (последнее, предпоследнее, ..., первое) выводятся на экран. Можно ли выполнить это упражнение с помощью цикла **while**? С помощью цикла **until**? Выберите по своему усмотрению какой-нибудь из этих циклов.

Если останется время: выведите все введенные слова в алфавитном порядке.

Ответ:

```
#!/bin/sh
# words_in: ввод и обработка слов, пока не будет введено слово "quit"
#
until
    echo Please enter a word. Enter "quit" when you are done.
    read input
    [ "$input" = quit ]
do
    words="$words\n$input"
done
echo $words
# вывод слов в алфавитном порядке
echo $words | sort
```

4. Включите в shell процедуру цикл **for**, который:

- создает каталоги **Adir**, **Bdir**, **Cdir**, **Ddir**, **Edir**,
- копирует в каждый каталог файл **funfile**,

Ответы и решения

- выводит содержимое каждого каталога, чтобы проверить результаты копирования,
- после того как завершается каждая итерация цикла, выводит на экран соответствующее сообщение.

Ответ:

```
for name in Adir Bdir Cdir Ddir Edir
do
    mkdir $name
    cp $HOME/funfile $name
    ls $name
    echo done with $name
done
```

Другой вариант решения этой задачи:

```
for name in A B C D E
do
    mkdir ${name}dir
    cp $HOME/funfile ${name}dir
    ls ${name}dir
    echo done with $name
done
```

5. Напишите shell процедуру **new_files**, которая допускает задание произвольного количества аргументов командной строки. Эта shell процедура создает новые файлы, имена которых определяются аргументами командной строки (используйте для этого команду **touch**), и выводит на экран сообщение, информирующее пользователя о создании каждого файла.

Ответ:

```
#!/bin/sh
# new_files: создание новых файлов, заданных аргументами командной строки
# Usage: new_files f1 f2 f3 f4 ...
#
for i in $*
do
    echo creating file $i
    touch $i
done
```

6. С помощью редактора **vi** создайте файл **mailtest**. В ответ на приглашение shell'a создайте интерактивный цикл **for** для отправки почтового сообщения **mailtest** (с помощью команды **mail**) всем пользователям, работающим в системе. (ПОДСКАЗКА: для генерации списка для цикла **for** используйте команды **who** и **cut** и подстановку команд.)

Ответ:

```
$ for i in $(who | cut -f1 -d" ")
> do
>     mail $i < mailtest
> done
```

7. Напишите shell процедуру **my_menu**, которая выводит на экран простое меню, содержащее три альтернативы:

- a. Запуск на исполнение процедуры **double_it** (упражнение 1).
- b. Запуск на исполнение процедуры **sum_them** (упражнение 2).
- c. Выход.

Меню должно заново выводиться на экран каждый раз после выполнения действий, определенных первой и второй альтернативами.

Ответ:

```
#!/bin/sh
# my_menu: интерфейс в виде меню
# Usage: my_menu
#
until
[ "$ans" -eq 3 ]
clear
echo
echo
echo
echo 1) Double a number.
echo 2) Add together 10 numbers.
echo 3) Quit
echo
echo "Enter your selection number ->\c"
read ans
do
case $ans in
1) double_it
;;
2) sum_them
;;
3|quit|q|Q) exit
;;
*) echo You have not entered a legal option.
echo Please try again.
;;
esac
sleep 3
done
```

перед выводом меню экран очищается

8. Напишите shell процедуру **msg_me**, которая в течение одной минуты каждые 5 секунд выводит на экран сообщение. (ПОДСКАЗКА: посмотрите описание команды **sleep**.) Целесообразно сохранить сообщение, выводимое на экран, в файле, тогда его можно будет легко менять.

Ответ:

```
#!/bin/sh
# msg_me: вывод сообщения на экран каждые 5 секунд
#
count=1
while
[ $count -lt 12 ]
do
```

Ответы и решения

```
cat msg.file > $(tty)
sleep 5
let count=count+1
done
```

9. Напишите shell процедуру **ison**, которая будет *выполняться в фоновом режиме* и каждые 60 секунд будет проверять, работает ли в системе некоторый конкретный пользователь. Имя пользователя передается в процедуру **ison** с помощью аргумента командной строки. Если пользователь работает в системе, на экран следует вывести сообщение, содержащее входное имя пользователя и имя терминала, с которого он вошел в систему. (ПОДСКАЗКА: используйте команду **sleep**.)

Если вы работаете в сети, можете попробовать использовать команду **rwho**.

Ответ:

```
#!/bin/sh
# ison: проверка, работает ли пользователь в системе
#      Использование: ison имя_пользователя
#
if [ "$#" -ne 1 ]
then
    echo "usage: $0 user_id" >&2
    exit 99
fi

until who | grep $1 > /dev/null
do
    sleep 60
done

# Если управление передано в эту точку, значит, пользователь работает в системе

echo $1 has logged on
who | grep $1
```

10. Создайте в своем *HOME* каталоге каталог **.waste**. Напишите shell процедуру **myrm**, которая перемещает все удаляемые вами файлы в каталог **.waste** (он будет служить мусорной корзиной). Это полезный инструмент, позволяющий вам восстанавливать файлы после их удаления. Следует помнить, что система UNIX не имеет функции *отмены удаления*.

Пусть процедура **myrm** позволяет также задавать следующие опции:

-l вывести содержимое корзины
-d очистить корзину

Ответ:

```
#!/bin/sh
# myrm: корзина для мусора
#
if [ "$1" = "" ]
then
    echo "Usage: $0 file [file ...]" >&2
    echo " or: $0 [-l] | [-d]" >&2
```

```

        exit 5
    fi
    opt=$(echo $1 | cut -c1)
    if [ "$opt" = "-" ]
    then
        case $1 in
            -l) echo;echo "The WasteBasket includes the following
files:"
                ls $HOME/.waste;;
            -d) echo;echo "The WasteBasket is being dumped!"
                rm $HOME/.waste/*;;
            -*) echo "$0: $1 invalid argument" >&2
                exit;;
        esac
    else
        echo "Are you sure you want to remove $*? [y/n]: \c"
        read ans
        if [ "$ans" = "y" -o "$ans" = "Y" ]
        then
            for i in $*
            do
                if test -f $i
                then
                    mv "$i" $HOME/.waste
                else
                    echo "$i: Does not exist" >&2
                fi
            done
        else
            exit
        fi
    fi
fi

```

19–4. УПРАЖНЕНИЯ: Хранение файлов во внешней памяти

Указания

Выполните приведенные ниже упражнения. Запишите команды, которые вы используете для решения задач, указав в них в качестве файла устройства **/dev/rmt/0m**. Если доступен конкретный драйвер устройства, ваш инструктор сможет реально выполнить некоторые из этих команд. Вы можете заменить имя файла устройства **/dev/rmt/0m** в командах резервного копирования именем обычного файла, если вы хотите выполнить эти упражнения, не обращаясь к лентопротяжному устройству:

```
$ tar cf /tmp/archive_file mydir
```

1. Используя команду **tar**, создайте резервную копию всех файлов, находящихся в вашем *HOME* каталоге, имена которых начинаются буквами **abc**.

Ответ:

```
$ tar cf /dev/rmt/0m abc*
```

2. Получите оглавление созданного архива:

Ответ:

```
$ tar tf /dev/rmt/0m
```

3. Используя команды **find** и **cpio**, создайте резервную копию всех файлов, находящихся в вашем *HOME* каталоге и во всех его подкаталогах, сохранив структуру каталога.

Ответ:

```
$ cd
$ find . | cpio -ocv > /dev/rmt/0m
```

4. Удалите файл **backup** из вашего текущего каталога. Затем восстановите его из резервной копии с помощью команды **cpio**.

Ответ:

```
$ rm backup
$ cpio -iumc "backup" < /dev/rmt/0m
$ ll backup
```

5. Создайте каталог **\$HOME/tree.cp**. В статье **cpio(1)** найдите описание режима прохода (pass mode) команды **cpio**. Используя команду **cpio** в режиме прохода, воссоздайте структуру каталога **\$HOME/tree** в каталоге **\$HOME/tree.cp**.

Ответ:

```
$ mkdir $HOME/tree.cp
$ cd $HOME/tree
$ find . | cpio -pcduv $HOME/tree.cp
```