INFO230

INFORMATION TECHNOLOGY WORKSHOP

UNIX/C SHELL PROGRAMMING

LECTURE NOTES

Department of Electronics
Macquarie University

Recommended Readings

* Paul W. Abrahams and Bruce A. Laason, Unix for the Impatient, Addison Wesley Pubs., 1992, ISBN 0-201-55703-7. (C shell)

* William A. Parrette, Unix for Application Developers, McGraw-Hill, Inc., 1991, ISBN 0-07-031700-3. (C shell)

* M. R. Arick, Unix for DOS users, J. Wiley & Sons, 1995, ISBN 0-471-04988-3. (C shell)

Tim Hill, Windows NT Shell Scripting, Macmillan Technical Publishing, 1998, ISBN 1-578-70047-7.

P. C. Poole and N. POOLE, Using UNIX by Examples, Addison-Wesley Pubs., 1986, ISBN 0-201-18535-0. (C shell)

CONTENTS

UNIX OPERATING SYSTEM

An operating system is a collection of computer programs written to manage the resources (eg. disk drive, printer, communication ports, VDU, anything connected to the computer) available within a computer system.

Components of the UNIX operating system

The four main software components of the UNIX operating system are the kernel, the file system, the shell, and the utilities.

The kernel - That part of the software that can actually be called the operating system. The kernel is the resource manager that fulfills the requests that your application programs make.

The file system - That part of the software that stores and retrieves files for you on the peripheral devices (eg. to and from a disk drive).

The shell - That part of the software that reads, interprets, and executes the commands that you type on a terminal.

The utilities - Little applications that are delivered with UNIX to make using UNIX easier. This software allows you to list filenames, display the contents of a file, sort a file, create archives, and convert a file format to another ...

Difference between a Command and a Utility in UNIX

Other operating systems have commands that allow you to direct the operating system to do something for you, like displaying filenames. These commands are actually interpreted and executed as part of the operating system itself. Under UNIX, there are no operating system commands - only utility programs. Commands are just utility programs.

In UNIX, instead of executing the operating system command to display the names of your files, you type the name of the utility program used to display filenames, and the shell tells the kernel to exectute that utility for you. The utility then executes as “a little application” and display the filenames.

Features of the UNIX Operating System

The four software components of the UNIX system combine to offer an impressive set of features.

Multiuser operation - There are many users connected to one computer and UNIX makes it appears as if they each had their own system.

Interactive operation - UNIX prompts you for a command at a terminal, executes, displays, and prompts you for another.

Multitasking operation - Each user has the ability to run several utilities/programs at the same time.

Security and privacy - Each file has a set of permission and each user has a separate area of disk to work with.

Device-independent I/O - Every I/O device connected to the computer is just another file to the UNIX file system.

Interprocess communication - One process may have to start up another and wait for it to finish before continuing itself.

Networking - Communicate with other users on the other computer systems that are connected to the same network.

Command/utility - Commands are just utility programs. If there isn't one, you can write one.

The shell - The shell provides a fully programmable batch user interface to UNIX. If you don't like your current user interface, you can use a different shell. User environment can therefore be tailored to the user.

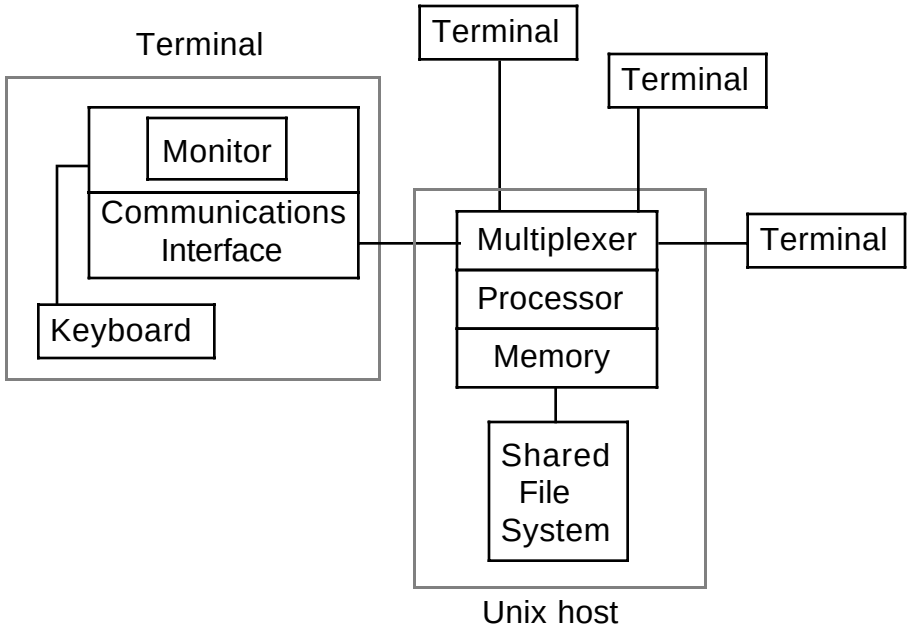


Figure 1.3 Architectural view of UNIX system.

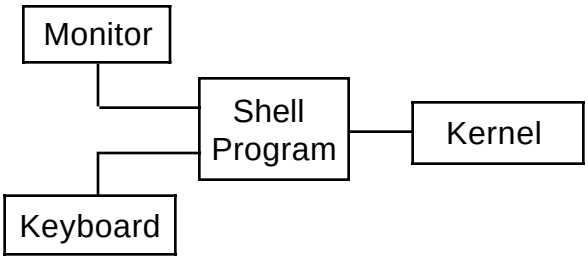


Figure 1.4 Unix system from user's point of view.

M. R. Arick, Unix for DOS users, J. Wiley & Sons., 1995, ISBN 0-471-04988-3.

Unix Directory Structure

/root directory - This master directory contains mostly other subdirectories, and only a few files, mainly the executable file called unix, which is run to start the system.

/bin directory - The most frequently used UNIX system programs are found in the binary directory.

/dev directory - Special files representing peripheral devices are found in this directory. Device handlers intercept read or write requests to these files and process them accordingly.

/etc directory - This directory contains administrative and configuration programs and files. Many basic system commands that start up the system and various subsystems that manage printing, networking, running jobs are found here.

/lib directory - The system libraries used by various programs are contained here.

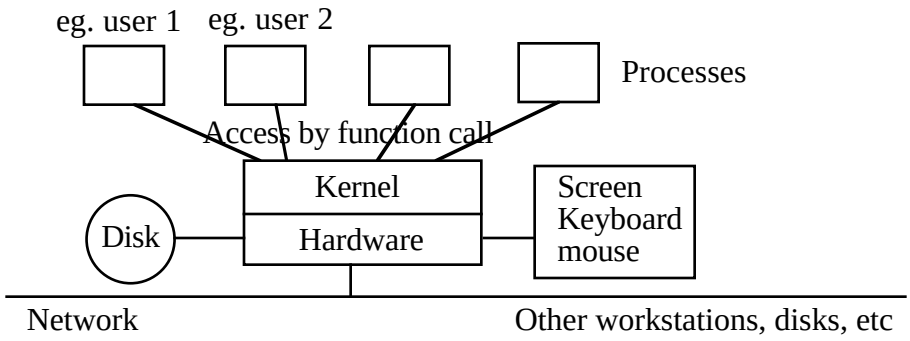
/tmp directory - Temporary files created by various system utility and users are stored here.

/u directory - Home directory for users.

/usr directory - Files used by many of the system applications, such as the mail and printer spoolers are found in this directory.Usually there is a directory /usr/local that contains commands that are installed on the system by the local system administrator. In addition, this same directory is often used applications to store various files or subdirectories that an application might need.

INTRODUCTION TO UNIX

- HP-UX is based on Unix System V and Berkeley 4.3BSD
  - Multi-user system
  - Multi-tasking system

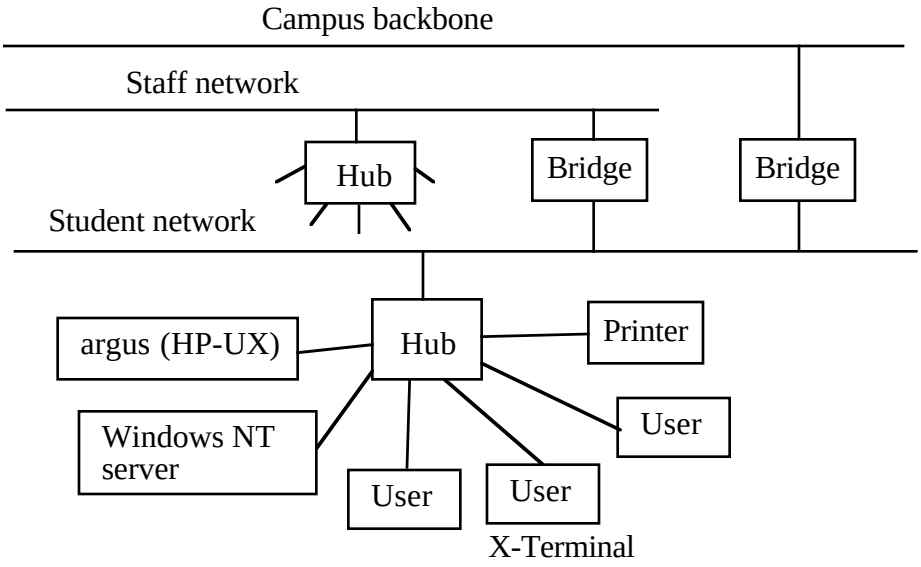


Process - may be a C shell script from a user

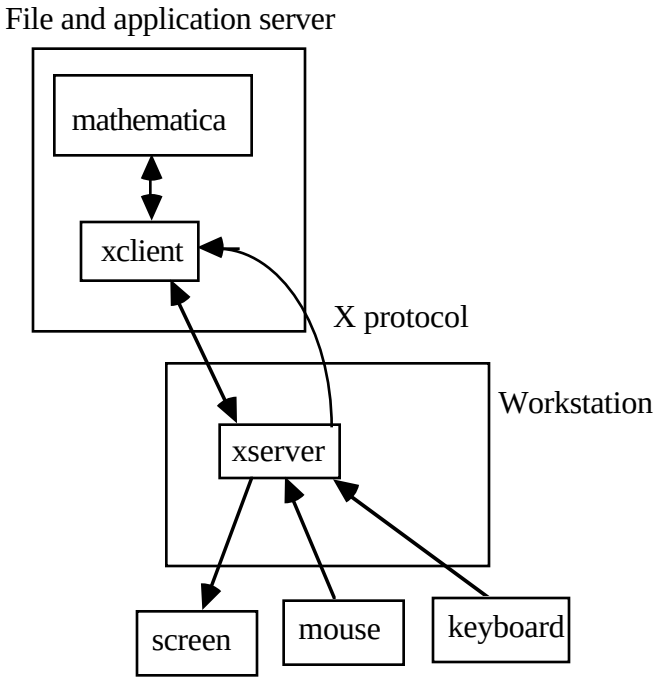
C shell variables and predefined variables are maintained in the processing side.

Environment variables are maintained in Kernel.

UNIX NETWORK ENVIRONMENT



REMOTE X SESSIONS



UNIX COMMAND LINE INTERFACE GENERAL INFORMATION

o Command line interface. % sign is prompt (can vary and be user defined)

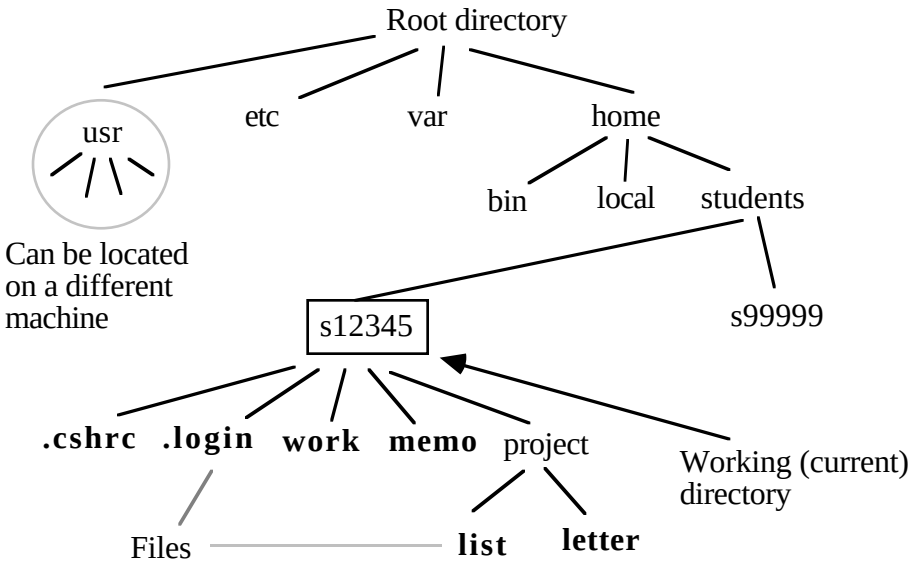
% command arguments	General format
% command -als	Typical option format
% command -a -l	Typical option format (alt)
% command arguments [arguments]	Optional arguments

Manual pages (help facility)

% man command	Look up command description and usage
% man -k keyword	Key word look up
% man 3 command	Look up man 3 pages for command (man pages are divided into sections)

UNIX FILE SYSTEM

- Strict hierarchical structure



- Working directory - default location of files
- Home directory - login directory and default location for changing directories
- Pathname identifies names of directories and files

  - General format of path names is

    [directory path]/filename

  - Directory path is

    /dir1/dir2/.../dirn

Naming Directories

/	Root directory
/var	Subdirectory of root
/home/students/s12345/project	Full path name
project	Name relative to /home/students/s12345 (shell completes pathname)
s12345/project	Name relative to /home/students
.	Current directory
..	Parent directory
~	Home directory

Naming Files

/home/students/s12345/memo	Full pathname
memo	Name relative to /home/students/s12345
./memo	Name relative to /home/students/s12345
s12345/memo	Name relative to /home/students
~/memo	Name relative to home directory

WILDCARD NAMES AND PATTERN MATCHING

- * matches any pattern

  ```
  % ls *          List all files in current directory

  % ls ad*        List all files in current directory
                  starting with ”ad”

  % ls *ing       List all files in current directory
                  ending in ”ing”
  ```

- ? matches a single character

  ```
  % ls ?ead       List all file names four letters long
                  ending in ”ead”
  ```

- [list] matches any character in list

  ```
  % ls [blm]ead   bead, lead, mead
  ```

- [lower-upper] matches any character in range lower to upper

  ```
  % ls [A-Z]ead   List all file names four letters long
                  ending in ”ead”, first letter is capital
  ```

- A{pat1,pat2,pat3}B matches patterns

  ```
  % ls H{ead,elm,eart}    Matches Head, Helm and Heart
  ```

- Spaces in file names

  ```
  % ls ”file name”
  ```

MANAGING FILES AND DIRECTORIES SUMMARY

Operation	Function	Examples
pwd	print working directory	% pwd
ls	list files in directory	% ls file1 % ls -l *.c % ls -a
mkdir	make directory	% mkdir projects
rmdir	remove (delete) directory	% rmdir projects
cd	change working directory	% cd project % cd .. % cd ~ % cd
cp	copy file	% cp assign2 assign3 % cp *.c projects
mv	move (rename) file	% mv assign1 assign2
rm	remove (delete) file	% rm assign1 % rm -i assign1 % rm -r projects
more	paginate file	% more assign1
cat	concatenate (join) file	% cat assign1 assign2
lp	print file	% lp assign1 % lp -datlantis assign1
lpstat	examine printer status	% lpstat
cancel	remove print job	% cancel datlantis-71
chown	change owner of file	% chown s12345 assign1
chmod	change file access rights	% chmod u+x assigna

CHANGING CURRENT DIRECTORY

o Display pathname of current directory

```
% pwd
```

o Methods of changing current directory

Change of directory required	Full name method	Relative name method	Other method
/home/student/s12345 ->/home/student/s12345/project	% cd /home/ student/s12345 /project	% cd project	
/home -> /home/student/s12345	% cd /home/ student/s12345	% cd student/ s12345	
/home/student/s12345/project ->/home/student/s12345	% cd /home/ student/s12345	% cd ..	
->/home/student/s12345	% cd /home/ student/s12345		% cd ~ or % cd
/home/student/s12345/project ->/home	% cd /home	% cd ../../..	% cd/ % cd home
/home/student/s12345/project ->/home		% cd ../../..	% cd/ % cd home

LIST DIRECTORIES

o Listing files in directories (% used here to indicate command line prompt)

```
% ls           List files in directory
% ls -l        Long listing (list file permissions,
                owner, size, etc)
% ls -a        List all files, including hidden files
% ls -la       List all files and long listing
% ls -R        List subdirectories as well
% ls -r        List in reverse order
% ls -F        Append: / (directory), * (executable),
                @ (symbolic link)
```

CREATING DIRECTORIES

- General command

  ```
  % mkdir pathname
  ```

- Create a new directory called (project2 under /home/student/s12345 and /home/student/s12345 is current directory)

  ```
  % mkdir project2
  ```

- Creating a new directory (called assignment under /home/student/s12345/project where/home/student/s12345 is current directory)

  ```
  % mkdir project/assignment
  ```

DELETING DIRECTORIES

- General command

  ```
  % rmdir pathname
  ```

- Delete directory a directory (project2 under /home/student/s12345 and /home/student/s12345 is current directory)

  ```
  % rmdir project2
  ```

COPYING FILES

- Three general methods

```
% cp filename newfilename    # makes copy of filename
                              called newfilename(filename
                              may also be a directory)

% cp filename directoryname  # creates copy of filename
                              with same name but in
                              directoryname

% cp filenames directoryname # copy multiple files
```

- Examples

```
% cp file1 file2   # make copy of file1 called file2

% cp file1 dir2    # make copy of file1 in dir2 with same
                    name

% cp file.* dir2   # copy all files matching file.* into
                    dir2
```

- Options

```
-f                  forced overwrite without confirmation

-i                  prompt for confirmation before
                    overwriting existing file

-r                  recursively copy subdirectories and
                    files
```

RENAMING FILES

- General command (similar format to cp)

```
% mv oldfilename newfilename   # filename may also be a directory

% mv filename(s) directoryname
```

- Examples

```
% mv file1 file2              # renames file1 as file2

% mv file.* dir2              # moves files into directory dir2
```

- Options

```
-i      prompt for confirmation before overwriting existing file

-r      recursively copy subdirectories and files
```

DELETING FILES

- General command

```
% rm pathname

% rm -i pathname               prompt before deleting
```

- Options

```
-f      forced removal of files without confirmation

-i      prompt for confirmation before removing file

-r      recursively delete subdirectories and files and then remove directory
```

WARNING

The most dangerous command in unix is rm -fr *
It will remove all your files, subdirectories and files in subdirectories without prompting for confirmation. Use it VERY CAREFULLY.

PRINTING FILES

- lp -print file or files
  - To print postscript file on default printer

    ```
    % lp file3.txt
    request id is atlantis-73 (1 file)
    ```

  - To print postscript file on printer “pandora”

    ```
    % lp -dpandroa file3.txt
    ```

- lpstat -examine print queue
  - To examine queue on default printer

    ```
    % lpstat
    atlantis-73 s12345 priority 0  Feb 3 13:50 on atlantis
    file3.txt   30 bytes
    ```

  - To examine queue on printer ”Pandora”

    ```
    % lpstat -dpandroa
    ```

- cancel -remove entry from printer queue
  - To remove entry atlantis-73 on default printer

    ```
    % cancel atlantis-73
    ```

VIEWING FILES

o more command - display one page at a time

```
% more file1
% more ch*      view all files beginning with ch
```

Commands available when inside more

```
[SPACE]         view next page
[RETURN]        view next line
q or Q          quit
:n              next file
/pattern        search forward for pattern
/               repeat search
?               help
```

o Example of more used with pipe

```
% ls | more file1
```

o cat file

o head file-view beginning of file

```
% head file
% head -n file          # first n lines
```

o tail file-view end of file

```
% tail file
% tail -n file          # last n lines
```

PROCESSING FILES

- wc (word count)

  o wc [options][files]

  options

-c	character count only
-l	line count only
-w	word count only

- sort file - sort in alpha/numeric order

  options

-n	numeric order
-r	reverse order
-u	identical lines only appear once
-d	dictionary order (ignore punctuation)
-f	ignore upper/lower case differences

- diff [options][diroptions] file1 file2 - show difference between two files

  options (lots more)

-i	ignore upper/lower case differences

- cmp file1 file2 - compare two files

  options

-s	work silently

  Exit codes:

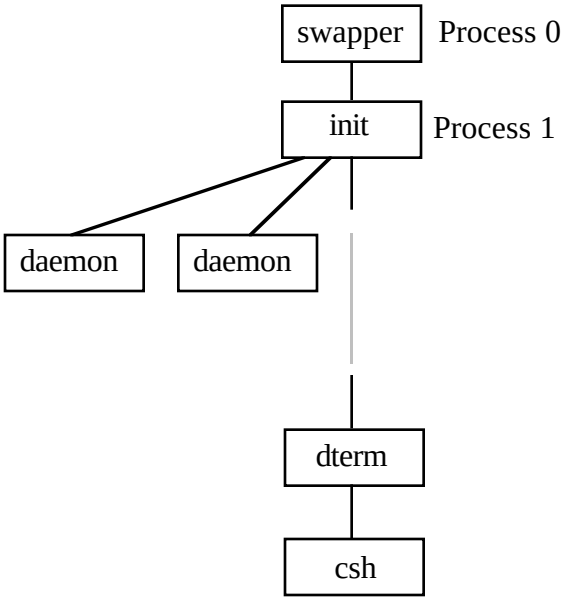
0:	same
1:	different
2:	files not accessible

CREATING FILES

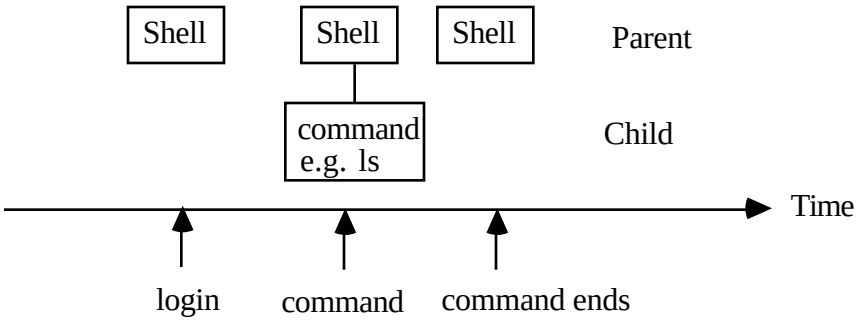
- Text files
  - Text editor (e.g. vi, emacs)
  - Notepad editor
  - cat command (Direct entry from keyboard)
  - Redirection of screen output to file
  - Applications
- Other files - variety of means
- Examples using cat

```
cat file1                  # displays file1 on screen

cat file2 file2            * displays file1 and then
                           file2 on output

cat file1 dile2 > file3    # combines file1 and file2
                           into file3

cat file4 >> file5         # appends file4 into file5

cat > file6                # reads input from keyboard
                           (terminal input with <return>
                           and CTRL-D) to create file


cat - file7 > file8        # creates file8 from stdin and
                           file7
```

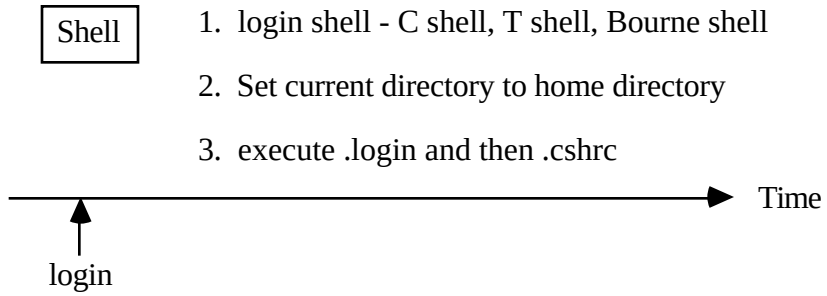
PROCESS HIERARCHY



EXECUTING A COMMAND



LOGGING IN AND OUT



- Other shells
  - /bin/sh Bourne shell
  - /bin/ksh Korn shell
  - /bin/tcsh T shell

SPECIAL FILES (235)

- .cshrc
- .login
- .logout
- .profile (used with Bourne shell)

EXITING A PROCESS

- exit exit from shell
- logout end terminal session from login shell

SHORT CUTS

- Use CTRL-C to interrupt execution of a command
- Cut and paste using mouse
  - select text by dragging with left button
  - paste by clicking with middle button
- Use ESC to extend filenames
- History commands
  - !! previous command
  - !$ past word of previous command
  - !vi repeat command beginning with vi
- Commands more than one line long (use \ at end of line and continue to next line)

```
% command -a argument1 -b argument2 -c argument3 \
-d argument4
```

FILE PERMISSIONS

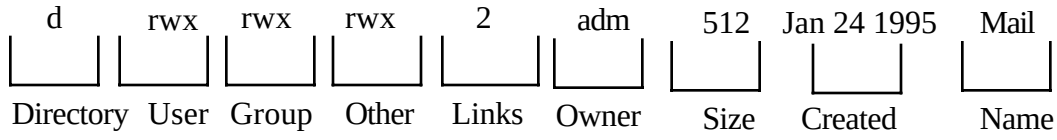
- Files have owners
  - changed by (needs superuser privilege)

```
% chown new_owner
```

- These are classified according to
  - u — user, owner of files
  - g — group, collection of users
  - o — other
  - a — all users regardless of owner or group
- Action to take:
  - \+ — add the access right
  - \- — Substract the access right
- Files and directories have access permissions
  - r — file can be read
  - w — write
  - x — execute
- Verify file permissions

```
% ls -l
```

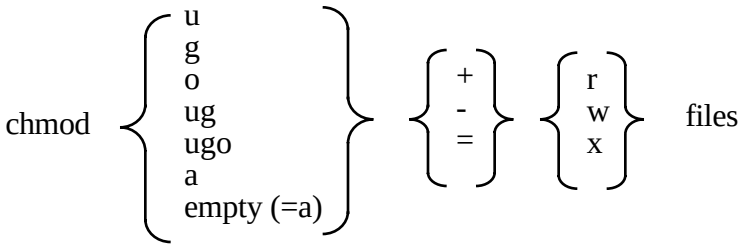
```
-rw-rw-r--  1  adm  4356 Jan 24  1995  DXclock
drw-rw-r--  2  adm   512 Jan 24  1995  Mail
-rwxr-x--x  1  adm  9876 Dec  2  1995  project.report
```



chmod

o Permissions changed using chmod command

o Method 1 (symbolic form)



o Examples

```
chmod u+w       # add user write permission
chmod u-x       # removes user execute permission
chmod u=x       # only allow user execute permission
chmod ug+rwx    # add read, write, execute permission
                to user and group
chmod a-w       # remove write permission from user,
                group, others
chmod -w        # same as chmod a-w
chmod =         # remove all permissions
```

chmod-Method 2 (absolute mode)

		user (owner)	group	other
read	4000 (set user ID on execution)	0400	0040	0004
write	2000 (set user ID on execution)	0200	0020	0002
execute	1000 (sticky)	0100	0010	0001

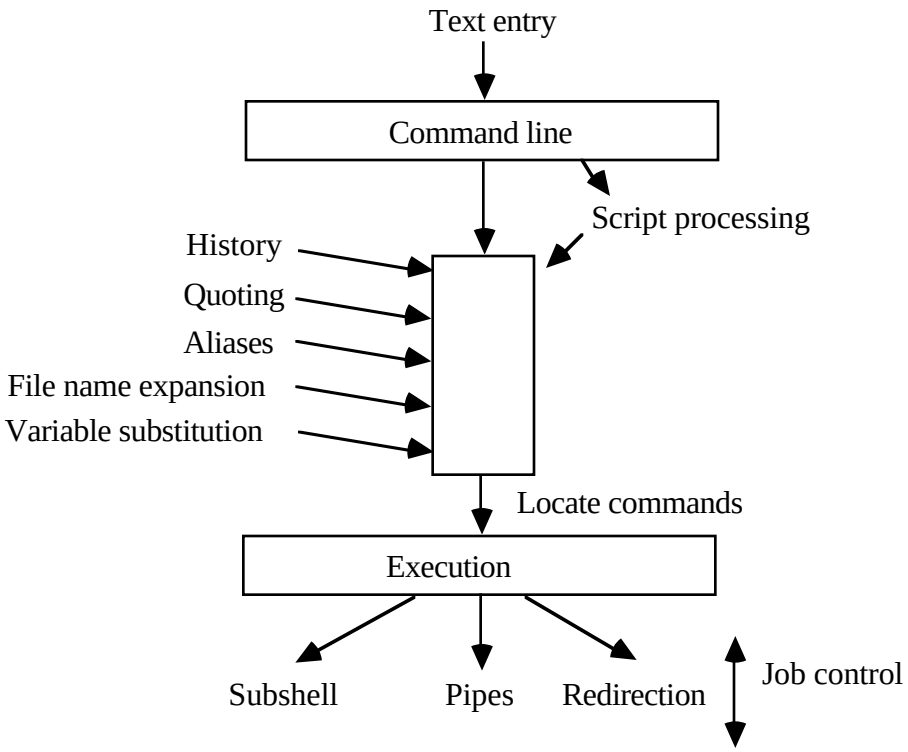
o Examples

```
chmod u=rwx     chmod 0700  (0400 + 0200 + 0100)

chmod go+rx     chmod 755  (400+200+100+40+10+4+1)
```

o set umask=002 Indicates which permissions to **disable** on creating a newfile/directory. Therefore set unmask=002 turns off other write permission.

o set noclobber turns on protection accidental overwriting of files

STEPS IN PROCESSING COMMANDS



QUOTING

- (Characters that need escaping)
  - \c where c is a single character
  - characters that must be escaped

SPACE, TAB	command argument separator
RETURN	command line terminator
$	variable identifier
+ []?{} ~ -	file expansion
> < & !	redirection
! ^	history
|	pipe
;	command delimiter
( )	command group
\ ’ ”	quoting
`	command substituting
&	background tasks

- ” ” allow variable expansion and command substitution ($, `, \`, ! not escaped) (N.B. ‘ is escaped)

```
% set abc=123
% echo ”$abc”
% 123
```

- ’ ’ no variable expansion and command substitution (\`, ! not escaped)

```
% set abc=123
% echo ’$abc’
% $abc

% echo ’Single quotes ”protect” double quotes’
Single quotes ”protect” double quotes

% echo ”Well, isn’t that \”special\”?”
Well, isn’t that ”special”?
```

- ` ` command substitution output of command is string (37)

```
% echo `date`
% Tue Jan 30 17:02:47 EST 1996
```

- :q quote a wordlist - same as double quote
- :x quote separate words in a word list - prevents filename expansion

COMMAND EXECUTION

- command & execute command as background process

  % netscape &

- ; sequential commands: command1; command2;command3

  % ls ; du

- As a subshell using (commands)

  - Execute commands without changing current environment

    ```
    % pwd
    /usr/home/student/s12345

    % ls
    ass1 ass2 ass3

    % (cd /usr/home; ls)
    file1 file4 file6
    file2 file5

    % pwd
    /usr/home/student/s12345
    ```

- command1 || command2

  successful || do not execute
  unsuccessful || execute

  e.g.

  % grep name file1 || echo 'String not found'

- command1 && command2

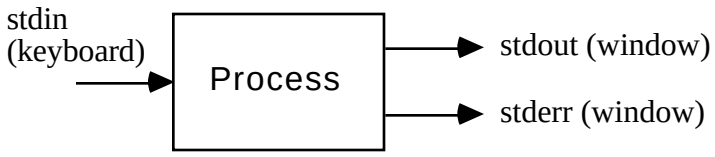
  successful && execute
  unsuccessful && do not execute

  e.g.

  % lpr file1 && echo 'File has been sent to printer'

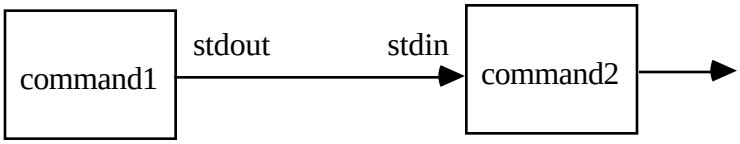
COMMAND INPUT/OUTPUT

STANDARD INPUT and OUTPUT STREAMS AVAILABLE



- Unix provides flexibility using plug and play commands
  - Many commands assume input from stdin if no file argument specified
  - This enables commands to be chained using pipes

PIPES



% command1 | command2

Examples

```
% ls | more
% ls | wc -w
```

OUTPUT REDIRECTION

o Direct output of command of pipe to file instead of screen

New/append	stderr and stdout	Override noclobber if set
>	&	!
>>		

Examples

```
% ls -l > file          # stdout to file
% ls -l >> file         # append to file
% ls -l >! file         # stdout with override
% ls -l >&! file        # stdout and stderr to file
                         and override noclobber
```

o To discard output redirect to /dev/null

```
% ls > /dev/null
```

o Redirection from subshell (output from both commands redirected)

```
% (date;du) > status
```

o Redirection from sequence (only final command output is redirected)

```
% date;du > status
```

INPUT REDIRECTION

<	Input read from a file
<<	Input read from the terminal until eof is signalled

o command < file

JOB CONTROL

Examining processes

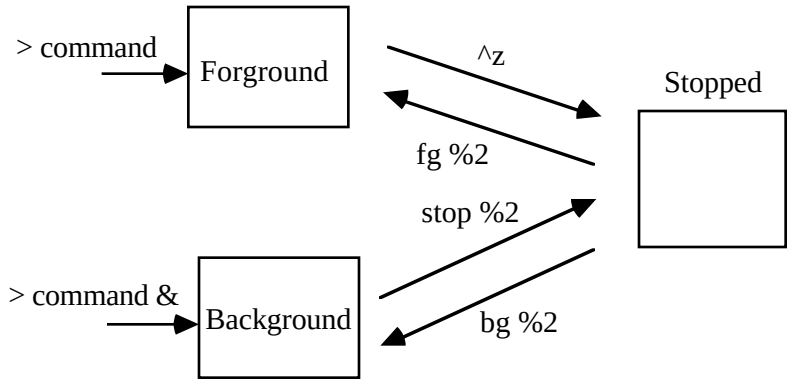
ps	list all your own processes
ps -ef	list all processes jobs list your background jobs

Terminating processes

exit	terminate shell from command line/script
kill %	kill current job
kill 1234	kill process ID
kill %2	kill job 2
kill %du	kill job starting with du
kill %?storage	kill job with string matching storage
kill -9 1234	Unstoppable kill

Foreground/background process control

command &	starting a background process
nohup command &	do not terminate job when you log out
^z	suspend foreground job
fg %2	resume suspend job %2 in foreground
stop %2	suspend background job 2
bg %2	resume job %2 in background



HISTORY COMMANDS

!!	repeat previous command
!7	repeat command 7
!v	repeat last command beginning with v
!vi	repeat last command beginning with vi
ls !$	use last word of previous command
!?letter?	use last command containing string letter
ls !!:3	use word 3 of previous command
ls !ls:2-$	use word 2 to end of last command beginning with ls
^too^to^	substitute too with to in previous command
!4:s/ch2.a/ch2/	substitute ch2.a with ch2 in command 4
pr !114:1-4:p	print command

Examples

% history	show history
% history n	last n commands
% history -r	reverse order
% history -h	no history/event numbers
% set history = n	save n history commands

o History saves in ~/.history

ALIASING

- Create new names for existing commands and command combinations

```
% alias      new_name              definition

% alias h    history

% alias ll   'ls -l'

% alias cx   'chmod +x'

% alias ah   'history | head -15'

% alias rm   'rm -i'

% alias who  'who; du'             # alias of who must
                                    appear first to define

% alias llm  'ls -l \!*|more'      # \!* means commands
                                    arguments (! escaped)
                                    command arguments
                                    usually passed to last
                                    command mentioned
```

- Listing aliases

```
% alias
```

- Removing aliases

```
% unalias cx
```

grep

- Used to search for patterns in files
  - Useful in pipes
- grep [options] pattern file
- can use regular expressions
  - Must be careful with quotes

```
% grep Wednesday file
# search file for lines containing Wednesday

% grep ”^Wednesday” file
# search file for lines starting with Wednesday

% grep -c ”^Wednesday” file
# print count of lines starting with Wednesday

% grep -v Wednesday file
# search file for lines not containing Wednesday

% ls -l | grep ’\.c’
# list files in directory ending with .c
# (note . needs escaping) and single quotes
```

REGULAR EXPRESSIONS

Used by vi, ex, grep, awk, sed, egrep, ed

.	Any character
*	Zero or more preceeding
^q	Beginning of line
$	End of line
\	Escape character
[]	From set
+	One or more preceeding (awk egrep only)

Example

bag	”bag”
^bag	”bag” at the beginning of a line
bag$	”bag” at the end of a line
[Bb]ag	Bag or bag
b[aeiou]	b followed by a vowel
b[^aeiou]	b followed by a consonant (not a vowel)
^...$	A line with three characters
[A-A]and	A word beginning with a capital and ending with ”and”
[A-Z]{a-Z]*	One or more uppercase letters
[a-zA-Z]	Any letter
[0-9]	A single numeric digit

FINDING INFORMATION

- man

  ```
  % man nice
  % man 2 plock     # look in section 2 for plock
  % man -f ls       # display one line summary
  % man -k keyboard # look for keyboard
  ```

- Adding new man page directories

  ```
  setenv MANPATH path:path:path
  ```

- apropos pattern — similar to man -k keyboard
- whatis command — one line summary of command
- who — show who is logged onto system
- whoami — display my user name
- users — list users logged onto system
- id — list user id and group id
- which — examine $path to determine binary command corresponds to
- find - used to locate a file

  ```
  % find /usr -name file -print
  ```

TEXT PROCESSING

- ex — non-screen based interactive line editor
- vi — screen based text editor
- emacs — screen based text editor
- ed — text editor (superceded by ex and vi)
- awk — pattern matching program similar to spreadsheet
- sed — non-interactive stream editor which applies edit operation to each line in file
- grep — string searching
- egrep — string searching
- cut — cut fields from columns in text file
- paste — merge files like columns
- spell — spell checker
  - /usr/lib/spell database
- troff — document formatting program
- nroff — document formatting program
- pr — pretty files up with multiple columns and formats

OTHER UNIX COMMANDS

```
% clear              # clear terminal screen

% ln                 # creates symbolic links
```

STATUS COMMANDS

```
% uname              # gives current UNIX system name

% time command       # determines how long command takes to
                      execute

% date               # current date and time

% bdf                # amount of disk space used on file
                      system

% du                 # show directory storage used

% quota -v           # shows disk quotas

% file path          # show type of files (e.g. ascii,
                      executable etc)
```

FILE COMPRESSION AND EXPANSION

o create tar archive

```
% tar cf archive files-to-compress
% tar cf project.tar project
```

o expand tar archive

```
% tar xvf archive
% tar xvf project.tar
```

o compress file

```
% compress project.tar
```

o uncompress file.Z

```
% uncompress project.tar.Z
```

o unencode

o undecode

o gzip

o gunzip

NETWORK APPLICATIONS

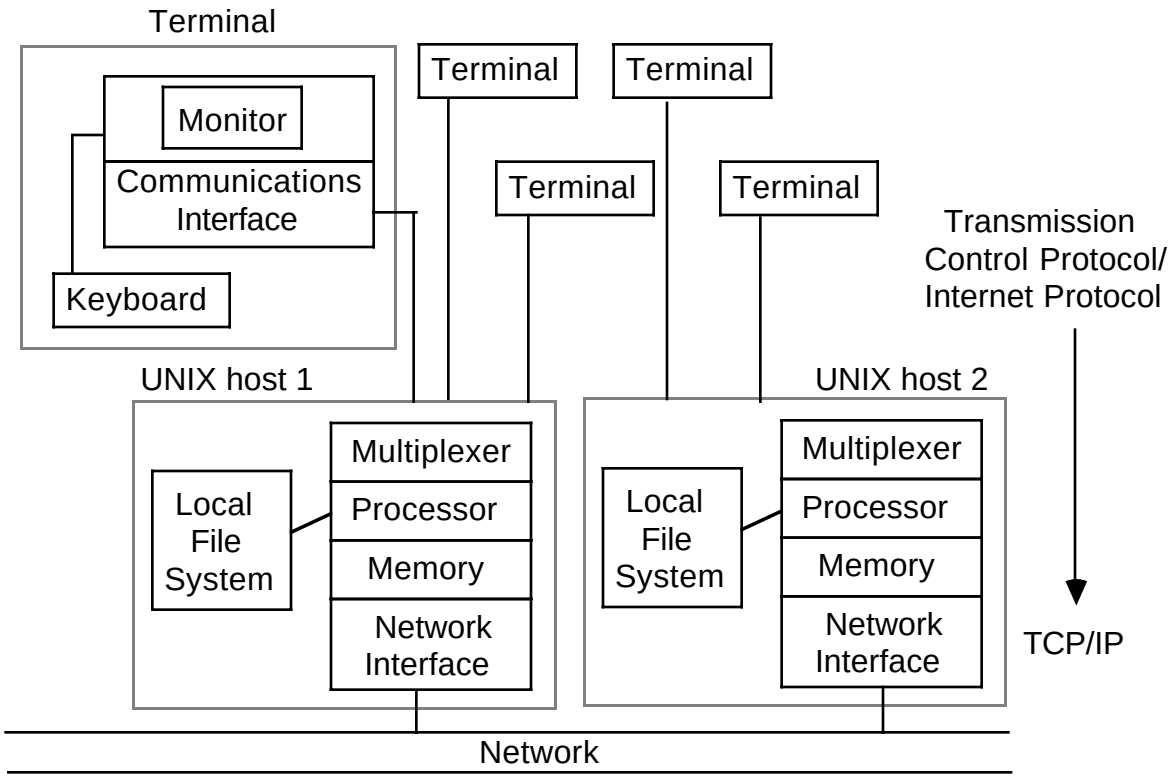


Figure 13.1 Networked UNIX systems in a local area network (LAN).

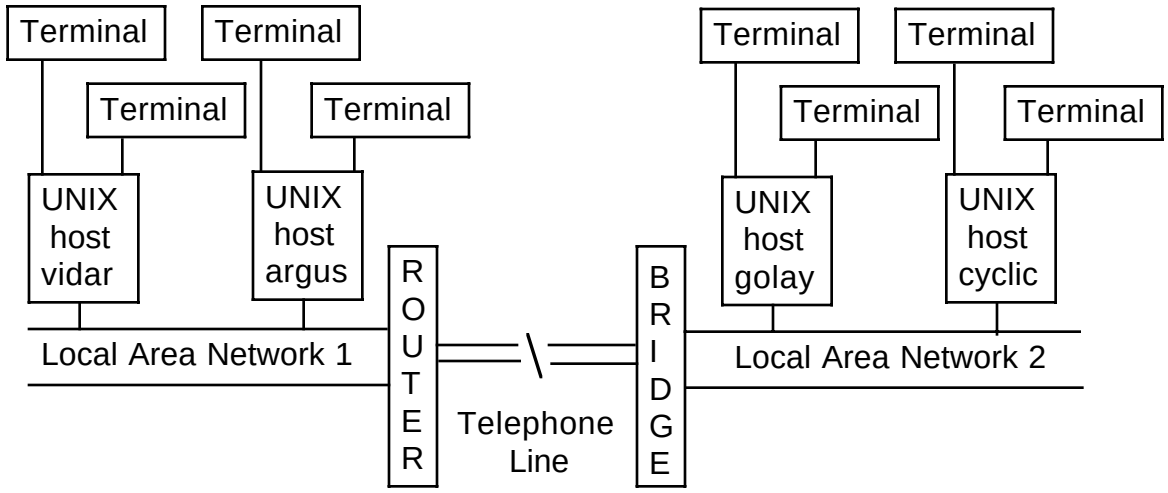


Figure 13.2 Networked UNIX systems in a wide area network (WAN).

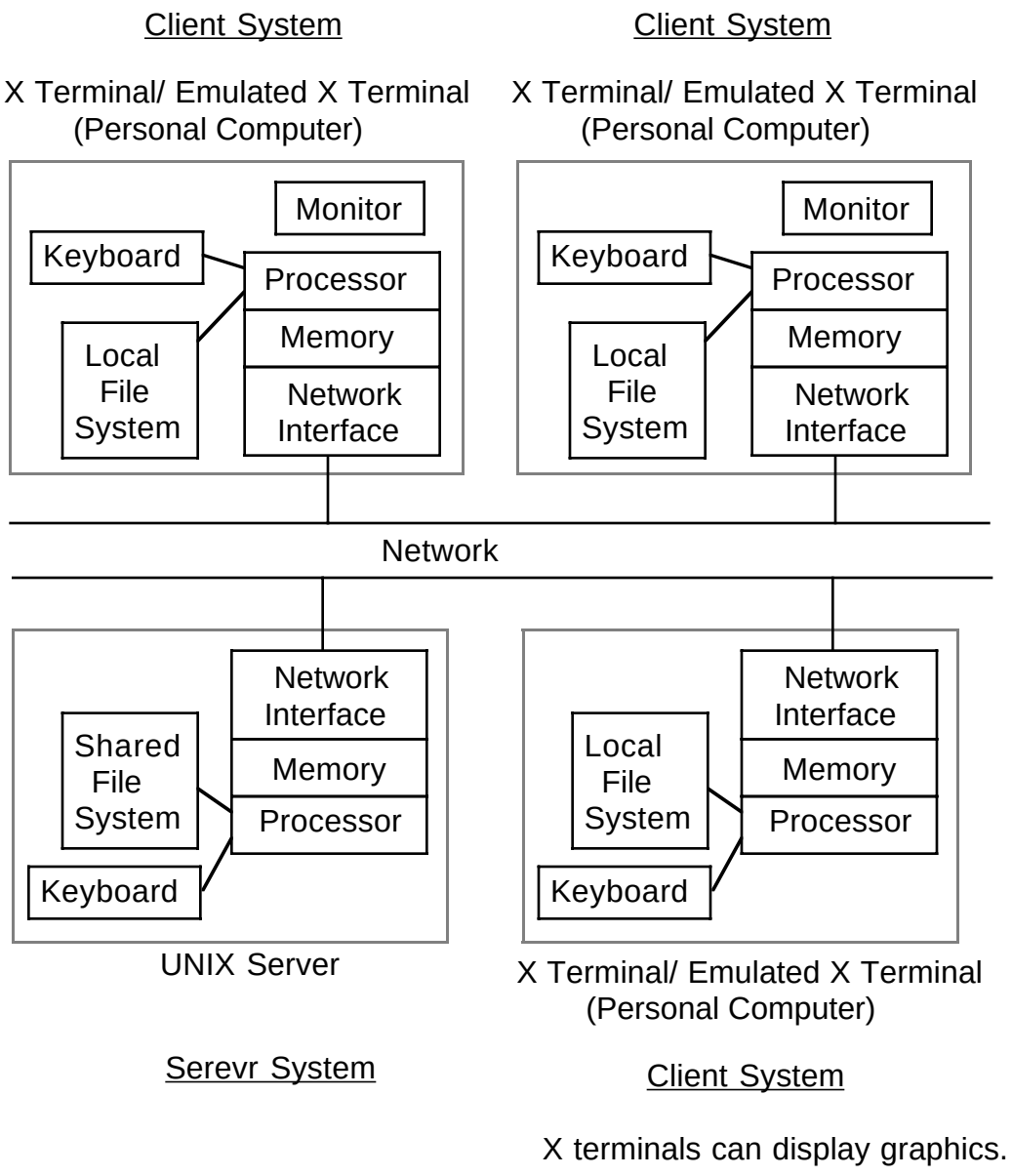


Figure 13.6 Using X or Emulated X Terminals in a UNIX network.

M. R. Arick, Unix for DOS users, J. Wiley & Sons., 1995, ISBN 0-471-04988-3.

NETWORK COMMANDS

- email — electronic mail
- mail — mail program
- ftp — file transfer
- telnet — remote terminal

```
% telnet macadam
Login: s232323
Password: ********
CTRL-]          # abort
telnet > quit
```

- rlogin — remote login to UNIX machines

```
% rlogin macmath
Password: ********
CTRL-D          # abort using EOF
```

- rsh — remote shell execution on UNIX machines (local and remote host need to be in ~/.rhosts file)

```
% rsh macmath   command
```

- rcp — remote copy
- netscape — world wide web client
- gopher — text-based network information system
- de (X.500) — distributed database client
- ping — check if machine is alive
- talk — two-way conversation text communication
- finger — identify user on remote machine

```
% finger mark

% finger mark@macmath
```

USES OF SHELL SCRIPT

A shell or command interpreter is a program that interprets and executes comands as you issue them from your terminal.

- Batch execution of commands
  - Frequently used sequences of commands
  - Save time
  - Can be parameterised
- Used for
  - saving time
  - file housekeeping
  - system administration (start-up, shut down)
  - tailoring your environment (.cshrc, .login, .logout)

In summary, shell scripts canhelp cut down on the time spent on repetitive manual tasks, and can be used to replace UNIX commands that you don’t like or to create new UNIX commands.

BASICS OF SHELL SCRIPTS

- Create and modify using text editor (Notepad, vi, emacs etc)
- Begin script file with #!/bin/csh on first line (beginning with # assumes Bourne shell /bin/sh) (can also use #!/bin/csh -f to speed up execution)
- To run script file

  (a) use source command and run within current shell

  ```
  % source script
  ```

  or

  (b) change mode of file and run as subshell

  ```
  % chmod u+x script
  % script
  ```

FEATURES

- Many similarities to other programming languages
- Specially tailored to running UNIX programs and the UNIX environment
- Shell variables
  - Predefined
  - User defined
- Operations based on string manipulation, especially words
  - File and directory names special operations possible
  - Special way of dealing with numbers
- Grammar (some similarity to C language)
  - if, if-then, if-then-else
  - foreach
  - while
- Access to UNIX commands

UNIX COMMANDS OFTEN USED IN SHELL SCRIPT

- echo (printf statement equivalent of C language)

```
% echo ”This is a line”      new line inserted at end

% echo -n ”This is a line:” no new line at end
```

- exit statement (exit from script)

```
exit                        # successful exit
                            (code is 0)

exit 1                      # indicates error exit with
                            code 1

exit (expression)
```

SIMPLE EXAMPLES

Example numfiles1

```
#!/bin/csh
#
date
ls | wc -w
```

o Comments

- o date prints out date and time
- o ls | wc -w This is a pipe. The output of the ”ls” command is passed directly to the input of the ”wc -w” command which counts the number of words in the input.

To run:

```
% ls
assign1   script1   zmodem1
assign2   script2   zmodem2
% chmod +x numfiles1            # only do this once
% numfiles11
Mon Feb 12 17:09:16 EST 1996
6
```

Example jackandjill1

```
#!/bin/csh
# This is a comment line
# anything after the hash is ignored
#
set char1 = Jack
set char2 = Jill
echo $char1 and $char2 went up the hill
```

o Comments

- o set char1 = Jack — Defines a shell variable called ”char1” and it is assigned the string ”Jack”
- o echo — Is used to output what follows to the terminal
- o $char1 — This refers to the value of the shell variable ”char1”

To run:

```
% chmod +x jackandjill1         # only do this once
% jackandjill1
Jack and Jill went up the hill
```

Example jackandjill2

```
#!/bin/csh
#
set char1 = Jack
```

```
set char2 = Jill
echo $char1 and $char2 went up the hill
echo ${char1}and$char2 productions
```

o Comments

  o ${char1} used quotes to delimit shell variable from surrounding letters.

To run:
```
% chmod +x jackandjill2          # only do this once
% jackandjill2
Jack and Jill went up the hill
JackandJill productions
```

Example jackandjill3

```
#!/bin/csh
#
echo $argv[1] and $argv[2] went up the hill
echo $argv[1]and$argv[2] productions
```

o Comments

  o Use command line to pass arguments

    - $argv[1] is first command line argument
    - $argv[2] is second command line argument

To run:
```
% chmod +x jackandjill3          # only do this once
% jackandjill3 Eric Mavis
Eric and Mavis went up the hill
EricandMavis productions
```

Example jackandjill4

```
#!/bin/csh
#
echo Jack and Jill went up the hill on `date`
```

o Comments

  o `date` This executes the UNIX command data and uses that in the script (command substitution, using back quotes)

To run:
```
% chmod +x jackandjill4          # only do this once
% source jackandjill4
Jack and Jill went up the hill on Mon Feb 12
17:09:16 EST 1996
```

SHELL VARIABLES

- Used by shell only
  - pre-defined
  - user defined
- Basic format and usage

set abc=/usr	setting
echo $abc	access using $ sign
unset abc	deleting
set	examine all shell variables

- Examples

ls $abc

set abc = (bob joe bill)	list of words
set abc = ”bob joe bill”	several words treated as one word

- Special constructions

$#abc	number of words in variable abc
${abc}2	protection from neighbouring letters
$$	process ID
$0	name of shell script
$<	terminal input via keyboard e.g. % set a = $< (terminate with CR)
$?var	Test if variable set
${?var}	Test if variable set (with protection)

PREDEFINED C SHELL VARIABLES

- Used to customise shell

  set name = val

  unset name

- path - where to look for files

  set path = (. ~ /bin /bin /usr/bin)

- cdpath - use with cd command to find directories

  set cdpath (~/work)

- Other predefined variables

home	use with cd command when no variable
shell	which shell is in use
status	exit status of last command
verbose	show command after history substitution
mail (/usr/spool/mail/bill)	look here regularly to check for new mail
history	maximum levels of history remembered, e.g. history=10
set savehist=5	save history commands between login sessions
set prompt = '\!%'	defines prompt
set ignoreeof	do not use CTRL-D to log out
set time = 3	report any command taking more than 3 seconds of CPU time

ENVIRONMENT VARIABLES

Each process has a collection of environment variables associated with it. These variables can be queried or set by the process and are inherited by its subprocesses. Whenever you start a new process as a child of another, UNIX sets the environment variables of the child process to a copy of those of its parent. Thereafter the environment variables of the child and the parent are independent, since one process cannot examine or modify the environment variables of another. When a process terminates, its environment variables disappear and any changes to these variables are lost.

- Setting environment variables

  setenv NAME string

- Examine environment variables

  env

  printenv

- Accessing environment variables

  $ABC
  ${ABC} with protection

- In uppercase (by convention)

- Predefined environment variables

TERM	terminal type
HOME	home directory
PATH	list of directories to find commands
LOGNAME	
USER	My user name
SHELL	shell being used
MAIL	
EXINIT	options used by vi and ex
TERMCAP	location of database for terminal files cursor position codes for terminal type
PWD	
DISPLAY	location of screen to be used for display

LISTS

- Refer to individual words by $var[1], $var[2], etc
- Number of words is given by $#var
- Defining lists - Quoting and grouping makes a difference

```
% set x = aaa bbb ccc
% echo $x $#x $x[1]
aaa 1 aaa

% set x = ”aaa bbb ccc”
% echo $x $#x $x[1]
aaa bbb ccc 1 aaa bbb ccc

% set x = (aaa bbb ccc)
% echo $x $#x $x[1]
aaa bbb ccc 3 aaa
```

SOME TRICKIER EXAMPLES

```
% set pat = ”ex.1 ex.2 project”
% echo $#pat
1

% set pat = ($pat)          % split one word into
                            multiple words
% echo $#pat $pat
3 ex.1 ex.2 project

% set pat = ”dict[1-4] *.doc”
% set pat = ($pat)          # shell does filename
                            expansion

% echo $#pat $pat
5 dict1 dict3 dict4 help.doc abc95.doc
% set pat = ”dict[1-4] *.doc”
% set pat = ($pat:x)        # use :x for no filename
                            expansion

% echo $#pat $pat
2 dict[1-4] *.doc
% set aaa = (ex.1 ex.2 project)
% set new = ($aaa $pat:q)   # use :q to preserve word
                            lists
% echo $#new $new
5 ex.1 ex.2 project dict[1-4] *.doc
```

COMMAND LINE SHELL VARIABLES

```
$argv[i]     i-th command line argument
$i           i-th command line argument (i = 1-9 only)
$argv        all of command line arguments
$argv[*]     all of command line arguments
$*           all of command line arguments
$#argv       number of command line arguments
${#argv}     number of command line arguments
```

o Example - demo_argv

```
#!/bin/csh
#
echo $argv
echo $argv[1]
echo $argv[2]
echo $3
echo $*
echo $#argv

% demo_argv aaa bbb ccc ddd
aaa bbb ccc ddd
aaa
bbb
ccc
aaa bbb ccc ddd
4
```

PARSING FILE NAMES

```
set abc =          /usr/local/gets.c

root               $abc:r /usr/local/gets

header             $abc:h /usr/local

tail               $abc:t gets.c

extension          $abc:e c
```

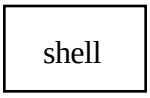
o Global parsing of file names (every word in list not just first)

```
root               $abc:gr
header             $abc:gh
tail               $abc:gt
extension          $abc:ge
```

SCRIPT EXECUTION

EXECUTE IN CURRENT SHELL

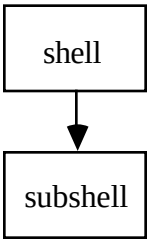
```
% source script        # run script in current shell
                        # no script arguments allowed

% source -h script     # do not update history list
```



Use aliases and variables
in current shell

- **Cannot pass arguments to script**

EXECUTE IN SUBSHELL



Shell variables, aliases and
environment variables defined
here not passed back to parent

```
% chmod u+x script
% script               # run script in subshell
```

- Adding shell scripts defined as executable files to command path

```
% chmod u+x file       # change prem. to executable
% set path = ($path /usr/local/student/s12345/bin)
                        # add location command
                        (directory) to path
%rehash                 # recompute hash tables to
                        locate commands
```

SHELL PROGRAMMING - if-then-else

o if (expression) then
   commands
endif

o Boolean values:

0: false otherwise 1: true

o Comparison operators

==	equal
!=	not equal
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
=~	string matches file name pattern
!~	string does not match file name pattern

o iftest1 arg1 arg2 ...

```
#!/bin/csh
#
if ($#argv == 0) then
   echo ”At least one argument must be present”
   exit 1
endif
```

o Boolean operations

&&	logical and
||	logical or
!	logical not
{ command }	return 1 if successful and 0 if not successful (opposite of normal return)

o Bitwise operations

&	bitwise and
|	bitwise or
^	bitwise exclusive or
~	binary complementation
<< >>	bitwise shift left and right

if (cont)

- File enquiry operators

```
-r                read access
-w                write access
-x                execute access
-e                file exists
-f                is file (not directory)
-d                is directory (not file)
-z                file is of zero length
-o                user owns file
```

- iftest2 arg1

```
#!/bin/csh
#
if (-d $argv[1]) then
   echo ”argv[1] is a directory”
endif
```

Other variants of if

-
```
if (expression) command
```

-
```
if (expression) then
   commands
else
   commands
endif
```

-
```
if (expression) then
   commands
else
   if (expression) then
      commands
endif
```

FOR LOOP

- foreach var (wordlist)
  commands
  end

- Example

```
#!/bin/csh
#
foreach name (bob bill jim)
   echo $name
end

% script
bob
bill
jim
```

- Example

```
#!/bin/csh
#
foreach name ($argv)
   echo $name
end

% script aaa bbb ccc
aaa
bbb
ccc
```

- Example - * means every file in the current directory

```
#!/bin/csh
#
foreach file (*)
   echo $file
end

% ls
Mail   project1   assign
% script
Mail
project1
assign
```

WHILE LOOP

o
```
while (expression)
   commands
end
```

o EXAMPLE

```
#!/bin/csh
#
@ n = 3
while ($n)
   @ n--
   echo $n
end

% script
2
1
0
```

o EXAMPLE

```
#!/bin/csh
#
@ n = 3
while (1)
   @ n--
   echo $n
   if ($n == 0) break
end
echo ”Loop finished”

% script
2
1
0
Loop finished
```

SHIFT STATEMENT

- Deletes first word in list and shifts remaining words one place along
- Automatically updates length of list
- shift var
- shift # operates on argv
- EXAMPLE

```
#!/bin/csh
#
while ($#argv)
   echo $#argv $#argv[1]
   shift
end

% script aaa bbb ccc
3 aaa
2 bbb
1 ccc
```

CASE STATEMENTS

-
```
switch (str)
case pat1:
   commands
   breaksw
case pat2:
   commands
   breaksw
:
default:
   commands
endsw
```

- EXAMPLE

```
#!/bin/csh
#
switch ($argv[1])
case ”-n”:
   echo ”Case 1”
   breaksw
case ”-s”:
   echo ”Case 2”
   breaksw
default:
   echo ”Invalid option”
endsw

% script -n
Case 1
% script -s
Case 2
% script -r
Invalid option
```

NUMERIC EXPRESSIONS

- @ followed by a space signifies that strings should be interpreted as numbers

```
@ count = 1       # note space between @ and shell
                  variable
echo $count
```

- Arithmetic operations

```
@ a = $b + 5
@ a = $b - 5
@ a = $b * 5
@ a = $b / 5
@ a = $b % 5      # modulo arithmetic
```

- C language type expressions

```
@ a += 5       a = a+5
@ a -= 5       a = a-5
@ a *= 5       a = a*5
@ a /= 5       a = a/1
@ a++          a = a+1
@ a--          a = a-1
```

OTHER ASPECTS ABOUT SHELL NOT CONSIDERED

o Can include csh options at start of shell script (e.g. -f does not source .cshrc before execution and speeds up script)

```
#! /bin/csh -f
```

o tee

```
% ls tee grep name
                    # output from ls is sent to screen and
piped
                    to grep
```

o	glob wordlist	do filename expansion on wordlist, \not recognised
o	exec command	execute command inplace of shell

o Interrupt handling (e.g. CTRL-C)

onintr label	go to label on interrupt
onintr -	disable interrupt handling
onintr	enable interrupt handling again

o Labels and goto

```
goto label
:
:
label:
```

o continue (while, foreach)

```
while (expression)
   :
   :
   if (expression) continue
end
```

o Moving between directories using a stack

pushd directory	change directories and save current on stack
dirs	show directory stack
pushd +n	change to directory n on stack (starts at 0)
pushd	switch top two directories
popd [+n]	remove directory n[0] from in definition stack

VISUAL INTERACTIVE EDITOR

- Starting

  ```
  % vi file
  ```

- Quit with save

  ```
  :wq
  ```

- Quit without saving

  ```
  :q!
  ```

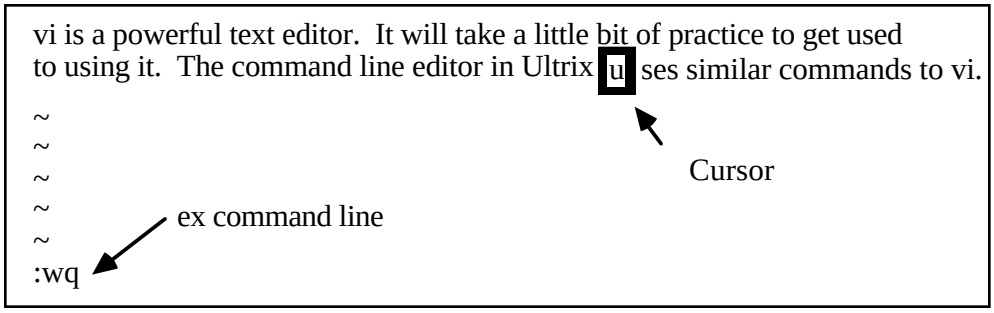
- 3 modes

  - command mode
  - insert mode

    Terminated by ESC (F11)

  - ex mode

    ```
    :command
    ```



Some overall principles

Objects in vi

characters	
word	w
end of line	$
beginning of line	^
next paragraph	}

Repeating commands

ndd	repeats the dd (delete command n times) command e.g. 5dd, 3x
.(dot)	repeats the previous command
u	undoes previous command

Movement

Character

←↓↑→	left, right, up, down
h, j, k, l	left, down, up, right
SPACE	right

Word

w, W, b, B	word forward/backward
e, E	end of word
)(	beginning next current sentence
}{	beginning next current paragraph

Line

0, $	first, last position of line
^	first non-blank char of line
+ -	first character next, previous line
RETURN	next line
n|	column n of line
H	top of screen
M	middle of screen
L	last line of screen
nH	n lines from top
nL	n lines from bottom

Screen

CTRL-F CTRL-B	scroll up/down screen
CTRL-D CTRL-U	scroll half screen up/down
CTRL-E	scroll one more line at bottom
z RETURN	reposition line with cursor at top of screen
z .	reposition line to middle of screen
z-	reposition line to bottom
CTRL-L	redraw screen

Line numbering

CTRL-G	display current line number
nG	move to line n
G	move to last line in file
:n	move to line n

Insert mode

a	append after cursor
A	append at end of line
i	insert before cursor
I	insert at beginning of line
o	open line above current line
O	open line below current line
ESC	end insert mode
RETURN	move down one line
BACKSPACE	move back one character
CTRL-H	move back one character
CTRL-I	insert tab
CTRL-U	delete line
CTRL-V	quote next character (useful for inserting form-feed (CTRL-V CTRL-L, explicit carriage return CTRL-V CTRL-M or bell CTRL-V CTRL-G)
CTRL-W	move back one word

Deletion

x	delete character
X	delete previous character
dd	delete line
dw	delete word
d$	delete to end of line
d^	delete to start of line
d}	delete to next paragraph
d/pat	delete next occurrence of pattern
dn	delete up to next occurrence of pattern
dfa	delete up to and including a on current line
dta	delete up to but not including a on current line
dL	delete to last line of screen
dG	delete to end of file

Undeletion

u	undelete

Change and replacement

CC	change line
CW	change word
C$	change to end of line
R	rewrite over text
r	substitute character
s	substitute character
S	substitute entire line

Copying and Pasting Text

Any delete text is put into temporary buffer (yank buffer)

Y	copy current line to new buffer
yy	copy current line
ye	copy to end of word
y}	copy to next paragraph
″xyy	yank line to buffer x
″xd	delete into buffer x
″Xd	delete and append into buffer x
″xp	put contents into buffer x
P	insert after cursor
p	insert before cursor

Searching

/text	search forward for pattern
n	repeat search
N	repeat search in opposite direction
/	repeat forward search
?	repeat previous backward search
?text	search backward
%	find match of current parenthesis, brace, bracket
fx	move search forward to x on current line
Fx	move search backward to x on current line

Marking

mx	mark position with character x
`x	move cursor to mark x
′x	move to line containing mark x
``	return to previous mark
′′	return to start of line
d′x	delete text to mark x

Miscellaneous commands

```
J                 join lines
:j!               join preserving spaces
<<                shift left 8 (default) spaces
>>                shift right 8 (default) spaces
CTRL-I            insert tab
CTRL-T            move to next tab
```

ex mode

```
:w                        write file
:w file                   write to file
:w!                       overwrite file protection
:q                        quit
:q!                       quit without saving
:wq                       save and quit
:r file                   read file
:r !command               read output of command

:!command                 shell escape
:sh                       create shell

:1,$s/pattern1/pattern2/  substitute first occurrence on
                          line of pattern1 with pattern2
                          from first line (1) to last line
                          ($) of file
:1,$s/pattern1/pattern2/g substitute each occurrence on
                          line of pattern1 with pattern2
                          from first line (1) to last line
                          ($) of file

:ab in out                abbreviate out with in
:ab                       list abbreviations
:map c sequence           char c used as sequence of commands
```

Unused characters

```
g K q V v
CTRL-A CTRL-K CTRL-O CTRL-T CTRL-W CTRL-X
_*\=
```

set commands

:set	show changed options
:set all	show all options
:set x?	show value of option x
:set x	set (enable) option
:set nox	unset (disable) option
:set x=val	set value for option which requires value

Examples of set options

autoindent	in insert mode line up line at same level as line above/below
beautify	ignore all control characters (except tab, newline, form feed)
ignorecase ic	disregard case during search
number nu	display line numbers
tabstop 8	
wrapscan	searches wrap around either end of file
wrapmargin=0	define right margin (if non-zero automatically insert line)

o Can be included in .exrc

```
set autoindent
set nowrapscan
set wrapmargin = 72
ab ITW Information Technology Workshop
map v dwElp
```