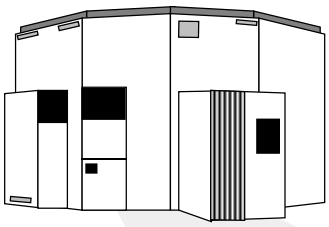
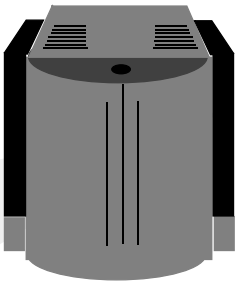
Naval Research Laboratory

Washington, DC 20375-5320





Introduction to UNIX

Course Notes

Instructor: Michael G. Vonk
Center for Computational Science
(202)767-3884
michael.vonk@nrl.navy.mil

Introduction to UNIX

1. Introduction

This introductory level course is designed to provide new UNIX users with an understanding of basic concepts and fundamentals. The course begins by covering the necessary background information you must have to effectively use the system. You will learn about:

- the various versions of UNIX and shells
- basic UNIX processing—from logging in, setting up your terminal, issuing commands, and getting help, to logging out
- UNIX directory structure and file naming conventions

The remainder of the course discusses actual commands, including how to:

- perform basic file and directory manipulation operations
- create and modify files using the editor
- share and protect files
- control input/output and command processing
- customize your UNIX environment using initialization files
- use electronic mail

2. UNIX History, Versions, and Shells

History of UNIX

- Developed in 1969 at AT&T Bell Laboratories
- Became widely available in 1975
- Distributed at low cost to universities--graduating students brought UNIX into commercial world
- Designed to run on a wide range of systems

Versions of UNIX

- Berkeley UNIX (BSD)
- AT&T UNIX (System V)

Shells (user interfaces)

- Serve as a command language interpreter, providing a customizable user interface and programming language. Two common shells are:

C shell	Commonly used by general users
Bourne shell	Commonly used for system administration

The C shell is covered in this course, although the same concepts generally apply to other shells as well.

3. Sample Session

3.1. Logging In

```
CS/200T> connect amp
Connecting... session 1 -- connected to amp

UNIX(r) System V Release 4.0 (amp)

login: stu01
Password: xxx              (password does not echo on screen)

    message of the day     (from file /etc/motd)
%
    .
    .
    .
% logout
```

Prompts: % Default prompt for C shell (/bin/csh)
$ Default prompt for Bourne shell (/bin/sh)

3.2. Changing Shells

```
$ chsh
Changing login info for stu01 on amp.
Old Shell:  /bin/sh
New Shell:  csh
$
```

This changes your default login shell and does not take effect until the next time you log in. Not all versions of UNIX (including the CCS Training workstation) allow users to change their default shell , contact the system administrator.

3.3. Terminal Setup

Terminal Type

```
% set term=vt100
```

Terminal Options

The `stty` command sets or reports terminal I/O options:

```
% stty [-a] [option]
```

`-a` show all settings

`stty` is commonly used to change the keys used to erase (delete) characters or lines and interrupt processes:

```
% stty erase '^?' kill '^U' intr '^C'
```

3.4. Changing Passwords

Passwords should be immediately changed whenever you first login to a new system and periodically thereafter.

```
% passwd
Changing password for stu01 on amp.
Old password:
New password:
Retype new password:
%
```

Note—The CCS Training workstation, "amp," uses NIS for account administration. Use the following:

```
% yppasswd
```

3.5. Help

The `man` command can be used to reference the online UNIX reference manual. Information can be viewed on:

- Specific commands
- General topics (using the `-k` option)

Examples

To view information on the `passwd` command:

```
% man passwd
```

The `man` command uses the `more` utility to output text. At the `more` prompt, press the space bar to go to the next page, the return key to go to the next line, or '`q`' to quit.

To view all commands related to passwords:

```
% man -k password
```

Section 5.2 explains how to use the `more` command.

3.6. Logging Out

```
% logout
% Ctrl-D
```

4. Commands, Files, and Directories

4.1. Command Syntax

Basic command syntax

```
% command [options] [arguments]
```

Examples

```
% ls -a
% cat test.dat
% cp -i test.dat old.dat
```

Options usually may be concatenated

```
% ls -a -l -F
% ls -alF
```

4.2. File Types

Ordinary	Programs, text,data, binary files (executables)
Directory	Contain indexes to all files and subdirectories within the directory
Special	Device drivers for terminals, disks, printers, and tape drives

4.3. File Naming Conventions

- Filenames can be up to 14 characters in length (and longer on most systems)
- All characters other than / are legal
- Certain characters, known as metacharacters, should not be used, as they have special meaning to the shell, including spaces, tabs, backspaces, and the following:

  ? @ # $ ^ & * ( ) [ ] \ | ; ' " < > { }

- Avoid using a +, -, or . . as first character in a file name

Notes

- UNIX is case sensitive—upper and lowercase letters are distinct in file names, command names, passwords, etc.
- There are no file "types" incorporated into the file name as in other operating systems. However, some utilities expect filenames to end in a particular extension. For example, the C compiler (cc) expects source file names ending in ".c"
- There are no multiple versions of files in UNIX as there are in some other operating systems. Be careful not to accidentally overwrite existing files when editing, copying, and renaming.

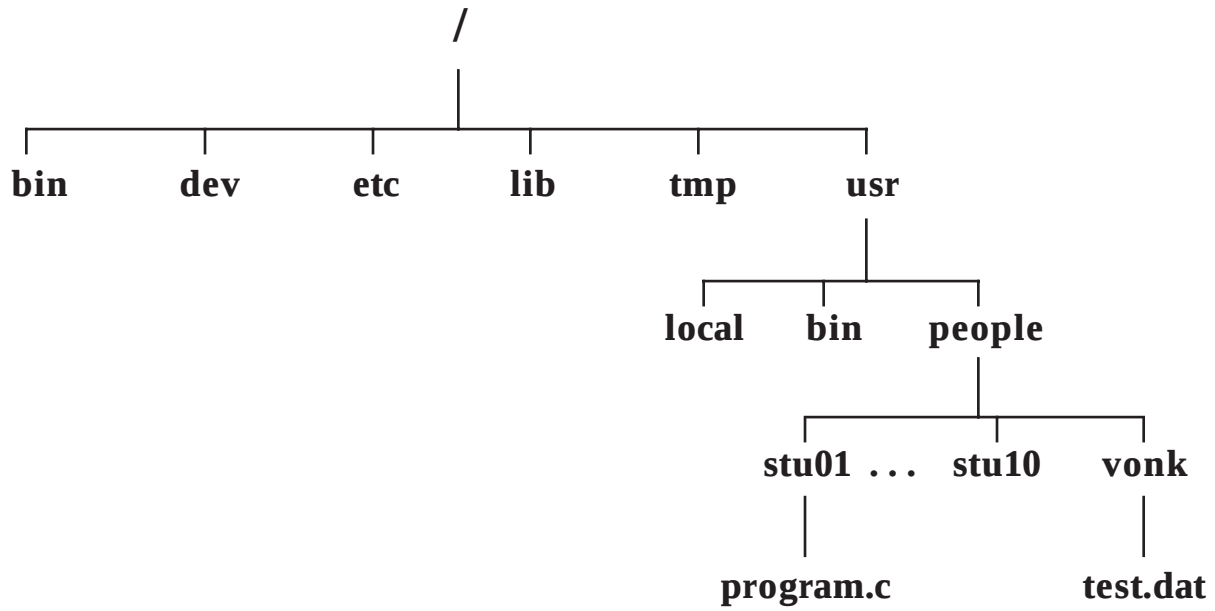
4.4. Wildcards

*	matches any string of characters (except for filenames beginning with ".")
?	matches any single character
[*ccc*]	matches any character in the list
[*lower-upper*]	matches any character in range
{*pat1, pat2, ... ,patn* }	matches any of the specified patterns
~	used as shorthand for your login directory name
~*username*	used as shorthand for *username*'s login directory

Examples

```
% ls test*
% ls test?.dat
% ls ~stu01
```

4.5. Directory Structure



Standard directories	`/` Root directory `/bin` Executable programs and utilities `/lib` Program and language libraries `/dev` Device drivers `/etc` System administration programs and data files `/tmp` Temporary files `/usr` User directories
Current working directory	Your location in the file system at a given time
Home directory	Your home directory is your login directory. For example: `/usr/people/stu01` or `~stu01`

4.6. Pathnames

There are two ways to specify a file:

- Full path name
- Relative path name

Full path name

- Each directory along the path from the root directory to the desired file is specified
- Must begin with /, with directories separated by /

Example

```
/home/vonk/test.dat
```

Relative path name

- Relative to current directory
- Use the following shorthand notations:

.	denotes current directory
..	denotes parent of current directory

Examples

```
test.dat
./myprog
../stu01/program.c
```

4.7. The Path Variable

Whenever you enter any UNIX command, you are actually specifying the name of an executable file located somewhere on the system. The system goes through the following steps in order to determine which program to execute:

1. Built in commands (such as `cd` and `history`) are executed within the shell
2. If an absolute path name (such as `/bin/ls`) or a relative path name (such as `./myprog`), the system executes the program from the specified directory
3. Otherwise the `path` variable is used

The `path` variable tells the system where to find command executables and is set as follows:

```
% set path=(directory ... directory)
```

Directories are named in the order you wish them searched. For example:

```
% set path=(/bin /usr/local/bin .)
```

Once the `path` is set, you can execute any program, including executables of your own, by simply entering their filename:

```
% ls
% myprog
```

Note—Since you are specifying the name of an actual file, commands cannot be abbreviated.

5. File Manipulation Commands

5.1. Listing Files

Format

```
% ls
% ls filename
% ls directory
```

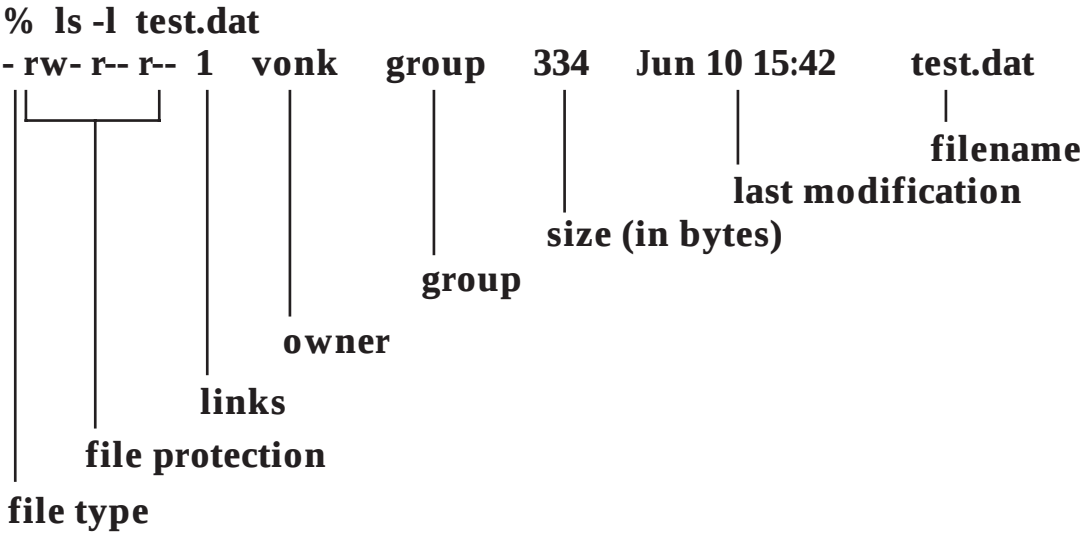
`-a` list all entries, including the "`.`" entries

`-l` long list showing protection, links, owner, size in bytes, date and time of last modification

`-F` places "`/`" after directories, "`*`" after executables and "@" after links

`-R` recursively lists subdirectories encountered

`-t` sorts files by time modified instead of by name



5.2. Displaying Files

The `cat` command displays a file:

```
% cat filename
```

The `more` command displays a file one screenful at a time:

```
% more filename
```

After each screenful of text, `more` shows the percentage of the file shown so far and prompts for one of the following commands:

`<return>`	display next line
`<space>`	display next screenful
`b`	display previous screenful
`q`	exit from `more`
`/ pattern`	search for next occurrence of *pattern*

5.3. Copying Files

The `cp` command can be used to:

- Copy one file to another file
- Copy one or more files to a directory
- Copy an entire directory structure (using the `-r` option)

Format

```
% cp file1 file2
% cp file1 [file2 ...] directory
% cp -r directory1 directory2
```

Notes

- if target filename already exists, it will be overwritten
- use the `-i` option to prompt for confirmation in case target file exists and is about to be overwritten (answer "`y`" or "`n`")

5.4. Moving (Renaming) Files

The `mv` command can be used to:

- Move (rename) one file to another file
- Move one or more files to a directory
- Move an entire directory structure

Format

```
% mv file1 file2
% mv file1 [file2 ...] directory
% mv directory1 directory2
```

Notes

- if target filename already exists, it will be overwritten
- when moving directories, if target directory exists, the original directory is moved to a subdirectory of the target directory
- use the `-i` option to prompt for confirmation in case target file exists and is about to be overwritten

Examples

```
% mv test.dat myprog.dat
% mv program.c sub1.c sub2.c source
% mv class intro-unix
```

5.5. Removing (Deleting) Files

The `rm` command can be used to:

- Remove files
- Remove entire directory structures (using the `-r` option)

Format

```
% rm filename
% rm -r directory
```

`-r` recursive directory removal (should be used with caution—see `rmdir` for removing directories)

`-i` prompts for confirmation ("`y`" removes file, any other letter will not remove file, only first character in answer is signicant)

Notes

- write permission on directory is required
- read or write permission on the files themselves is not necessary (in this case, the file permission is displayed and you are prompted for confirmation of deletion)
- can not remove "`.`" or "`..`"

5.6. Printing Files

The `lpr` command is used to print files:

```
% lpr filename
```

This prints *`filename`* to the default system printer. To print on another printer, use the `-P`*`printer`* option. For example:

```
% lpr -Pduplex filename
```

The default printer can be set using the `PRINTER` environment variable:

```
% setenv PRINTER printer
```

The `lpstat` command displays the status of your print jobs:

```
% lpstat
```

To delete a job from the print queue, use the `lprm` command:

```
% lprm job-id
```

Using the `-a` option on the `lpstat` command displays the status of all printers:

```
% lpstat -a
```

6. Directory Manipulation Commands

6.1. Creating Directories

The `mkdir` command is used to create directories:

```
% mkdir directory
```

Example

```
% mkdir class
```

`mkdir` requires write permission in the parent directory.

6.2. Changing Directories

The `cd` command changes your current working directory:

```
% cd directory
```

Example

```
% cd class
```

`cd` used without a directory name changes directory to your login directory:

```
% cd
```

6.3. Displaying Your Current Directory

Your location in the file system at a given time is called your current working directory.

The `pwd` (print working directory) command displays the full path name of your current directory:

```
% pwd
```

6.4. Removing (Deleting) Directories

The `rmdir` command removes the specified directory:

```
% rmdir directory
```

The specified directory must be empty before it can be removed.

7. Miscellaneous Commands

7.1. Listing Users

`% who`	identifies all users currently logged in
`% whoami`	identifies the current session

7.2. Finding Files

The `find` command can be used to search the file system, starting at a specified directory, for files that match a specified pattern:

```
% find directory -name filename -print
```

The `-print` option must be used for results to be displayed. For example:

```
% find . -name myprog.c -print
```

Operations can be performed on the files found using the "`-exec`" option. For example:

```
% find ~ -name core -exec rm {} \;
```

The command following "`-exec`" must be terminated with a quoted semicolon. See the `find` man page for other options.

Caution `find` performs a recursive search of the file system—this can consume a large amount of processing time if done from high up in the directory structure.

7.3. Searching Within Files

The `grep` command can be used to search for a specified string in a set of files:

```
% grep string filename
```

All lines in the specified files containing *`string`* are displayed. `grep` is case sensitive—use the `-i` option to ignore case.

7.4. Sorting Files

The `sort` command can be used to sort files:

```
% sort [-o output-file] filename
```

See the `sort` man page for more information on `sort` options.

7.5. Comparing files

The `diff` command displays differences between pairs of files, producing a list of lines that must be changed (`c`), appended (`a`), or deleted (`d`) to make the first file match the second. Lines from the first file are prefixed by "<", lines from the second by ">":

```
% diff filename1 filename2
```

The `-b` option ignores trailing blanks and treats all other strings of blanks as equivalent. The `-i` option removes case sensitivity.

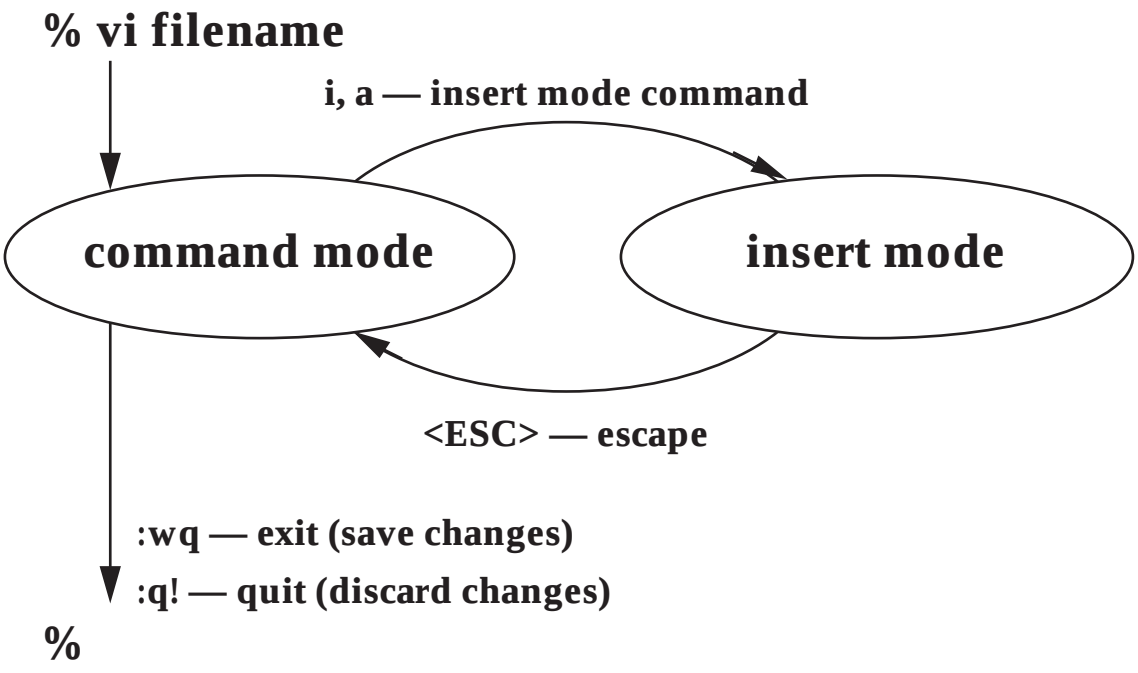
The `cmp` command compares two files, and if different, displays the position of the first difference:

```
% cmp filename1 filename2
```

8. vi Editor

Full screen editor available on almost all UNIX systems. vi is different than most editors in that it has two modes:

- Command mode Keystrokes used as commands
- Insert mode Characters inserted into file



Invoking and Terminating

Invoking	`% vi` *`filename`*	
	Make sure terminal setup is correct (`%set term=vt100`)	
	`% vi -R` *`filename`*	invoke `vi` in read only mode
Terminating	`:wq`	exit `vi`, saving changes
	`:q!`	quit `vi`, without saving changes
Recovering Lost Editing Sessions	`% vi -r` *`filename`*	recover *`filename`* after system crash
	`% vi -r`	list all recoverable files
Initialization	`.exrc`	initialization file
	`EXINIT`	initialization environment variable

Cursor Movement

Moving by characters	`h` (or backspace)	move left one character
	`j`	move down one line
	`k`	move up one line
	`l` (or space)	move right one character
	Arrow keys may also be used	
Moving by lines	`1G`	go to first line
	`n``G`	go to *`n`*th line
	`G`	go to last line
	`^`	move to beginning of line
	`$`	move to end of line
Moving by words	`w`	move forward one word
	`b`	move backward word
Moving by screens	`Ctrl-f`	move forward one screen
	`Ctrl-b`	move backward one screen
	`Ctrl-d`	move down one half screen
	`Ctrl-u`	move up one half screen

Inserting Text

Inserting Text	The following commands place you in insert mode: `i` insert text before cursor `a` append text after cursor `I` insert text at beginning of line `A` append text at end of line `o` insert text in new line below present line `O` insert text in new line above present line Use `Escape` to return to command mode
Replacing Text	`r` *`char`* substitute *`char`* for current character `R` overwrite text (end with `Esc`) `cw` change current word (end with `Esc`)
Setting Margins	`:set wrapmargin=`*`x`* Sets right margin to *x*, automatic word wrap

Deleting, Copying, and Pasting Text

Deleting Text	`x` delete character `dd` delete line *`n`*`dd` delete *n* lines `dw` delete word *`n`*`dw` delete *n* words `dG` delete to end of file `d1G` delete to beginning of file
Yanking (Copying) Text	`yy` yank current line *`n`*`yy` yank *n* lines `yw` yank word *`n`*`yw` yank *n* words
Pasting Text	`p` put most recently deleted or yanked characters or words after cursor; put lines below current line `P` put most recently deleted or yanked characters or words before cursor; put lines above current line

Searches and Substitution

Simple Searches	`/`*`string`* search forward for *`string`* `?`*`string`* search backward for *`string`* `n` repeat search `N` repeat search in opposite direction
Global Search	`:g/`*`string`*`/`*`command`* execute *`command`* on lines containing *`string`* `:v/`*`string`*`/`*`command`* execute *`command`* on lines not containing *`string`*
Substitution	`:`*`x`*`,`*`y`*`s/`*`oldstring`*`/`*`newstring`*`/`*`flags`* Substitutes *`newstring`* for *`oldstring`* in line range *x*,*y*. *`flags`* (optional): `c` wait for confirmation (`y` or `n`) `g` replace all occurrences within specified lines (default is only first occurrence) Range specifiers: `.` current line `$` last line `%` entire file
Search Metacharacters	`[`*`ccc`*`]` matches any specified character `[^`*`ccc`*`]` matches any character except those specified `[`*`c1-c2`*`]` matches any character in range `^` matches beginning of line `$` matches end of line `.` matches any one character `*` matches zero or more occurrences of previous character `.*` matches any number of characters
Case Sensitivity	`:set ignorecase` disable case sensitivity `:set noic` enable case sensitivity

Miscellaneous Commands

Undoing Commands	`u` `U`	undo last change undo recent changes made to current line
Repeating Commands	`.`	repeat last command that made an editing change
Reading in Files	`:r` *`filename`*	read *`filename`* into file (insert below current line)
Shell Commands	`:!`*`command`* `:sh` `:r !`*`command`*	execute shell *`command`* invoke subshell (use the `exit` command to return to `vi`) read shell *`command`* output into file (insert below current line)
Joining Lines	`J`	join next line to current line

Line Numbers	`Ctrl-g`
	
	`:set number`
	`:set nonumber`
		display current line number and number of lines in file
		
		display line numbers
		remove displayed line numbers

Screen Refresh	`Ctrl-l`	refresh screen

9. File Protection

9.1. User and Permission Types

User types

- `u` file owner (u is short for user)
- `g` group members
- `o` all other users

Permission types

- `r` allows users to read or copy a file
- `w` allows users to write to, modify, or copy a file; you must have write access to a directory to delete its file, regardless of an individual file's protection
- `x` allows users to execute a file (for directories, execute permission allows users to use the directory name in a pathname)

9.2. Displaying File Protection

The `-l` option on the `ls` command displays a file's protection:

```
% ls -l test.dat
-rw-r--r--  1    vonk     group    334   Jun 10 15:42     test.dat
```

This shows that `test.dat` is a regular file whose owner can read and write it and the group and all other users may read it.

9.3. Changing File Protection

The `chmod` command is used to change file protections:

```
% chmod [ugo][+-=][rwx] filename
```

Examples

To add execute access to file program for its owner:

```
% chmod u+x myscript
```

To subtract read access from group and all other users:

```
% chmod go-r confidential.msg
```

To set read and execute access for all other users:

```
% chmod o=rx myscript
```

A numeric method can also be used for specifying permissions:

```
% chmod nnn filename
```

where *nnn* comes from the following table:

	user	**group**	**other**
r	400	40	4
w	200	20	2
x	100	10	1

9.4. Default Protection

The `umask` command is used to set a default protection for newly created files:

```
% umask nnn
```

The `umask` value is subtracted from 666 to determine a new file's protection—and subtracted from 777 for newly created directories.

You can set your `umask` in your `.login` file.

Example (using `umask` value of `022`)

Default file protection	Default directory protection
666	777
- 022 umask	- 022 umask
644 permission	755 permission

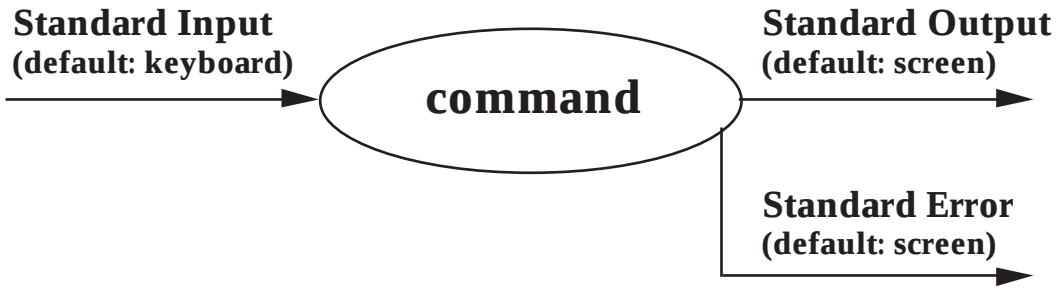
The `umask` command can be used without specifying a value to show the current `umask`:

```
% umask
```

10. Command Processing

10.1. I/O Files Redirection

Normally, a command read its input (if any) from the keyboard, and writes output and error messages to the screen.



Redirection Operators

The C shell alllows users to easily redirect input and output:

`% command < file` redirects standard input from *file*

`% command > file` redirects standard output to *file* (if *file* already exists, it is over-written)

`% command >& file` redirects both standard and error output to *file* (if *file* already exists, it is overwritten)

`% (command > file) >& errfile`
redirects standard output to *file* and error output to *errfile* (if *file* or *errfile* already exist, they are overwritten)

Output can be appended to existing files using the following:

`% command >> file` appends standard output to *file*

`% command >>& file` appends both standard and error output to `file`

To prevent existing files from being accidentally overwritten, the `noclobber` variable can be set:

```
% set noclobber
```

Setting the `noclobber` variable only affects output redirection, it has no affect on files overwritten using `cp`, `mv`, etc. An exclamation point following the redirection operator overrides the effect of the `noclobber` variable:

`% command >! file`

Examples

```
% cat file1 file2 > file3
% cat > file
% cat
% ls -al > listing
% program < input.dat > results.dat
```

Pipes

Output from one command to be piped into another command as follows:

```
% command1 | command2
```

Output from *command1* is piped into input for *command2*.

This is equivalent to, but more efficient than:

```
% command1 > temp
% command2 < temp
% rm temp
```

Examples

```
% ls -al | more
```

```
% who | sort | lpr
```

```
% man -k network | more
```

10.2. History

The history mechanism keeps track of previous commands:

```
% set history=n
```

The `history` command used by itself displays previous commands and event numbers:

```
% history
```

Re-executing previous commands:

`!!`	refers to the previous command
`!`*n*	refers to *n*th command
`!-`*n*	refers to *n*th most recent command
`!`*string*	refers to the most recent command beginning with *string*
`!?`*string*`?`	refers to the most recent command containing *string*

The following allows you to substitute one string for another in the previous command and re-execute it:

```
% ^old^new^
```

Saving the history list:

```
% set savehist=n
```

The last *n* history entries will be saved in a "`.history`" file in your home directory and made available to your next session.

10.3. Command Aliasing

Command aliases have the following format:

```
% alias alias-string command-string
```

alias-string can then be used on the command line as a shorthand for *command-string*. For example:

```
% alias ls ls -alF
% alias lo logout
```

`alias` with no arguments shows all existing aliases—`alias` with one argument shows a specific alias.

The `unalias` command is used to remove aliases:

```
% unalias lo
```

11. Job Processing

Two basic types of jobs:

Interactive	Executed either in the foreground (default) or background
Batch	Submitted using `cron`, `at`, or `batch` (see `man` pages for details)

11.1. Background Jobs

Interactive jobs may be put into the background by following the command with an ampersand:

```
% command &
```

For example:

```
% find / -name core -print >& core.lis &
```

The advantage of background jobs is that you do not have to wait for the command to finish before entering new commands, thus freeing the terminal for other processing.

11.2. Background I/O

By default, all output from a background job is written to the screen. Redirection can be used to prevent output from interfering with other commands.

Input to a background job comes from the keyboard and can only be entered while the job is in the foreground.

11.3. Job Control

The `jobs` command lists all current jobs:

```
% jobs
```

Jobs can be terminated using the `kill` command:

```
% kill %job-number
% kill process-id
```

A job running in the foreground can be stopped by issuing ^Z:

```
% program > results.dat
        .
        .
        .
^Z
stopped
%
```

Jobs can be moved between foreground and background:

`% fg %job-number`	brings job to the foreground (useful when background job needs input from the terminal)
`% bg %job-number`	sends job into the background

11.4. Process Status

The `ps` command displays information about currently executing processes:

`% ps`	lists all your running processes
`% ps -ef`	lists all running processes

12. Initialization Files

.cshrc Automatically executed every time you log in, issue a shell escape, or execute a shell script. Must begin with "#" character to indicate C shell script.

```
#!/bin/csh
set path=(/bin /usr/local/bin .)
set noclobber
set history=40
set savehist=20
#
alias cp cp -i
alias mv mv -i
```

.login Automatically executed (after `.cshrc`) upon login. Must begin with "#" character to indicate C shell script.

```
#!/bin/csh
set term=vt100
stty erase '^?' kill '^U' intr '^C'
umask 022
```

Generally, environment variables should be set in `.login` and `alias` and `set` commands should be in `.cshrc` so that every new copy of the C shell will be able to use them.

.logout Automatically executed at logout.

```
#!/bin/csh
cd
rm -r core
```

13. UNIX Electronic Mail

Several UNIX mail utilities exist, two of which are:

- Berkeley Software Distribution (BSD) Mail
- AT&T UNIX mail (more basic)

BSD mail is located in different directories on different systems. On the CCS Training workstation, "`amp`", it is `/usr/ucb/Mail`, and is invoked as follows:

```
% Mail
```

On other systems, BSD mail may be `/usr/bin/mailx`.

13.1. Email Addresses

Electronic mail addresses have the following forms:

1.	*username*	to user on same system
2.	*username*@*host*	to user on host in same domain
3.	*username*@*host*.*domain*	Internet address format

If you have a preferred email address in the NRL Locator database (see your Administrative Officer to update your address), mail can be addressed to you at:

first.*last*@nrl.navy.mil

13.2. Sending Mail

The following format is used to send mail:

```
% Mail address-list [< message-file]
```

For example, to send a mail message, entered from the keyboard, to user `smith` on the local system, the following would be used:

```
% Mail smith
Subject: Today's meeting
We are having a meeting at 1:00 PM in the conference room.
                                        -Your boss
<CTRL-D>
%
```

You will be prompted for a subject, after which the message can be entered, followed by `<CTRL-D>`.

Input redirection is used to send a message contained in a file, as follows:

```
% Mail jones@ccf.nrl.navy.mil < test.dat
```

You will not be prompted for a subject. The `-s` option could be used to include a subject:

```
% Mail -s "Test" stu01 < test.dat
```

To send mail to multiple users, simply specify each address separated by spaces, as shown below:

```
% Mail smith stu01@ra < test.dat
```

Distribution lists may be set up to mail messages to groups of users. They take the form:

```
alias distribution user1 user2 ... usern
```

Mail addressed to "*distribution*" will then be mailed to each user listed, as shown below:

```
% Mail distribution < test.dat
```

Distribution lists such as this should be placed in the mail setup file "`.mailrc`".

13.3. Reading Mail

To read your mail, invoke the mail utility by itself (without an address list), and a list of your messages will be displayed.

```
% Mail
Mail version SMI 4.0 Thu Oct 11 12:59:09 PDT 1990  Type ? for help.
"/usr/spool/mail/smith": 3 messages 3 new
>N  1 stu01               Fri Sep 11 14:21   11/311
 N  2 stu01               Fri Sep 11 14:23   52/1616
 N  3 smith               Fri Sep 11 14:39  858/28029
& q
Held 3 messages in /usr/spool/mail/smith
%
```

The message header list shows the following:

- message status—"N" for new, "U" for unread—the current message is indicated by a ">"
- message number
- sender
- received date
- message size (lines / characters)
- message subject (if there is one)

If you have no new or unread messages, the mail utility will not be invoked and a message will be displayed.

Previously read messages are stored in the file "~/mbox". To read them, use the following:

```
% Mail -f
```

The following table lists common Mail commands:

<cr>	display current message
d [*message-list*]	delete messages
h	display active message headers
m [*user-list*]	send mail to specified users
n	display next message
p [*message-list*]	display current (or specified) message
q	quit, saving unresolved messages in mbox
r [*message-list*]	reply to sender (only) of messages
R [*message-list*]	reply to sender and all recipients of messages
s [*message-list*] *file*	save messages to *file* (append to file if file already exists); if |*command* is used in place of *file*, then the message will be piped into *command*
u [*message-list*]	undelete messages
v [*message-list*]	edit messages
w [*message-list*] *file*	save messages to *file*, without from line
x	quit, do not change system mailbox
!	shell escape
?	display help information

A [*message-list*] consists of integers, ranges of same, or user names separated by spaces. If omitted, Mail uses the current message.

13.4. Mail Setup Files (.mailrc)

The file "`~/.mailrc`" serves as a mail initialization file and typically contains the following:

- distribution lists
- mail variable settings

The following example shows a typical `.mailrc` file:

```
alias  class stu01 stu02 stu03 ... stu12
set askcc    # set Mail to prompt for carbon copy list
```

Fully qualified addresses should be used in distribution lists for portability.

The system file `/usr/lib/Mail.rc` is used as a global setup file by the BSD mail utility.

13.5. Mail Forwarding

Mail forwarding can be set up by creating a file named "`~/.forward`". For example:

```
smith@ccfsun.nrl.navy.mil
```

Mail can be forwarded to multiple users by placing their addresses on a single line separated by commas.