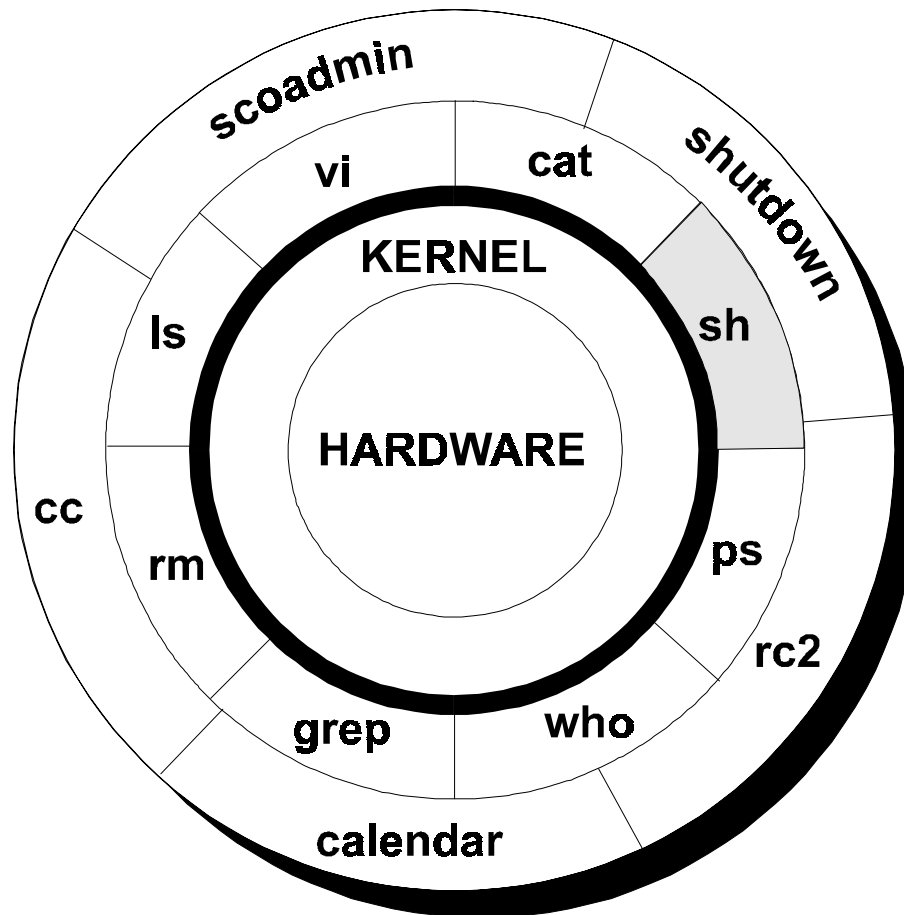


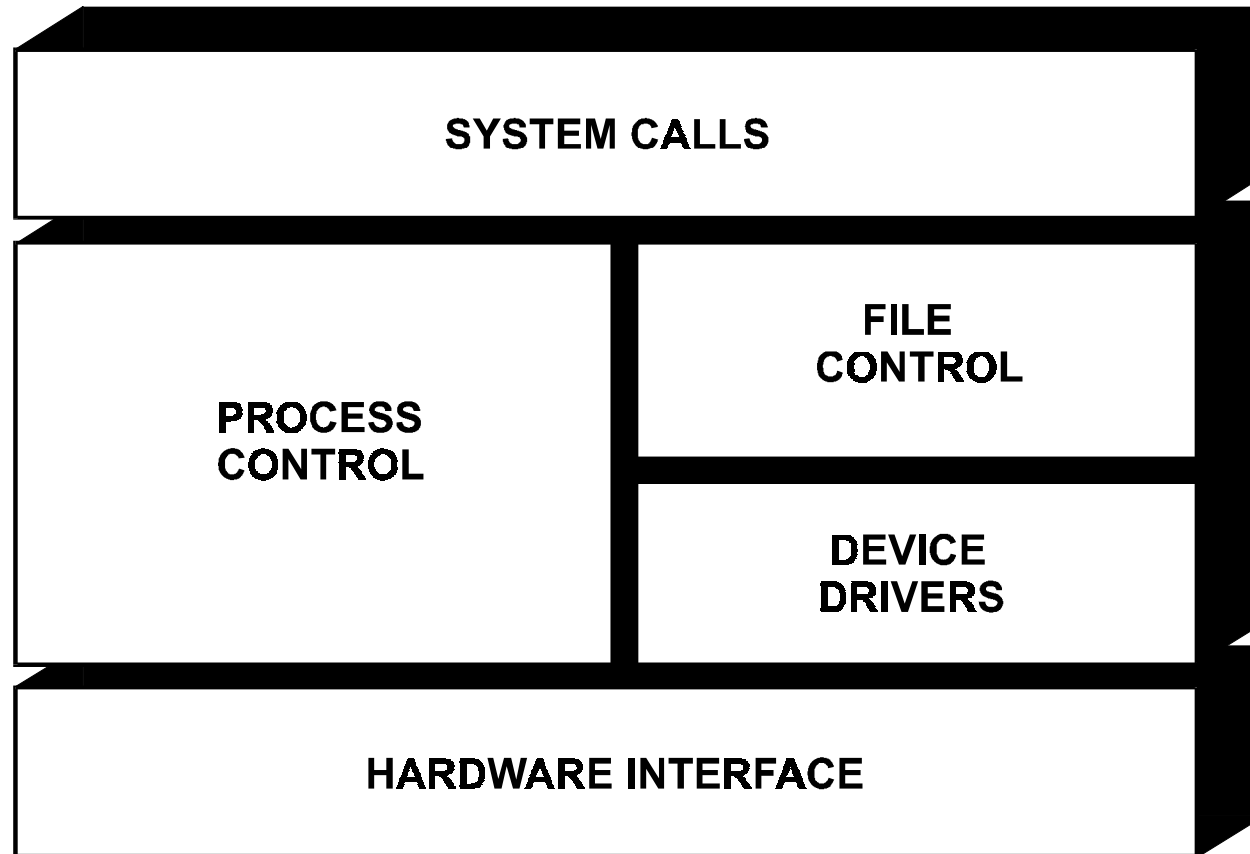
Shell Overview

- Differentiate kernel and shell features
- Use wildcard metacharacters for filename expansion
- Use shell I/O redirection mechanisms
- Use shell pipes
- Identify shell environment variables

UNIX Architecture



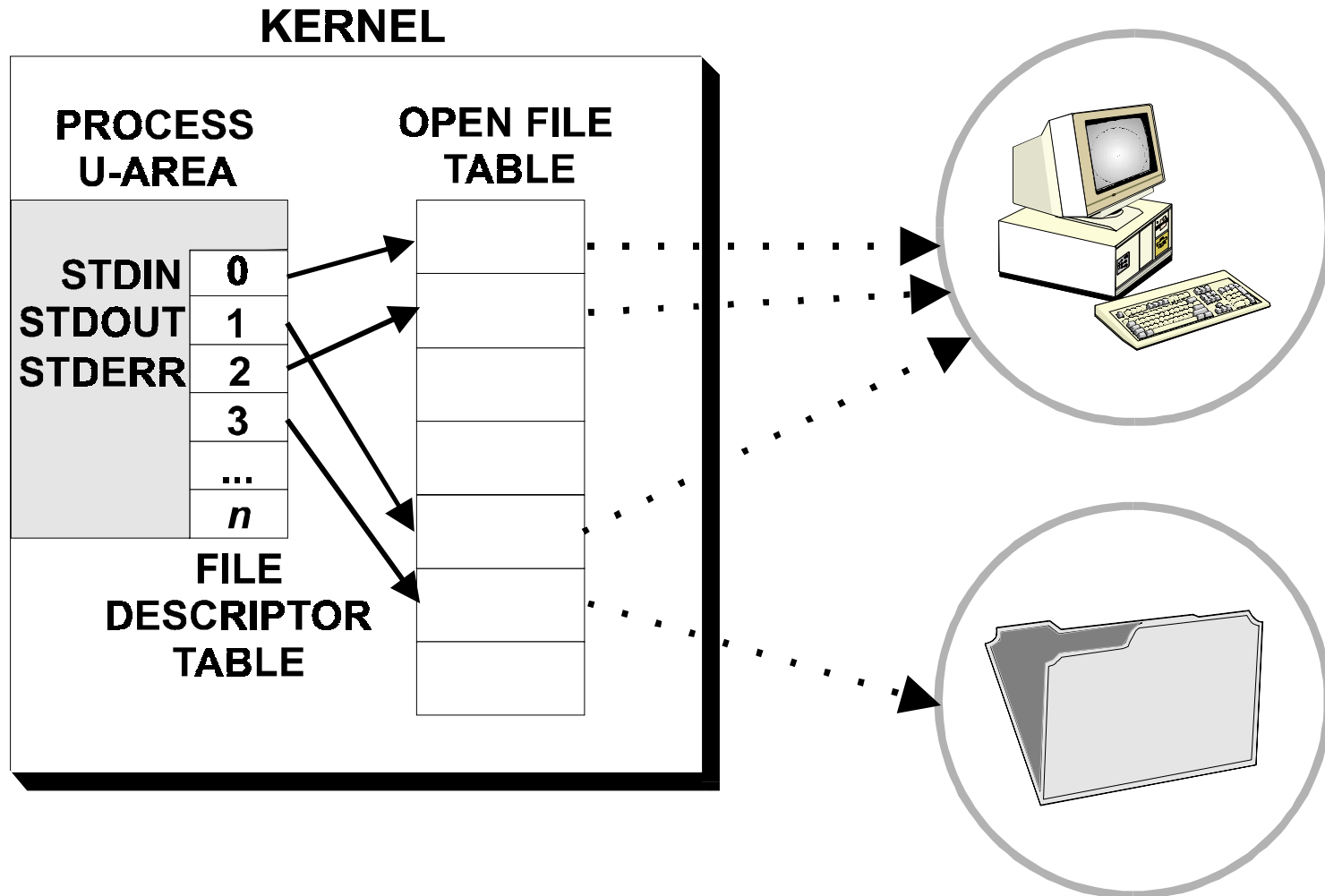
SCO UNIX Kernel



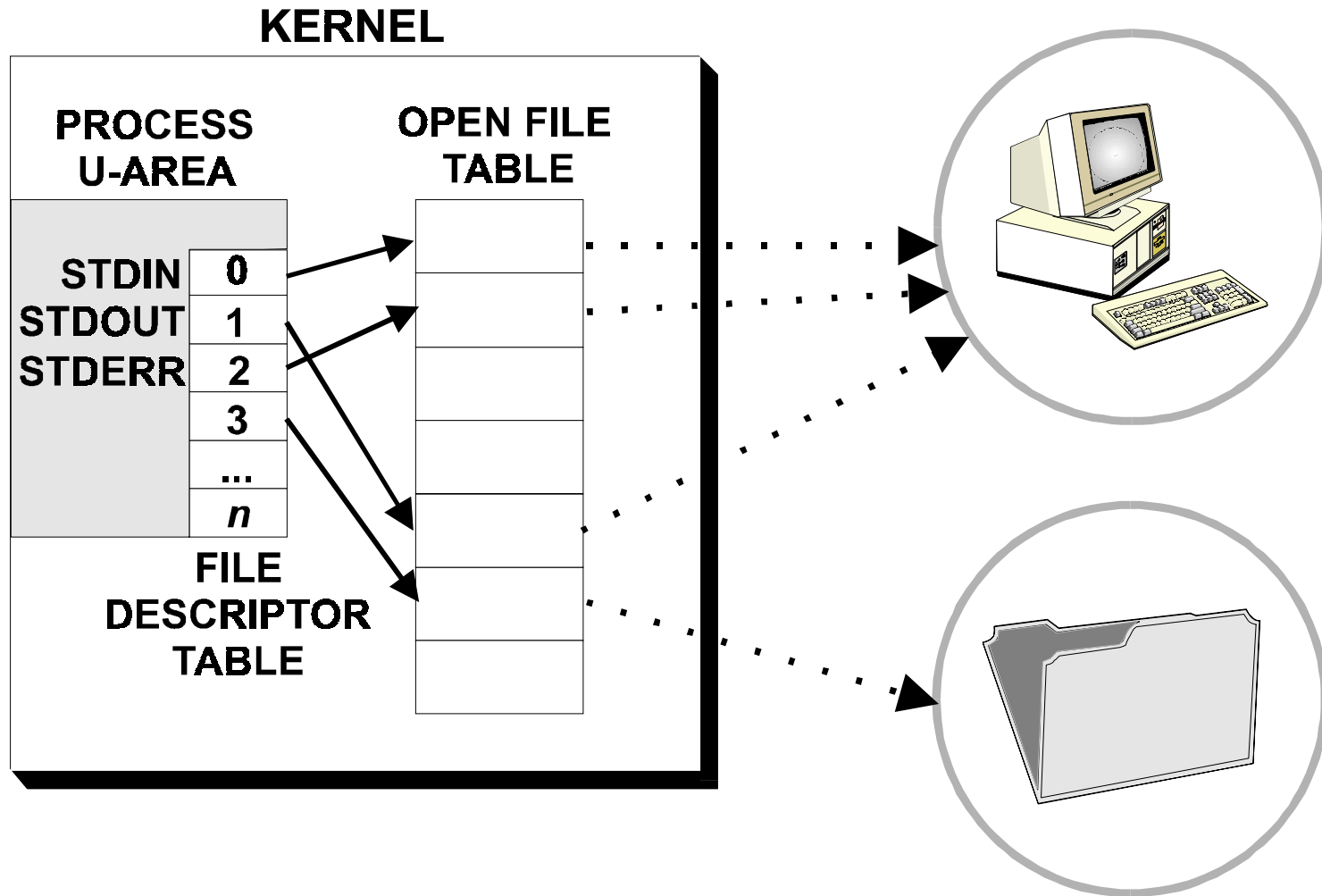
Shell Wildcard Metacharacters

*	Matches zero or more of any character
?	Matches exactly one of any character
[<i>string</i>]	Matches exactly one of any character in <i>string</i>
[<i>a-z</i>]	Matches exactly one of any character in the range <i>a-z</i>
[!<i>string</i>]	Matches exactly one of any character <i>not</i> in <i>string</i>

File Descriptors



File Descriptor Table

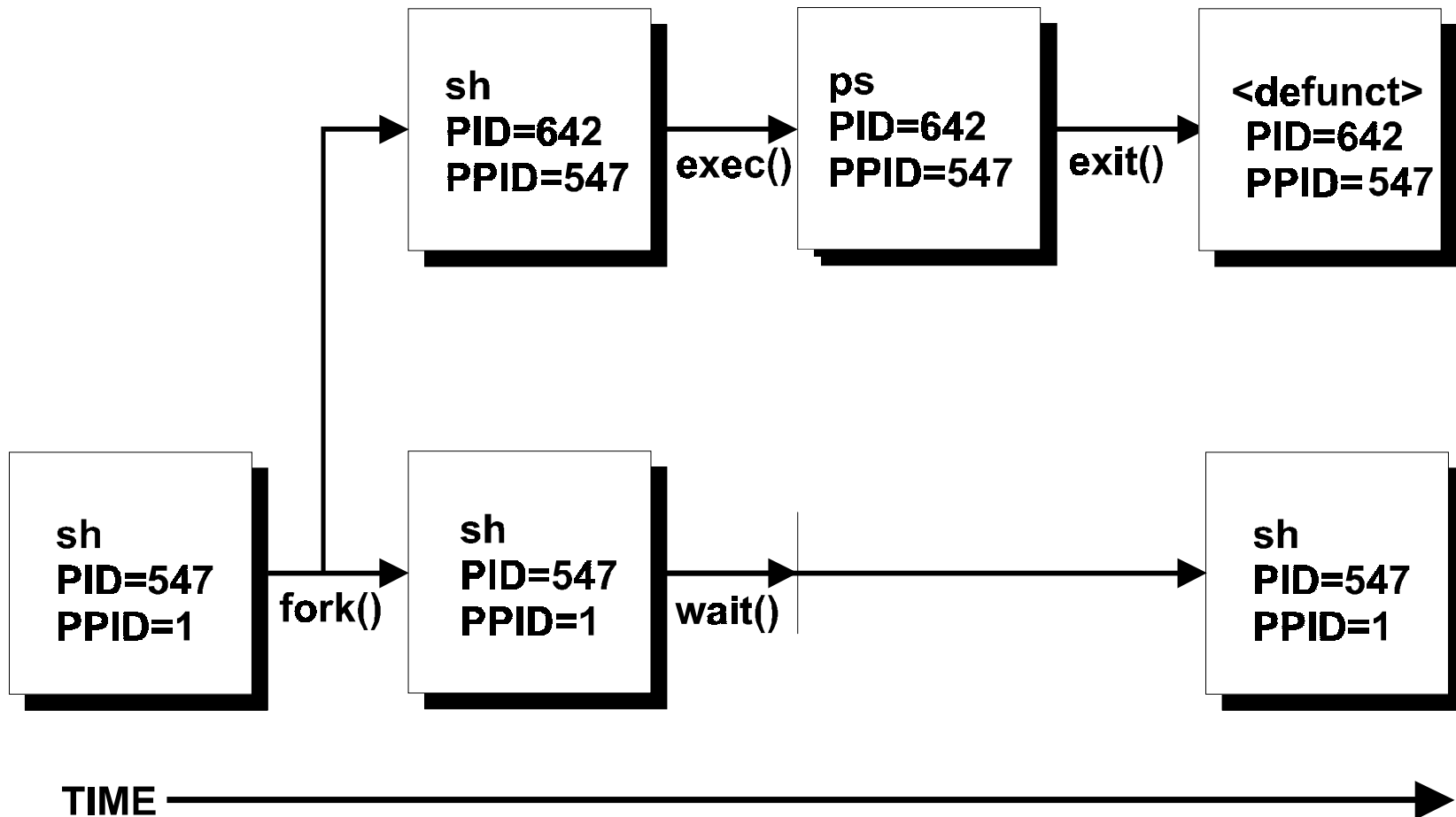


Command Execution

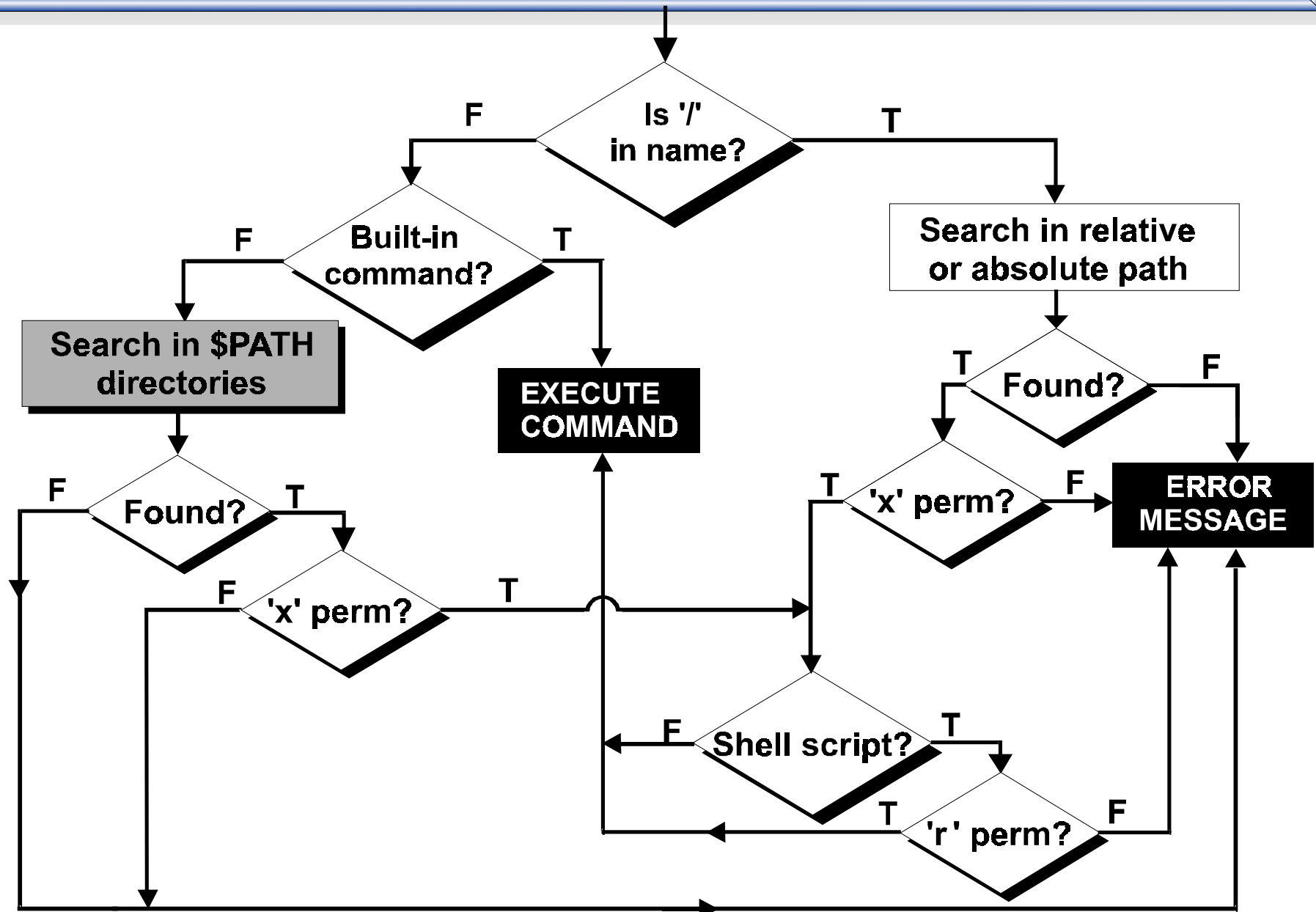
- Identify three UNIX command types
- Describe a UNIX process
- Describe how the shell locates commands
- Describe three command execution techniques

Command Execution

\$ ps



Finding Commands

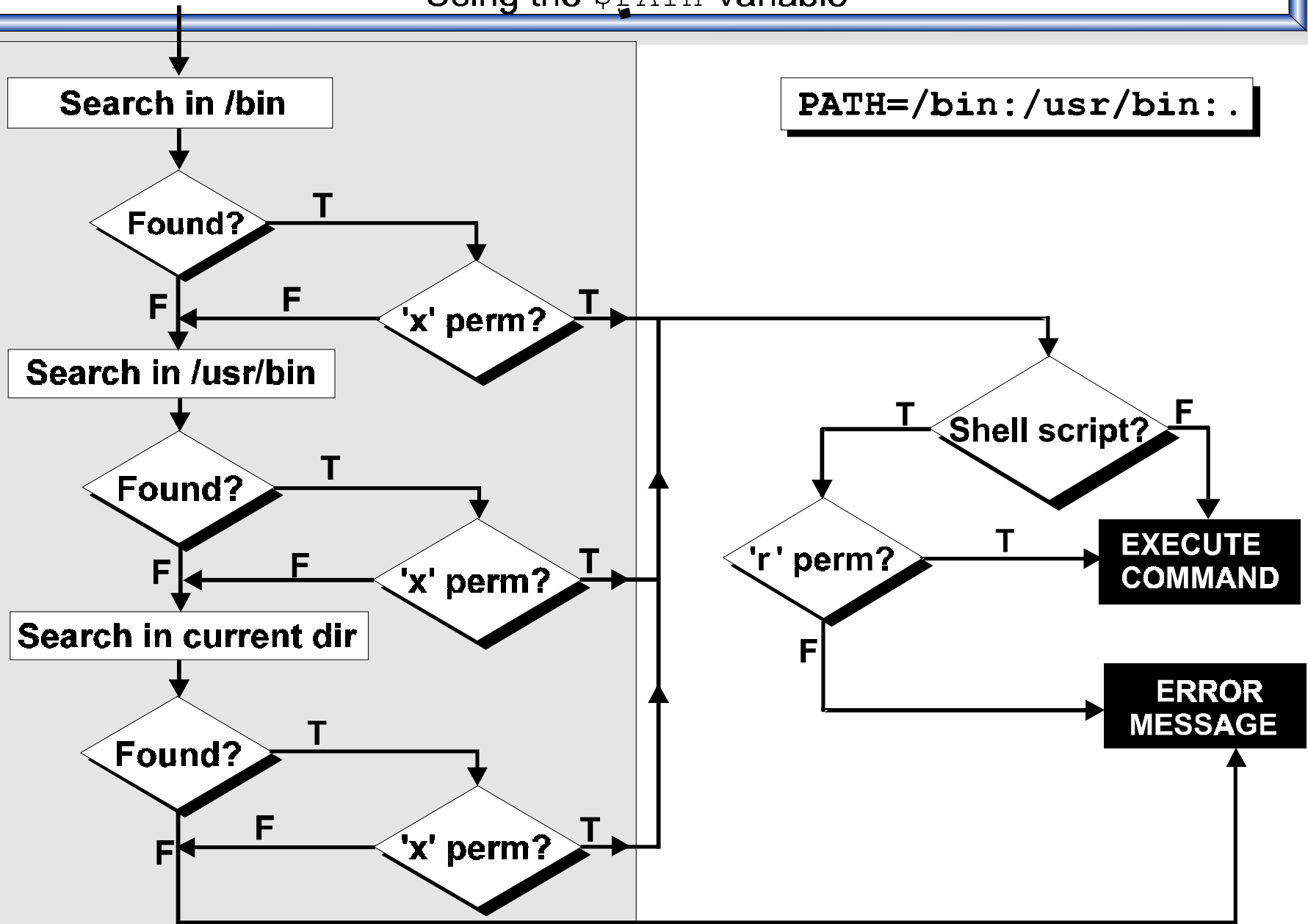


sp0303

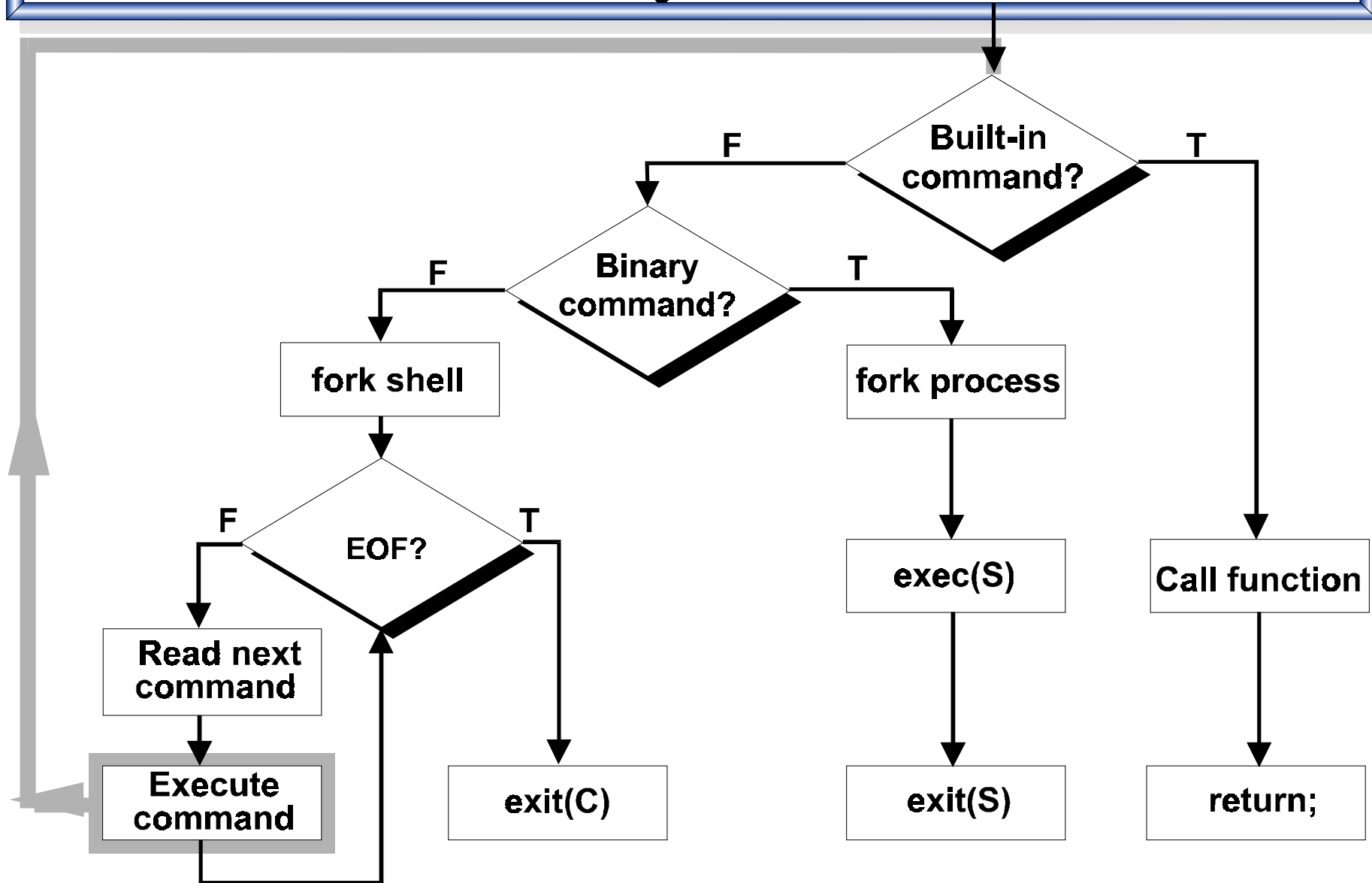
© The Santa Cruz Operation, Inc.

Using the \$PATH Variable

PATH=/bin:/usr/bin:.



Executing Commands



Basic Scripts

- Define the term *shell script*
- Create and execute a simple shell script
- Place comments in a shell script
- Specify which shell is to execute a script
- User the **echo** command
- Use shell options

Sample Shell Script

```
$ cat sample1
echo Today is
date
echo Calendar for this month:
cal
```

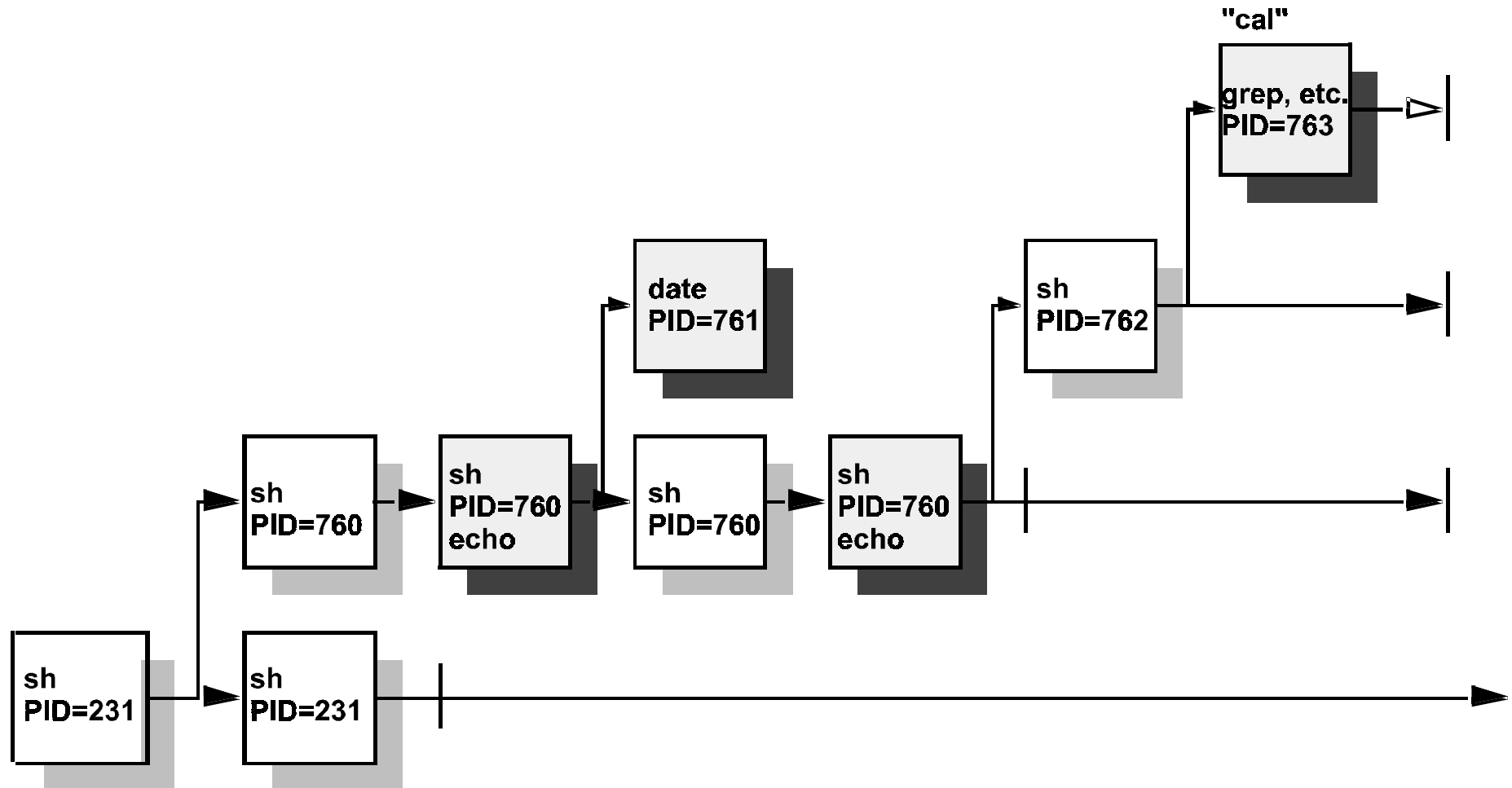
```
$ sh sample1
Today is
Sun Jun 28 19:52:46 EDT 1998
Calendar for this month:
```

```
    June 1998
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

```
$chmod +x sample1
ls -l sample1
-rwxr-wr-w  1 steve admin  53 Jun 28 19:56 sample1
```

```
$sample1
Today is
Sun Jun 28 19:52:46 EDT 1998
Calendar for this month:
    June 1998
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

Shell Script Execution



Shell Script: *sample1*

```
$ cat sample1  
echo Today is  
date  
echo Calendar:  
cal
```

Comments

```
$ echo one # two
one
$ echo one #two
one
$ echo one#two
one#two
```

echo Escape Sequences

\b	Backspace
\c	Print line without new line (equivalent to -n option)
\f	Formfeed
\n	Newline
\r	Carriage return (no linefeed)
\t	Tab
\v	Vertical tab
\\	Backslash
\0n	The character whose ASCII code is a 1, 2 or 3-digit octal number

Shell Variables

- Create and access user-defined variables
- Describe the shell's standard environment variables
- Use the **read** command to obtain input
- Use positional parameters in a shell script
- Define special variables

The **export** Command

```
$ FLOPPY=/dev/fd0135ds18  
$ PRINTER=laser5  
$ export FLOPPY
```

```
$ cat devicescript
```

```
#!/bin/sh
```

```
# Sample script to verify the effect of exported variables
```

```
echo The value of FLOPPY is $FLOPPY
```

```
echo The value of PRINTER is $PRINTER
```

```
# Rest of program which can use these variables can follow...
```

```
$ devicescript
```

```
The value of FLOPPY is /dev/fd0135ds18
```

```
The value of PRINTER is
```

Shell Environment Variables

```
HOME=/usr/steve
PATH=/bin:/usr/bin:/usr/local/bin:$HOME/bin:..
CDPATH=.:usr:/etc:$HOME:/
MAIL=/usr/spool/mail/steve
MAILPATH=/usr/spool/mail/ucp:/usr/spool/mail/cron
MAILCHECK=60
PS1='<root> '
PS2='> '
IFS=$IFS,
```

The **read** Command

```
$ cat printit
#!/bin/sh
# Friendlier command to print a file to a given printer

echo 'Which file do you want to print? ==> \c'
read FILE DUMMY

echo Default printer information:
lpstat -d

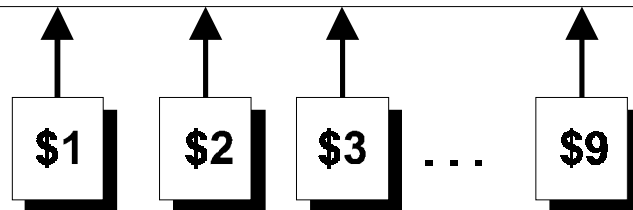
echo To which printer do you want to send the file ${FILE}?
echo '==> \c'
read PRINTER DUMMY

echo Sending $FILE to $PRINTER
lp -d$PRINTER $FILE

$ printit
Which file do you want to print? ==> /etc/motd
Default printer information:
system default destination: curdev
To which printer do you want to send the file /etc/motd?
==> laser5
Sending /etc/motd to laser5
request id is laser5-20087 1 file
```

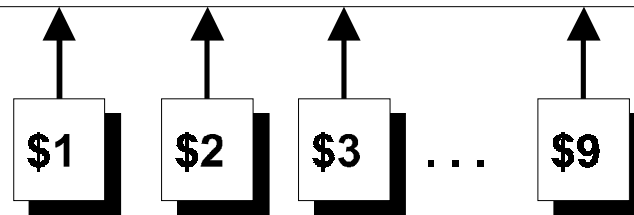
Positional Parameters

```
$ mycommand arg1 arg2 arg3 ... arg9 arg10 arg11 arg12 ...
```



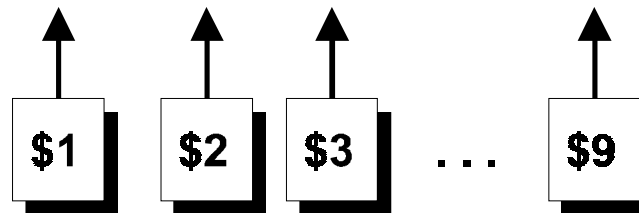
The **shift** Command

```
$ mycommand arg1 arg2 arg3 ... arg9 arg10 arg11 arg12 ...
```



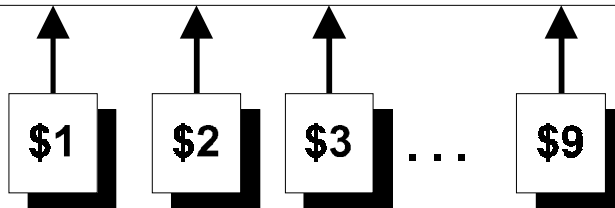
shift

```
mycommand arg1 arg2 arg3 ... arg9 arg10 arg11 arg12 ...
```

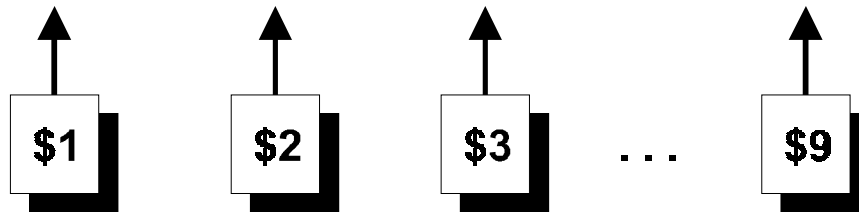


The **set** Command

```
$ mycommand arg1 arg2 arg3 ... arg9 arg10 arg11 arg12 ...
```



```
set newarg1 newarg2 newarg3 ... newarg9 newarg10 newarg11 ...
```



Special Shell Variables

<code>\$0</code>	Command name
<code>\$#</code>	Number of positional parameters
<code>\$*</code>	All positional parameters
<code>@</code>	All positional parameters
<code>?</code>	Exit status of last foreground command
<code>\$\$</code>	PID of current shell
<code>!</code>	PID of last background process
<code>-</code>	Flags set in the current shell

Quoting

- Identify shell metacharacters
- Use quoting mechanisms
- Use the shell command substitution mechanism
- Describe the shell command line parsing sequence
- Use the **eval** command to reparse a line

Bourne Shell Metacharacters

#	Comment
\$	Shell variable
and ^	Pipe
< and >	I/O redirection
&	Background execution
*, ?, and []	Filename wildcards
() and { }	Command grouping
', ", and \	Quoting characters
`	Command substitution
Tab and Space	Internal field separator
Enter	Internal field separator and command terminator
;	Command terminator

Single Quotes

```
$ ls
mailfolder
memo
report
$ find . -name m* -print
find: missing conjunction
$ find . -name 'm*' -print
./memo
./mailfolder
```

```
$ echo Home is $HOME
Home is /usr/steve
$ echo 'Home is $HOME'
Home is $HOME
```

```
$ PS1=<steve>
syntax error: `newline or ;' unexpected
$ PS1='<steve> '
<steve>
<steve> echo one      two
one two
<steve> echo 'one      two'
one      two
```

Double Quotes

```
$ find . -name "m*" -print
./memo
./mailfolder
```

```
$ echo Home is $HOME
Home is /usr/steve
$ echo 'Home is $HOME'
Home is $HOME
$ echo "Home is $HOME"
Home is /usr/steve
```

```
$ VAR='one      two'
$ echo VAR is $VAR
VAR is one two
$ echo "$VAR"
VAR is one      two
```

```
$ VAR='two
> lines'
$ echo VAR is $VAR
VAR is two lines
$ echo "VAR is $VAR"
VAR is two
lines
```

Backslash

```
$ find . -name m'*' -print
./memo
./mailfolder
$ find . -name m\* -print
./memo
./mailfolder

$ echo two'
> 'words
two
words
$ echo two\
> words
twowords

$ TAPE=/dev/rct0
$ echo "TAPE is \$TAPE"
TAPE is $TAPE

$ echo "Backslash: \\"
Backslash: \
$ echo "Double Quote: \""
Double Quote: "
$ echo "Asterisk: \*"
Asterisk: \*
$ echo "Percent: \%"
Percent: \%
```

Quoting Exceptions

Quote Mechanism

Single quotes ('..')

Double quotes ("...")

Backslash (\)

Metacharacters not affected

Other single quotes

Other double quotes

Dollar (\$)

Backslash (\)

Backquote (`)

Enter ()

Command Substitution

```
$ echo "The date is:    <`date`>"
```

```
The date is:    <Wed Oct 05 10:26:43  
PDT 1994>
```

```
$ CWD=`pwd`
```

```
$ cd /some/directory
```

```
$ cd $CWD
```

```
$ cat editnames
```

```
memo  
report  
letter
```

```
$ vi `cat editnames`
```

```
$ GREETING=`banner hello`
```

```
$ echo $GREETING
```

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

```
$ echo "$GREETING"
```

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

Parsing a Command Line (1)

```
$ set -x
$ echo This is `pwd` : * > $HOME/$$.tmp
+pwd
+echo This is /home/gen : README README.general
fixup maillist message1 setup title
$
$ lf $HOME/*.tmp
/home/chris/1071.tmp
$
$ cat /home/chris/1071.tmp
This is /home/gen : README README.general
fixup maillist message1 setup title
```

Parsing a Command Line (2)

```
$ set -x
$ echo `id | sed 's/[^a-z0-9=].*//'\` > userid
+sed 's/[^a-z0-9=].*//'
+id
+echo uid=0

$ cat userid
uid=0
```

The eval Command

```
$ PIPE='|'  
$ ls $PIPE wc  
| not found  
wc not found  
$ eval ls $PIPE wc  
4 4 29
```

```
$ tset -s  
export TERM;  
TERM=ansi;  
$ `tset -s`  
TERM:: is not an identifier  
$ eval `tset -s`  
$
```

Coding a Shell Escape

```
$ cat shellescape1
#!/bin/sh
# Sample code to demonstrate how to do a "shell escape"
...
echo "Command? ==> \c"
`line`
...

$ shellescape1
Command? ==> ls -l > logfile
> not found
logfile not found

$ cat shellescape2
#!/bin/sh
# Even better sample code to demonstrate how to do a "shell escape"
...
echo "Command? ==> \c"
eval `line`
...

$ shellescape2
Command? ==> ls -l > logfile
$
```

Decision Constructs

- Define the conventional successful and unsuccessful error codes
- Use the **exit** command to terminate or abort a shell script
- Use the **test** command
- Use the **if** statement to conditionally execute command lines
- Use the **case** statement to conditionally execute command lines
- Use special shell constructs for conditional command execution

Testing Files

`test option file`

or

`[ option file ]`

Option

Description

<code>-r</code>	True if <i>file</i> exists and is readable
<code>-w</code>	True if <i>file</i> exists and is writable
<code>-x</code>	True if <i>file</i> exists and is executable
<code>-f</code>	True if <i>file</i> exists and is a regular file
<code>-d</code>	True if <i>file</i> exists and is a directory
<code>-h</code> or <code>-L</code>	True if <i>file</i> exists and is a symbolic link
<code>-c</code>	True if <i>file</i> exists and is a character special file
<code>-b</code>	True if <i>file</i> exists and is a block special file
<code>-u</code>	True if <i>file</i> exists and its set-user-ID bit is set
<code>-g</code>	True if <i>file</i> exists and its set-group-ID bit is set
<code>-k</code>	True if <i>file</i> exists and its sticky bit is set
<code>-s</code>	True if <i>file</i> exists and has a size greater than zero
<code>-t</code> <i>n</i>	True if the open file whose file descriptor number is <i>n</i> (1 by default) is associated with a terminal device

Testing Character Strings

```
test string1 option string2      [ string1 option string2 ]  
test option string1               or [ option string1 ]  
test string1                       [ string1 ]
```

Option

Description

=

True if *string1* and *string2* are identical

!=

True if *string1* and *string2* are not identical

-z

True if the length of *string1* is zero

-n

True if the length of *string1* is non-zero

(no option)

True if *string1* is not the null string (-n alternate)

Testing Integer Expressions

test *string1* option *string2*

or

[*string1* option *string2*]

Option

Description

-eq

True if *string1* is algebraically equal to *string2*

-ne

True if *string1* is algebraically not equal to *string2*

-lt

True if *string1* is algebraically less than *string2*

-le

True if *string1* is algebraically less than or equal to *string2*

-gt

True if *string1* is algebraically greater than *string2*

-ge

True if *string1* is algebraically greater than or equal to *string2*

The if Statement

```
if test "$ANSWER" = quit
then
    echo "Quitting program at your request."
    exit 0
fi

if [ $# -lt 2 ]
then
    echo "$0: Insufficient arguments ($#)"
    echo "Usage: $0 f1 f2"
    echo "$0 f1 ... fn d1"
    exit 1
fi >&2

if mkdir "$NEWDIR"
then
    cd "$NEWDIR"
fi

if who | grep "$1" > /dev/null 2>&1
then
    write "$1" <<- EOF
        Please call me ASAP!
EOF
```

The **else** Statement

```
if mkdir "$NEWDIR"
then
    cd "$NEWDIR"
else
    echo "Cannot create directory $NEWDIR" >&2
    exit 1
fi

if who | grep "$1" > /dev/null 2>&1
then
    write "$1" <<- EOF
        Please call me ASAP!
EOF
else
    mail -s "Please call" "$1" <<- EOF
        Please call me ASAP!
EOF
fi

if [ -d "$FILE" ]
then
    rmdir "$FILE"
else
    rm "$FILE"
fi
```

The elif Statement

```
if mkdir "$NEWDIR"
then
    cd "$NEWDIR"
elif mkdir "/tmp/$NEWDIR"
then
    cd "/tmp/$NEWDIR"
else
    echo "Cannot create directory" >&2
    exit 1
fi

if [ "$MENUITEM" = c -o "$MENUITEM" = C ]
then
    cat "$FILE"
elif [ "$MENUITEM" = m -o "$MENUITEM" = M ]
then
    more "$FILE"
elif [ "$MENUITEM" = l -o "$MENUITEM" = L ]
then
    lp "$FILE"
elif [ "$MENUITEM" = r -o "$MENUITEM" = R ]
then
    rm -i "$FILE"
else
    echo "Unknown menu choice" >&2
```

Nested if Statements

```
echo "Do you want to format the disk now? [y/n] \c"
read REPLY1 DUMMY
if [ "$REPLY1" = y ]
then
    echo "Are you sure? [y/n] \c"
    read REPLY2 DUMMY
    if [ "$REPLY2" = y ]
    then
        echo "Last chance. Are you really, really sure? [y/n] \c"
        read REPLY3 DUMMY
        if [ "$REPLY3" = y ]
        then
            echo Formatting...
            format "$DISK"
        fi
    fi
fi
else
    echo "Installation cannot proceed at this time. Please
    backup the data on the disk and try again later."
    exit 1
fi
```

Nested if Statements

```
echo "Do you want to format the disk now? [y/n] \c"
read REPLY1 DUMMY
if [ "$REPLY1" = y ]
then
    echo "Are you sure? [y/n] \c"
    read REPLY2 DUMMY
    if [ "$REPLY2" = y ]
    then
        echo "Last chance. Are you really, really sure? [y/n] \c"
        read REPLY3 DUMMY
        if [ "$REPLY3" = y ]
        then
            echo Formatting...
            format "$DISK"
        fi
    fi
fi
else
    echo "Installation cannot proceed at this time. Please
    backup the data on the disk and try again later."
    exit 1
fi
```

The **case** Statement

```
#!/bin/sh
# Sample program which demonstrates menus with a case statement

echo "Enter filename ==> \c"
read FILE DUMMY

echo "FILE MENU
-----
c)at filename
m)ore filename
l)ine printer filename
r)emove filename
Enter choice ==> \c"

read MENUITEM DUMMY
case "$MENUITEM" in
    c | C ) cat "$FILE"
            ;;
    m | M ) more "$FILE"
            ;;
    l | L ) lp "$FILE"
            ;;
    r | R ) rm -i "$FILE"
            ;;
    * )      echo "\07Unknown menu choice" >&2
            exit 1
            ;;
esac
```

The **&&** and **||** Constructs

```
[ -r report -a -w report ] && vi report  
mkdir /tmp/dir$$ && cd /tmp/dir$$ && PS1="`pwd` "  
mkdir /tmp/dir$$ || echo "Cannot create /tmp/dir$$" >&2  
write jeff < message || mail -s 'Read this' jeff < message
```

The **&&** and **||** Constructs

```
[ -r report -a -w report ] && vi report  
mkdir /tmp/dir$$ && cd /tmp/dir$$ && PS1="`pwd` "  
mkdir /tmp/dir$$ || echo "Cannot create /tmp/dir$$" >&2  
write jeff < message || mail -s 'Read this' jeff < message
```

Looping Constructs

- Use the **expr** command
- Use **while** and **until** statements
- Use a **for** statement
- Control loop execution

The **expr** Command

```
$ expr 2 + 13
```

```
15
```

```
$ NUMBER=42
```

```
$ expr "$NUMBER" / 5
```

```
8
```

```
$ expr "$NUMBER" % 5
```

```
2
```

```
$ expr "$NUMBER" % 6
```

```
0
```

```
$ expr "$NUMBER" % 60
```

```
42
```

```
$ expr 7 + 3 \* 8
```

```
31
```

```
$ expr \( 7 + 3 \) \* 8
```

```
80
```

Variable Arithmetic

```
$ COUNT=10  
$ COUNT=$COUNT+1  
$ echo $COUNT  
10+1
```

```
$ COUNT=10  
$ COUNT=`expr $COUNT + 1`  
$ echo $COUNT  
11
```

The **while** Statement

```
#!/bin/sh
# Send a file multiple times to a printer with no formfeed
# between copies. (Useful for label printers)
# Syntax: lplabel ncopies filename

if [ $# -ne 2 ]
then
    echo "Syntax: $0 ncopies filename" >&2
    exit 1
fi

> tmp$$ # Create temporary file
COUNT=0

while [ "$COUNT" -lt $1 ]
do
    cat "$2" >> tmp$$
    COUNT=`expr "$COUNT" + 1`
done

lp tmp$$
rm tmp$$
```

Nested while Statements

```
#!/bin/sh
# Display several files with line numbers
# Syntax: number file ...
while [ $# -gt 0 ]
do
    LINE_NUM=1
    cat $1 | while read LINE
    do
        echo "$LINE_NUM\t$LINE"
        LINE_NUM=`expr $LINE_NUM + 1`
    done

    shift

    if [ $# -gt 0 ]
    then
        echo "Press <Enter> to continue..."
        read DUMMY
    fi
done
```

The **until** Statement

```
#!/bin/sh
# Call a user as soon as he or she logs in
# Syntax: caller username

if [ $# -ne 1 ]
then
    echo "Syntax: $0 username" >&2
    exit 1
fi

until who | grep "^$1 " > /dev/null 2>&1
do
    echo '.\c'
    sleep 2
done

# The user has logged in. Allow time to complete login sequence.
sleep 30

write $1 < $MSG_FILE
```

The **for** Statement

```
$ for USER in bonnie charlie david jeff steve sue
> do
>     write $USER << EOF
> Don't forget the meeting starts in 5 minutes. Don't be late!
> EOF
> done
```

```
$ lc
letter memo      report status
$ mv * *.txt
mv: *.txt not found
$ for FILE in *
> do
>     mv $FILE ${FILE}.txt
> done
$ lc
letter.txt memo.txt      report.txt status.txt
```

Positional Parameters and **for** Loops

```
$ cat sample_for_1
#!/bin/sh
# Sample for-loop using positional parameters

echo The number of positional parameters is $#

for NAME in $*
do
    echo The next name is $NAME
done

$ sample_for_1 mary ellen pete
The number of positional parameters is 3
The next name is mary
The next name is ellen
The next name is pete

$ sample_for_1 "mary ellen" pete
The number of positional parameters is 2
The next name is mary
The next name is ellen
The next name is pete
```

Using \$@ in a for Loop

```
$ cat sample_for_2
#!/bin/sh
# A better sample for-loop using positional parameters

echo The number of positional parameters is $#

for NAME in "$@"
do
    echo The next name is $NAME
done

$ sample_for_2 mary ellen pete
The number of positional parameters is 3
The next name is mary
The next name is ellen
The next name is pete

$ sample_for_2 "mary ellen" pete
The number of positional parameters is 2
The next name is mary ellen
The next name is pete
```

The **break** and **continue** Statements

```
#!/bin/sh
# Rename files with standard extensions
# Syntax: rename file(s)

for FILE
do
    echo "With which extension do you want to rename $FILE?"
    \tt) \".txt\" \td) \".doc\" \tr) \".rpt\"
    \tn) no extension? (q to quit) ==> \c"
    read REPLY DUMMY

    case "$REPLY" in
        [tT] )      EXT=txt ;;
        [dD] )      EXT=doc ;;
        [rR] )      EXT=rpt ;;
        [nN]* )      continue ;;
        [qQ]* )      break ;;
        * )          echo "Invalid response. Bye!" >&2
                    exit 1 ;;
    esac

    mv "$FILE" "$FILE.$EXT" && echo "$FILE renamed to $FILE.$EXT"

    LIST="$LIST $FILE.$EXT"
done

echo "Summary: These files now have new names:"
ls -l  LIST
```

The **break** and **continue** Statements

```
#!/bin/sh
# Rename files with standard extensions
# Syntax: rename file(s)

for FILE
do
    echo "With which extension do you want to rename $FILE?"
    \tt) \".txt\" \td) \".doc\" \tr) \".rpt\"
    \tn) no extension? (q to quit) ==> \c"
    read REPLY DUMMY

    case "$REPLY" in
        [tT] )      EXT=txt ;;
        [dD] )      EXT=doc ;;
        [rR] )      EXT=rpt ;;
        [nN]* )      continue ;;
        [qQ]* )      break ;;
        * )          echo "Invalid response. Bye!" >&2
                    exit 1 ;;
    esac

    mv "$FILE" "$FILE.$EXT" && echo "$FILE renamed to $FILE.$EXT"

    LIST="$LIST $FILE.$EXT"
done

echo "Summary: These files now have new names:"
ls -l  LIST
```

Shell Functions

- Describe the execution of a shell function
- Pass arguments to a shell function
- Use the **return** statement

Shell Functions

```
$ WHO ()  
> {  
>     who | sort  
> }
```

```
$ who  
jeff          tty02          Oct 19 09:46  
donna         tty04          Oct 24 09:11  
steve         tty02          Oct 05 13:43  
bonnie        tty03          Oct 19 08:19  
donna         tty09          Oct 24 09:11
```

```
$ WHO  
bonnie        tty03          Oct 19 08:19  
donna         tty04          Oct 24 09:11  
donna         tty09          Oct 24 09:11
```

Viewing Functions

```
$ set  
HOME=/usr/steve  
...  
TERM=ansi  
WHO() {  
who | sort  
}  
  
$ type WHO  
WHO is a function  
WHO() {  
who | sort  
}
```

Exporting Functions

```
$ cat func_test
:
# test whether the function "WHO" is available
echo "About to call WHO..."
WHO
```

```
$ func_test
About to call WHO...
bonnie      ttyp3      Oct 19 08:19
donna       ttyp4      Oct 24 09:11
donna       ttyp9      Oct 24 09:11
jeff        tty02      Oct 19 09:46
steve       tty2       Oct 05 13:43
```

```
$ sh func_test
About to call WHO...
func_test: WHO: not found
```

Function Arguments

```
$ CDP () { cd $1; pwd; }
```

```
$ pwd
```

```
/usr/steve/bin
```

```
$ CDP ../../lib
```

```
/usr/lib
```

```
$ del ()
```

```
> {
```

```
> echo Trying to delete $*
```

```
> rm -i "$@"
```

```
> }
```

```
$ del [lmr]*
```

```
Trying to delete letter memo report
```

```
letter: ? n
```

```
memo: ? y
```

```
report: ? n
```

```
$ set arg1 arg2 arg3
```

```
$ echo $*
```

```
arg1 arg2 arg3
```

Using the **return** Statement (Part 1)

```
$ cat myfunctions
getyn()
{
while echo "${*}? [y/n] ==> \c" >&2
do
    read REPLY DUMMY
    case "$REPLY" in
        [yY]* ) return 0 ;;
        [nN]* ) return 1 ;;
        * ) echo "\07Please enter <y> or <n>..." ;;
    esac
done
}

abort()
{
    # set default error message"
    [ $# -eq 0 ] && set "Aborting program.  See ya!"
    echo "\07*** $*" >&2
    exit 2
}
```

The **return** Statement (Part 2)

```
$ cat rm+mkdir
#!/bin/sh
# Remove a directory file and replace it with a new one
# Syntax: $0 dirname

. myfunctions

if getyn "Delete $1 file?"
then
    if getyn "Cannot be undone. Are you sure?"
    then
        rm -r $1
    else
        abort
    fi
fi

echo "Please enter new name for directory file ==> \c"
read NAME DUMMY
getyn "Is $NAME correct?" || abort "Please try again another time..."
echo "$NAME" | grep '[A-Z]' && abort "Please use lower case characters only"

mkdir "$NAME"
```

Shell Signal Handling

- Define the term signal
- Identify sources of signals
- Use the shell trap command

Catching Signals

```
#!/bin/sh
# Template which shows characteristics of trap command

# At this point, any incoming signal has the default action
...
# Now set a trap for the SIGINT (2) signal
trap 'rm tmp$$; exit 1' 2
# Also set a "cleanup" trap
trap 'rm tmp$$; exit 0' 0
...
# Critical code coming up which should not be interrupted.
# Completely ignore the SIGINT (2) signal
trap '' 2
...
# Out of critical section. It no longer matters what happens if
# SIGINT (2) is sent. Reset signal to system default action.
trap 2
...
# Now at end of program. The trap for "signal" 0 is still in
# effect, so the temporary file will be automatically removed now.
exit 0
```

Efficiency Considerations

- Use the **time** command
- Redefine the `PATH` variable
- Use absolute and relative pathnames effectively
- Identify the shell's built-in commands
- Identify efficient use of pipelines

The **time** Command

```
$ time whodo > output
```

```
real      1.1
user      0.2
sys       0.2
```

```
$ time format /dev/rfd0
```

```
real      2:26.12
user      0.06
sys       0.74
```

```
$ time tar cv6L /etc/default > tar.log
```

```
real      6.67
user      0.06
sys       0.74
```

```
$ time (find /etc/default | cpio -ocvBL -O/dev/fd0 > cpio.log)
```

```
real      4.08
user      0.01
sys       0.07
```

Absolute vs. Relative Pathnames

```
$ time grep BUF /usr/include/sys/[m-z]*.h > $HOME/data
```

```
real          3.0  
user          0.7  
sys           0.5
```

```
$ cd /usr/include/sys
```

```
$ time grep BUF [m-z]*.h > $HOME/data
```

```
real          1.2  
user          0.7  
sys           0.2
```

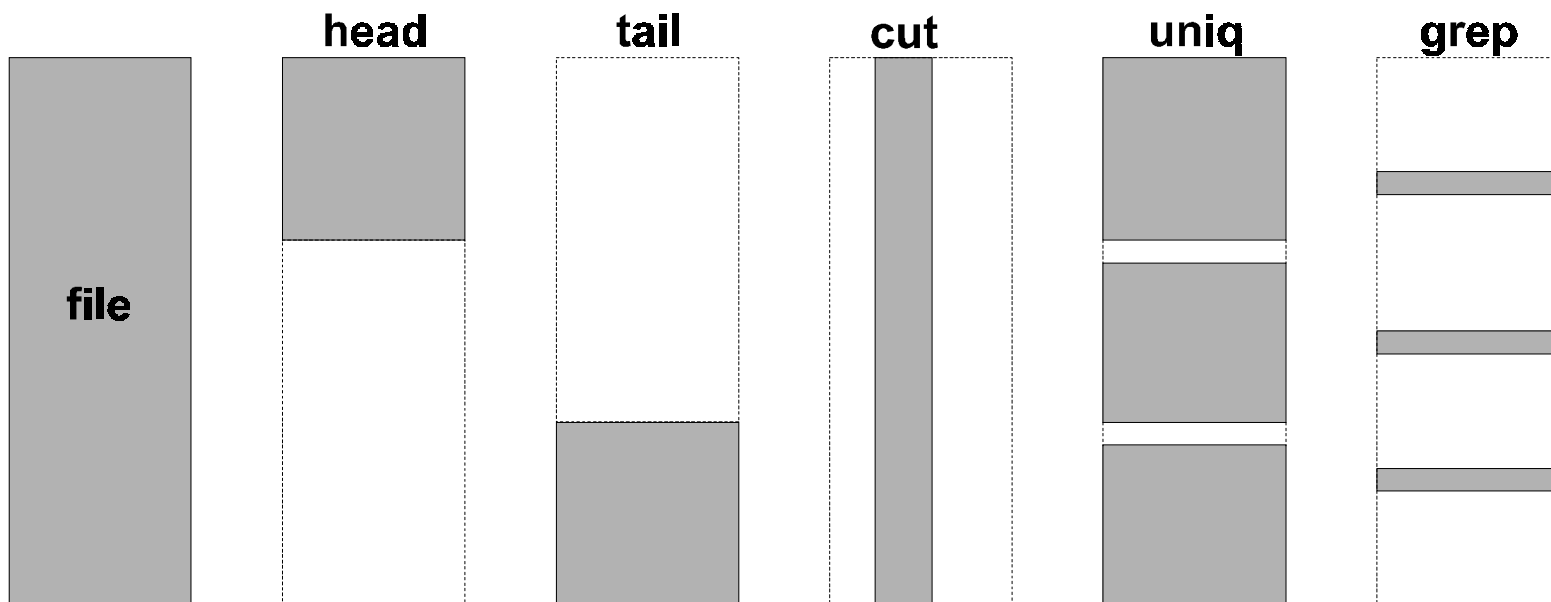
Shell Built-in Commands

:	for	test
.	getopts	times
break	hash	trap
case	if	type
continue	newgrp	ulimit
cd	pwd	unset
echo	read	umask
eval	readonly	until
exec	return	wait
exit	set	while
export	shift	

Programming Tools

- Use commands for extracting data from a file
- Use regular expression metacharacters
- Use commands for combining and sorting files
- Use **sed** and **awk** to edit the contents of a file

Extracting Data from a File



Using Metacharacters with **grep**

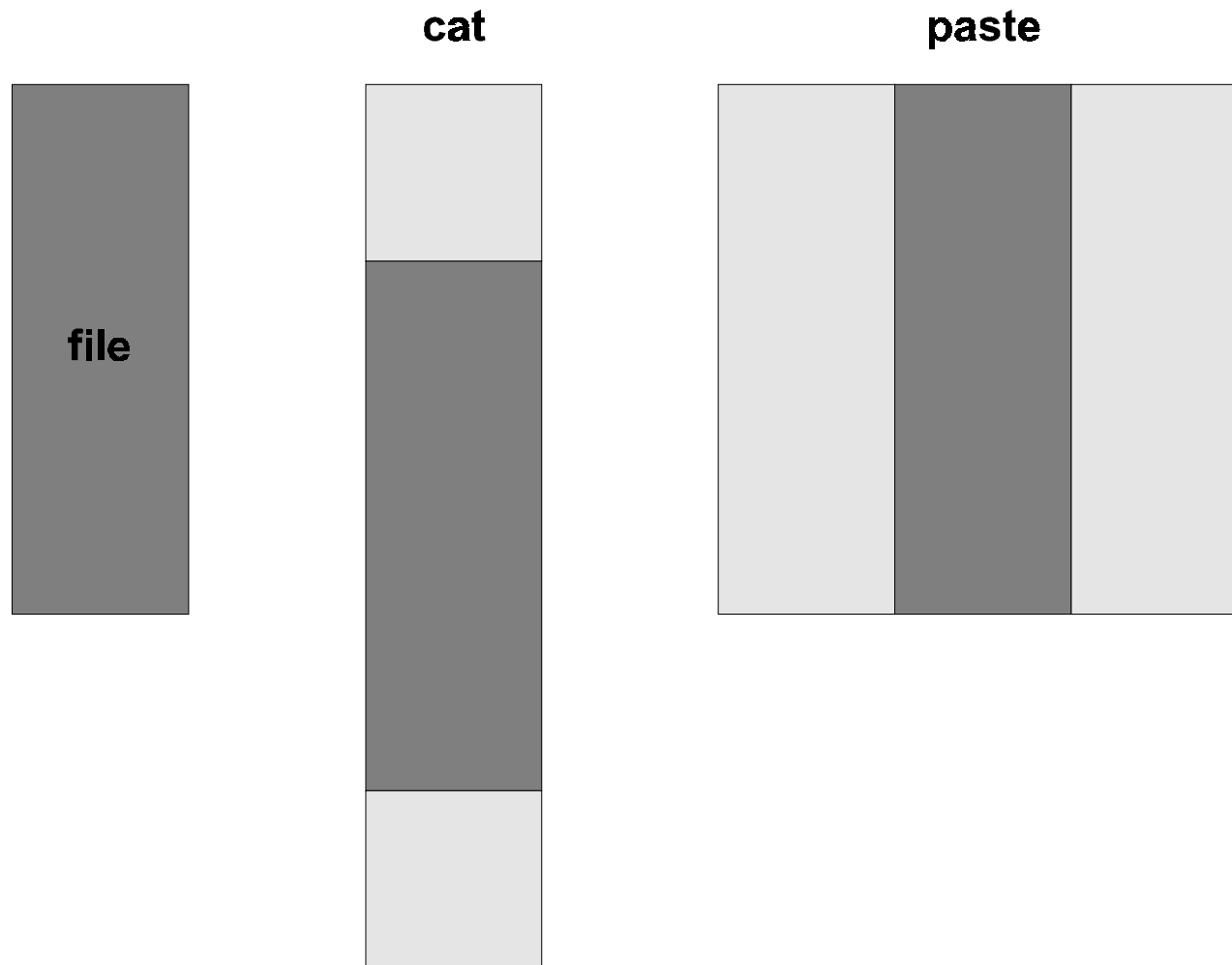
```
$ grep foo* letter  
your foot injury  
football games are today  
april fools day
```

```
$ grep foo? letter  
your foot injury
```

```
$ grep ^foo letter  
football games are today
```

```
$ grep '[Hh]enr(y|ietta)' letter  
Hello Henry  
and as I was saying to Henrietta
```

Combining Files



Using sed Scripts

```
$ cat greeting  
/Hello/s/Mike/John/g  
$ sed -f greeting < inputfile > outputfile
```

```
$ cat inputfile  
Hello Mike;  
I am pleased to be able to say  
a big "Hello" to you, Mike.  
Regards, CEO
```

```
$ cat outputfile  
Hello John;  
I am pleased to be able to say  
a big "Hello" to you, John.  
Regards, CEO
```

Using **sed** Functions (Part 1)

Change the first occurrence of “Unix” to “UNIX” on lines 1-15 of *myfile*:

```
$ sed '1,15s/Unix/UNIX/' myfile
```

Change all occurrences of “Unix” to “UNIX” on all lines of *myfile*:

```
$ sed 's/Unix/UNIX/g' myfile
```

Delete (substitute with null) the last two characters on a line, but only on lines that begin with a digit character:

```
$ sed '/^[0-9]/s/..$//' myfile
```

Delete line 5 of *myfile*, saving the results:

```
$ sed 5d myfile > newfile
```

Delete lines 8 through the end-of-file (note, this is equivalent to the command line `head -7`; results go to the *stdout*):

```
$ sed '8,$d' myfile
```

Delete lines that do not begin with a capital letter:

```
$ sed '/^[^A-Z]/d' myfile
```

Print line 5 of *myfile*, saving the results:

```
$ sed -n 5p myfile > newfile
```

Print lines that do not end with a period:

```
$ sed -n '/\.$/p' myfile
```

Using **sed** Functions (Part 2)

Append a border line (a string of asterisks) and blank line after any line that has the string "Plan" or "plan". Also delete lines containing the word "mail".

```
$ cat sedfile
/[Pp]lan/a\
*****\

/mail/d
$ finger -l | sed -f sedfile
```

Insert two lines of text at the beginning of *myfile* :

```
sed '1i\
These two lines are inserted \
at the top of the file' myfile
```

Replace lines 10-20 of *myfile* with new lines:

```
sed '10,20c\
These two lines replace\
lines 10-20 of the file' myfile
```

Replace blank lines with a border line (a string of asterisks). This example shows output being sent to the printer:

```
finger -l | sed '/^$/c\
*****' | lp
```