

The University of Maine
Department of Computer Science
UNIX User's Guide

Geoff Davis¹

11 June 1991

Last Revised 1 January 1999

¹Last Edited by Jonathan Thomas

Contents

1	Startup	1
1.1	Logging On	1
1.2	Changing Your Password	1
2	Where To Find Help	2
3	Basic Directory Commands	4
3.1	Home Directory	4
3.2	Listing Directory Contents	5
3.3	Changing Directories	6
3.4	Creating Directories	7
3.5	Directory Removal	7
3.6	Searching Directories	8
3.7	Archiving Directories	8
3.8	Disk Usage	9
4	Basic File Commands	10
4.1	Displaying Contents of Text Files	10
4.2	Naming Files	11
4.3	Copying Files	11
4.4	Moving/Renaming Files	12
4.5	Removing Files	13
4.6	Changing File Permission Modes	13
4.6.1	chmod	13
4.6.2	umask	14
4.7	File Compression	15
5	Creating And Editing Files using VI	16
5.1	Starting VI (The <i>visual</i> Display Editor)	16
5.2	VI modes	16
5.3	The Essential VI Commands	16
5.4	Using VI	19
5.5	VI User Variables	19
5.6	VI quirks	20
6	Creating And Editing Files Using EMACS	21
6.1	Starting emacs	21
6.2	Escape and Control Sequences	22
6.3	Character Operations	22
6.4	Word Operations	22
6.5	Line Operations	23
6.6	Screen Operators	23
6.7	File Operations	23
6.8	Regions	24
6.9	Searching	24

6.10 Multiple Buffers	24
7 Document Preparation	25
7.1 nroff and troff	25
7.2 L ^A T _E X	26
8 Checking The Spelling Of Documents	27
9 Printing Text Files	27
10 Unix C Shell Interface	29
10.1 C Shell Initialization Files	29
10.2 Shell Commands	30
10.3 Default File Handles	31
10.4 Redirection of Output	32
10.5 Using Pipes	32
10.6 Unix Filters	33
10.6.1 grep	33
10.6.2 sort	34
10.7 Job Control: Background and Foreground Processes	35
10.8 C Shell Programming	36
11 X-Windows	36
11.1 Changing Default .xsession	36
11.2 Using the X-Windows Environment	37
12 Network Commands	42
12.1 Who is using the Network?	42
12.2 MAIL	42
12.2.1 Berkeley Mailer	42
12.3 TALK	43
12.4 Remote login	44
12.5 File Transfer Program (ftp)	44
12.6 Network News	46
13 Compilers	47

1 Startup

1.1 Logging On

In order to login to the University of Maine UNIX cluster you need to enter your logon ID and password. When your account was issued your password was set to your student or employee ID number, without the dashes. Enter your logon ID, hit RETURN and then enter your password and hit RETURN again to logon to the system.

Please note: For security reasons, your password will not appear on the screen as you type it.

If you make a typing mistake in either your logon ID or your password, type the backspace key to erase previous characters. If you get either logon ID or password incorrect, you will be prompted again for both until you get them correct.

Please remember that UNIX is CASE SENSITIVE. That means you must enter all characters in the proper case. ALL user accounts are LOWER case and must be typed in as such. For example, user001 and uSeR001 are NOT the same in UNIX.

If you are on a SUN workstation, after a successful logon, you are presented with a number of X-terminal windows on your screen. To enter a command, move the mouse *pointer* into a window to bring that window into *focus*. Before a window is selected, the *pointer* is either an arrow or a hollow "X" depending on where it is on the screen. When a window is put in focus, the *pointer* looks like a capital "I" and the *text* cursor changes from a hollow rectangle to a solid one. You can customize the X screen as you like and have any number of programs and windows execute upon login. See the section on *CustomizingX – Windows* for more information.

If you are on a Silicon Graphics workstation, then you will also be presented with a number of windows. In order to bring a window into focus on the SGI's, you need to move the mouse pointer over that window.

1.2 Changing Your Password

The password file that is used to verify your password is shared by all Unix workstations, Sun's and SGI's. You can only change your password on a Sun machine - if you try to change it on a SGI or Linux machine you will get a message explaining that you must change it from a Sun.

You should change your password immediately after logging on to any system for the first time. You can use the **passwd** command to change your login pass-

word at any time. To do so, simply type **passwd** and respond to the questions that follow. You will be asked to supply your OLD password for verification and then your new password. After entering the new password, you must type it in again to verify that you typed it correctly. If the passwords do not match, you will be reprompted to enter a new password, and then to verify it again. If you typed in your old password incorrectly, the system will reject your request for a new password.

A password may be composed of as few as five characters if those characters selected are a combination of both upper and lower case. The minimum length for a password is six characters if it consists entirely of either upper or lower case characters. With the exception of the password that is issued with your account, entirely numeric passwords are not allowed, at least one non-numeric character must be provided. Passwords are significant up to 8 places, any extra characters are ignored. You should use passwords that contain both alpha and numeric characters, or a combination of alpha characters and special characters (\#%\$&(), etc). You should avoid passwords that are common English words, any word that can be looked up by a dictionary-checking program, or any password that someone who knows you can guess (no spouse names, etc). Your password is the only protection you have against other people using your account. It should be changed frequently. You should never write down your password or tell it to anyone. If you should forget your password, you may request that another one be set for you. You will need to provide proper identification when requesting a new password.

Please be sure to protect your password and account as you would protect any other possession of value. Easy to guess passwords are a tremendous security risk for both you and other users of the system. If your account presents a security risk to this system because of an insecure password, systems staff may temporarily disable your account.

2 Where To Find Help

The UNIX system provides on-line manuals for most commands, services, system calls, and other useful material. Most of the time, these on-line manual pages are exactly what you would get if you purchased a copy of the printed manuals for the system, in fact, most UNIX vendors won't even sell you hard-copy manuals anymore. To access the manual pages enter:

```
man < subject >1
```

This will display manual pages for *subject*, providing that it exists. Keep in mind that searches done by **man** are case-sensitive (as are all UNIX commands,

¹Many systems alias the *man* command to *fem*, for political correctness

filenames, etc.); a search for "GCC" will yield a "No manual entry" message, while a search for the command "gcc" will open the appropriate manual page.

To look for a command or manual page when you don't know what the command is called, you can do a manual page lookup by keywords found in the command or subject's description.

```
man -k < keyword >
```

An alternative way to do the keyword search is with the **apropos** command, it is equivalent to **man** with the *-k* argument:

```
apropos < keyword >
```

For information on how to use the *man* command itself enter:

```
man man
```

Sun's and Linux

Pressing < *CR* > causes the manual pages to scroll one line at a time while < *spacebar* > scrolls one page at a time. < *ctrl - b* > returns to the previous page.

SGI's

Pressing < *CR* > causes the manual pages to scroll one page at a time. Typing *-1* returns you to the previous page.

While reading a manual page, you can search for a specific word in the manual by entering a slash (/) followed by the word you wish to locate.

Under X windows you can use the *xman* menu item, or type **xman** to get windows based access to manual pages.

Printing a man page is somewhat complicated. First you need to find the man source which will be under */usr/man/man*/**. For example the man page for **ls** is */usr/man/man1/ls.1v*. Copy this file to your home directory:

```
cp /usr/man/man1/ls.1v
```

and then login to a Linux machine, if you aren't already:

```
rlogin acadia
```

and run **groff** on the file:

```
groff -man -Tps ls.1v > ls.ps
```

This will create a PostScript version of the man page that you can print on the laser printers in our labs. See the section on printing for more info on how to print.

Also, most Linux systems come with other documentation like FAQ's and howto's under `/usr/doc`.

3 Basic Directory Commands

3.1 Home Directory

When you login to the system, your current working directory is set to your home directory, which is where all of the files that you own reside. It is named `/directoryname /accountname`, where *accountname* is your logon ID.

Many UNIX commands and file operations make reference to your home directory. You should always reference your home directory by using by using a *tilde* "`~`". For example, the command:

```
cat ~/.cshrc
```

is the same as typing:

```
cat /sulu/myuserid/.cshrc
```

Similarly, the command:

```
cd ~/bin
```

sets your current working directory to the directory *bin* in your home directory.

You can of course reference your directory as `/directoryname /accountname`, but if for some reason you were to *hardcode* this into a program, and your files were to be moved to `/newdirectory /accountname`, you would have to modify all of your programs. The "`~`" eliminates this requirement and will *always* work as a reference to your home directory.

3.2 Listing Directory Contents

To list the contents of a directory use the **ls** command. The **ls** command issued alone, without specifying a directory, causes a listing of all subdirectories and non-hidden files in the current working directory to be displayed. By default files are listed in alphabetic order in a series of columns across the screen. Alternative ways of displaying directory contents can be specified by using command-line arguments to the **ls** command. One reason for providing arguments is that the default listing does not distinguish between subdirectories text files and other special files. To do this, use the *-F* option of the **ls** command:

```
ls -F
```

The resulting list of files denotes subdirectories with a trailing slash ('/'), executable programs with a trailing asterisk ('*'), symbolic links with a trailing at-sign ('@'), and sockets with a trailing equals sign ('='). Hidden files, files beginning with a '.', are still not displayed; use the *-a* argument (for list *all* files) if you wish to list hidden files as well. To obtain specific information about files and directories, such as mode, size in bytes and time of last modification, use the *-l* argument with the **ls** command.

All of the various arguments available for use with any UNIX command can be combined into one character string for all arguments to take effect simultaneously, ie: **ls -alFC**. Full descriptions for all of the **ls** flags and options can be found under *man ls*.

The following *aliases*² are available to make the **ls** command more friendly:

- The command **ls** has been aliased to **ls -FC**
- The command **la** is the same as issuing **ls -a**
- The command **ll** is the same as issuing **ls -al**

Pattern matching can be applied to **ls**, as well as to any other UNIX command. For example, if you wished to list only files that end with the '.c' extension:

```
ls [-options] *.c
```

The asterisk 'wildcard' character matches *any* characters. To look for all files with the same name but different extensions, do a listing for *< filename > .**.

Listing can be done on directories other than the current one by specifying a path to the directory after the listing options:

²For more information on command aliasing, see the C Shell Interface section on command aliases

```
ls [-options] ~/backup/*.c
```

This command would give a listing of all files with a *.c* extension in the subdirectory *backup* off of your home directory. For more information on *WildCards* see the C Shell Interface section on *WildCards*.

3.3 Changing Directories

The **chdir** command is used for changing from one directory to another. The command has a shorter form **cd** which is used more often. There are two ways to specify the directory you wish to change to:

1. Absolute Pathnames

An absolute path is one that is specified completely from the root directory:

```
cd /sulu/usr
```

The *'/'* character before the *sulu* directory name indicates that the path is absolute. The *'/'* refers to the system *root* directory, ie: the top level directory.

An absolute path that doesn't really look like it's absolute is any path that begins with *'~/'*. The *'~'* character is the complete path from the root directory to your home directory. This form is very useful when you wish to ascend from deep within your directory tree, but you don't want to type an absolute path from the root. It is also useful when you are elsewhere in the system and you wish to read or write files to one of your subdirectories.

2. Relative Pathnames

Relative pathnames assume the current directory as the base from which paths to other directories are specified:

```
cd pascal/lib
```

The lack of a preceding *'/'* character before the *pascal* directory name means that the path is relative to (*below*) the current directory. To move *up* in a directory tree, use *'..'*. Every UNIX directory has a file named *'..'* in it, which refers to the *parent* directory. Thus to move to the directory right above your current directory, issue:

```
cd ..
```

To move up two directory levels issue:

```
cd ../../
```

Using `..` you can ascend to the parent directory, then descend into another child directory with one statement. If your current directory were `/sulu/c++/lib`, then issuing:

```
cd ../bin
```

would make your current directory `/sulu/c++/bin`

3.4 Creating Directories

The UNIX file system allows for the creation of subdirectories to help organize files according to purpose or similar function. To create a subdirectory, use the **mkdir** command:

```
mkdir <dirname>
```

Issued without a path, **mkdir** creates a subdirectory of the current working directory. The command **mkdir** can be used to make a subdirectory anywhere in the directory tree by specifying a path before the directory name:

```
mkdir ~/smalltalk/source/<dirname>
```

Giving the **mkdir** command in this way assumes that you have already created the parent directories specified in the path. **mkdir** will return an error if any of the parent directories do not exist. If you wish **mkdir** to create all the directories in the tree that you specified, use the `-p` option.

3.5 Directory Removal

To remove a file use the **rm** command:

```
rm [options] filename
```

To remove a file you need only have write permission on the directory that contains the file, write permission on the file itself is not necessary. If you wish to remove a file that has write protection, a warning message asking you to confirm deletion will appear, but you are allowed to delete the file. To delete a file that exists in a write-protected directory, you must change the protection mode of the directory first.

To prevent accidental file removal, the *-i* option should be used. This will cause **rm** to ask you if you really wish to delete the file. Remember, once a file is removed, it can not be recovered. You may wish to alias **rm** to '**rm -i**' and **cp** to '**cp -i**' and **mv** to '**mv -i**'. More information on the **alias** command can be found in the section titles "Unix C Shell Interface" under Shell Commands.

3.6 Searching Directories

The **find** command can be used to locate files within in a directory tree. Searching for a file by name is done in the following way:

```
find < directoryname > -name < filename > -print
```

This will cause a search for the filename starting at the directory given. After searching the parent directory, **find** will recursively descend into any subdirectories of that directory. The *-print* option tells **find** to print the absolute path to all of the occurrences of the filename in the directory tree. As an example, to search the current directory and all subdirectories for a file named "core" you would issue the following command:

```
find -name core -print
```

If the file is not found, then **find** returns to the command prompt without doing anything. Use the **-ls** option to list file attributes and sizes of the files that **find** found. The **find** command can be used to find files of a specific type, with specific attributes, or a combination of attributes.

The **find** command can also be used to remove files matching a pattern, or to execute a specific command using the found files as input *-exec option*.

See **man find** for more details and options to the **find** command.

3.7 Archiving Directories

Entire directory structures may be archived to a single file with the **tar** command. The format for archiving with the **tar** command is:

```
tar cvf filename.tar directory1
```

Where *filename.tar* is the name of the resulting tar file and *directory1* is the name of the directory that you wish to archive. Note that the original directory is not affected by the archiving operation, the archive is a *copy* of all the files and information contained in the directory.

The format for unarchiving a tar file is:

```
tar xvf filename.tar
```

The directory structure that was saved to the tar file is restored to the directory where the unarchiving command is given.

Some of the more useful options for tar are:

- **c** : Create an archive.
- **x** : Extract files from an archive.
- **t** : Table of contents of files contained in the tar file.
- **p** : Restore files with original permission mode, ignore current umask.
- **f** : Use a file instead of default system tape drive. The argument after *-f* is the name of the file to be used.
- **v** : Verbose. Have tar tell you what it is doing.

Care should be used when *untar*'ing a file to disk as it may clobber existing disk files you may have.

3.8 Disk Usage

Each user is assigned a disk quota which limits how much they may store in their home directory. To view your quota issue the following command from a Sun.

```
quota -v
```

This will produce output similar to the following:

```
Filesystem usage  quota  limit  timeleft  
/sulu          9091   9000   50000  1.42 weeks
```

Usage is your current disk usage in kilobytes, *quota* is the soft quota for your account. *Limit* is your hard quota. *Timeleft* will appear only if your account is over its disk quota; it refers to the amount of time you have to get your disk usage down below your allowed quota. If you fail to reduce your disk usage within the *timeleft* period, you will not be able to save files in your home directory until you get below your hard quota.

A user may exceed his soft quota for short periods of time. When you reach this limit, the system will give you a warning every time you try to create or

write a file. You can not exceed your hard limit, ever. Once the hard limit is reached, the system will not allow you to create any files, or save any edit sessions.

Please remember that disk space is a finite and limited resource which all users share. We do not have nearly enough disk space for all users to be at their soft quotas, and we certainly don't have enough for many to be at their hard limits. Requests for increased disk quotas will only be accepted if you can demonstrate genuine need and can show that your current usage is justified.

To see how much disk space is taken up by the contents of a directory, use the **du -s** command. If given alone, without specifying a directory, the default is to list the disk usage (in kilobytes) for the current directory. (To get output in kilobytes on the SGI's use **du -sk**.) The number given is the total disk usage of all files and subdirectories. If you include a target directory, the same listing will occur, except that the listing will start with the target directory. The **du** command is often useful when you get warnings from the system telling you to reduce your disk usage by a certain amount.

4 Basic File Commands

4.1 Displaying Contents of Text Files

The contents of a text file can be viewed one page at a time by using the **more** command:

```
more textfilename
```

A page will be displayed at your screen, and **more** will then pause with the message '-More-' followed by a percent sign and a number. The number represents the percentage of the file that you have already seen, i.e. '-More-(5%)' means that you have 95% more text to view. Commands can be entered while **more** is paused. Some of the commands to control **more** are as follows:

- $(n) < CR >$
Scroll forward n lines at a time. Default is one line.
- $< spacebar >$
Advance to next page of the file.
- $(n)b$
Page back n pages. Default is one page. This doesn't work on the SGI's or Linux. See below for use of *less* and *pg*
- q
Quit showing the document.

On Linux machines you may want to use the **less** command which supports moving backwards in a file. The commands are the same as for **more**. On the SGI's you may want to use the **pg** command which supports moving backward in a file:

- $(n) < CR >$
Advance to next page of the file.
- $-n$
Page back n pages.
- q
Quit showing the document.

Another way of displaying text files is with the **cat** command:

```
cat textfilename[textfilename..]
```

The name 'cat' comes from 'concatenate', and that is exactly what **cat** does when supplied with multiple text files. When used with only one file argument, the **cat** command concatenates that file with a null file, then sends the file to stdout, the terminal screen. In English, this displays the contents of the file on your screen. Displaying files with **cat** has the disadvantage of not stopping the scroll at every screenful, you must use $< ctrl - s, ctrl - q >$ to stop and start the output respectively. To join two or more files into a single file, issue:

4.2 Naming Files

When creating or renaming a file, you need to be aware of the file naming conventions. Filenames on SUN UNIX are significant up to 255 characters, but for portability and ease-of-use you should try to keep your filenames below thirty characters in length. Note that non-alphabetic characters such as a period ('.') can occur more than once in a filename string (unlike the MSDOS filenames that you may be familiar with). Standard file extensions are used to denote different types of files, i.e. *.c* for C language source, *.o* for object files, *.a* for library files, *.p* for pascal, *.f* for fortran, etc. Files should always be named as descriptively as possible to enable you to determine file contents from the name. Subdirectories should be used for organizing related files.

4.3 Copying Files

There are three basic forms of the copy command:

- **cp** [*options*] fromfile tofile
- **cp** [*options*] fromfile todirectory

- `cp -r [options] fromdirectory todirectory`

The first form (with default options) creates a new file at the location specified by the *tofile* path, or it overwrites a file of the same name if that file already exists (and that existing file has a mode that allows writing). To prevent the unwanted destruction of a same-named file, use the *-i* option. This option causes an interactive prompt to appear if there are any filename conflicts.

The second form is like the first, except that no filename is specified, just a target directory. In this case the source file is copied to the target directory, and the name of the source file is preserved. Again, if a file already exists in the target directory that has the same name as the source file, it will be overwritten unless you specify the *-i* option.

The third form works two ways to recursively copy all files and subdirectories in the source directory to the target directory. If the target directory exists before the copy, then the source directory will become a subdirectory of the target directory. If the target directory does not exist, then the source directory is simply copied and given the *todirectory* name at the destination. Be as careful of recursive copying as recursive deletion, since a statement like the following will cause a recursive copy that won't stop until the disk is full or until you have exhausted your disk quota:

```
cp -r lisp/tools lisp/tools/parser
```

4.4 Moving/Renaming Files

Moving a file to a new location and renaming a file are the two functions of the `mv` command:

- `mv [options] fromfile tofile`
- `mv [options] fromfile todirectory`
- `mv [options] fromdirectory todirectory`

The first form relocates the sourcefile to the location specified by the targetfile. If the sourcefile's name already exists at the destination, and the existing file's mode allows writing, then the existing file will be overwritten. If either interactive mode has been selected (*-i* argument) or the existing file is write-protected, then the user will be prompted to make certain that overwriting the existing file is the action intended.

The second form is like the first, except that no filename is specified, just a target directory. In this case the source file is moved to the target directory,

and the name of the source file is preserved.

If the target directory exists, the third form moves the source directory so that it is a subdirectory of the target. If the target directory does not exist, then the source directory is moved to the path specified in naming the target directory, and is given the target directory's name.

In both the first and third form, if the destination file/directory is in the same directory as the source file/directory, and the source and destination names are different, then the **mv** command has the effect of renaming the file/directory.

4.5 Removing Files

To remove a file use the **rm** command:

```
rm [options] filename
```

To remove a file you need only have write permission on the directory that contains the file, write permission on the file itself is not necessary. If you wish to remove a file that has write protection, a warning message asking you to confirm deletion will appear, but you are allowed to delete the file. To delete a file that exists in a write-protected directory, you must change the protection mode of the directory first.

To prevent accidental file removal, the *-i* option should be used. This will cause **rm** to ask you if you really wish to delete the file. Remember, once a file is removed, it can not be recovered. You may wish to alias **rm** to '**rm -i**' and **cp** to '**cp -i**' and **mv** to '**mv -i**'. More information on the **alias** command can be found in the section titled "Unix C Shell Interface" under Shell Commands.

4.6 Changing File Permission Modes

4.6.1 chmod

Every UNIX file has a protection mode associated with it. Listing the contents of a directory with **ls -l** command displays the protection mode for each file. The mode is composed of ten flags divided into four groups:

Directory	User	Group	Others
d	rx	rx	rx

The *Directory* flag indicates whether or not a name represents a directory. *User* is a category that refers to the file/directory owner. The category *Group* refers to a group of users that you might belong to, such as COS398. This protection category is provided because you may wish to let other users in your group have access to certain files, but exclude people outside of that group. *Others* refers to all the other users that aren't you, or members of your group.

The three permission types are **read**, **write**, and **execute**. To change permissions:

```
chmod ugo ± rwx filename/directoryname
```

or

```
chmod 3 – digitoctalnumberfilename/directoryname
```

The first form uses the 'u','g', and 'o' characters to specify user, group, and others, respectively. By typing the following,

```
chmod go-w afile1
```

the write privileges are removed from the group and others. Other combinations which add and subtract privileges at once are possible:

```
chmod o+r-w afile1
```

This allows others to read *afile1*, but not write to it.

The second form of chmod is the *absolute* form, which uses octal digits to define the permission mode. Each octal digit represents the three modes of the user, group, and other categories. Each bit within the octal digit corresponds to the read, write, and execute flags.

```
chmod 755 afile1
```

The above command would change a file so that it had the following permission mode:

Directory	User	Group	Others
	rwX	r X	r X

4.6.2 umask

The umask command changes the default protection mode for all files and directories subsequently created. umask is initially called from within the .cshrc initialization file to set your default protection value. To use umask, you provide a three-digit octal number that has bits set where you want permissions to be removed:

```
umask 033
mkdir pascal
```

The directory *pascal* will be created with the following mode:

Directory	User	Group	Others
d	rx	r	r

umask 077 (the usual setting in *.cshrc*)
gcc -o life life.c (produces an executable file, *life*)

The file *life* will have the following protection mode:

Directory	User	Group	Others
	rx		

4.7 File Compression

The **compress** command is used to reduce the size of a file to approximately one-third its original size. This can be useful to reduce the size of *.tar* files (which can become extremely large) as well as any other large file that you aren't currently using but need to keep online. The format of the **compress** command is simple:

compress filename

This produces a file with the same name as the original, but with a *.Z* extension. Unlike the **tar** command, which archives without deleting the target directory, **compress** *replaces* the original file with the compressed version.

To restore a compressed file to its original uncompressed state:

uncompress filename

Some systems don't have a **uncompress** command, but instead use **compress -d** to uncompress a file.

Files ending in *.gz* are files that have been compressed using **gzip**, another compression program. To unzip these files, use the **gunzip**.

gunzip filename.gz

A file that ends in *.tar.gz* or *.tgz* is a compressed tar file. You need to unzip the file and then **untar** it.

5 Creating And Editing Files using VI

5.1 Starting VI (The *visual* Display Editor)

To begin editing a file (either one existing or one you wish to create) type,

```
vi [pathtofile] < filename >
```

If the path to the file isn't included, **vi** assumes that the current directory will be used.

5.2 VI modes

vi has two different modes. One is Command Mode, the other is Text Entry Mode:

- Command Mode
In Command Mode everything that you type is interpreted as a command. For most commands, a single keystroke is the entire command, while other commands require more information, such as a linecount followed by a single-character command. Command mode is used to perform powerful editing functions, as well as such actions as writing the text buffer to a file. To enter Command Mode from Text Entry mode, type an *< ESC >* character.
- Text Entry Mode
In Text Entry mode all characters typed are printed on the screen and become part of the current document. Editing in this mode is restricted to insertion and typeover. To enter Text Entry Mode from Command Mode, type the character 'i' (for insert mode), 'a' (for append mode), or 'o' (for open mode). The significance of the different modes will be discussed later in the section 'Using VI'.

5.3 The Essential VI Commands

1. Cursor Movement Commands

- ←↓↑→h j k l
Moves cursor in direction of arrow in command mode.
- < *linenum* >G < CR >
Goes to beginning of < *linenum* > line in document.
If < *linenum* > isn't included, goes to end of document.
- w b e
Skip forward one word, back one word, or to the end of the word in a sentence.
- 0 (zero)
Sends cursor to the beginning of a line.

- \$
Sends cursor to the end of a line.

2. Screen Management

- H,M
Move cursor to the Home position (Top of screen).
Move cursor to the Middle position (Halfway down screen)
- < *ctrl-f* >, < *ctrl-b* >
Move Forwards or Backwards one screenful.
- < *ctrl-y* >, < *ctrl-e* >
Scroll screen up or down one line.
- < *ctrl-l* >
Redraw screen. Useful for cleaning up screen if mode information on the lower right of the screen starts to be written in strange places or if a message from another process or from the system is written to your screen.

3. Text Change Commands

- r < *char* >
Replace character under cursor with < *char* >.
- R
Replace (typeover) all existing characters.
Press < *ESC* > to stop typeover.
- <<, >>
Shift entire line of text to the left or right.
- J
Append the following line to the current line.
- ~
Change the case of the character under the cursor and move right one position.

4. Marking Text

- m < *lowercasechar* >
Mark location with a mark named after the lower case character.
- ' < *lowercasechar* >
Goto line marked with < *lowercasechar* >.

5. Text Searches

- / < *targetstring* > < *CR* >
Forward search for target string.

- ? < *targetstring* >< *CR* >
Backward search for target string.
- n N < *targetstring* >< *CR* >
Repeat last search for Next occurrence of target string.
'n' is a forward search, 'N' is a backward search.
- " (Two single quotes) Returns cursor to spot where search began.

6. Text Deletion

- [*count*]x
Deletes count characters from cursor position. Default is 1 character.
- [*count*]dw
Delete count words from cursor position. Default is 1 word.
- [*count*]dd
Delete count lines from cursor position. Default is 1 line.
- D
Delete from cursor position to end of line.
- u
Undo last editing change.

7. Text Buffers

- < *numberoflines* > Y
Yanks lines of text into a buffer.
- p P
Puts text after (p) or before (P) current line.

8. File Manipulation

Note: to use a colon (:) command, get into command mode then press ':'.

- :r filename < *CR* >
Reads in a copy of the file specified on the line after the current line.
(if no filename is given, the file you are editing is read in from disk)
- :w < *CR* >
Write edit buffer to original file (This assumes that you started **vi** by typing **vi** < *filename* >).
- :w filename < *CR* >
Write edit buffer to named file.
- :wq < *CR* > or ZZ < *CR* >
Write edit buffer to original file, then exit **vi**.
- :e newfilename < *CR* >
Start editing a new file.

- `:q < CR >`
Quit with a warning if edit buffer not saved.
- `:q! < CR >`
Quit with no warning. If edit buffer has not been saved to a file, it will be lost.

5.4 Using VI

The screen you are initially confronted with is a mostly blank page with a series of tilde characters down the left side of the window and a filename at the bottom left corner. The tilde characters represent the end of your text buffer; you will see the column of tilde characters get pushed farther down the screen as you begin to fill the buffer.

The initial mode of **vi** is command mode. To begin editing an existing document you can stay in command mode, but if you wish to start a new document you must switch to text-entry mode. To do this, press either `'i'`, `'a'`, `'o'`, or `'O'`. The character `'i'` selects *insert* mode, which causes characters to be inserted at the current cursor position. `'a'` selects *append* mode, which moves the cursor one character to the right before inserting text. `'o'` and `'O'` stand for *open* mode; selecting the lower/uppercase letter will "open" an empty line after/before the current line and place the text cursor at the beginning of this line. The important thing to remember is that all of these modes are equivalent after the initial selection of the mode. All modes act as insertion mode after they perform their initially different behaviors.

After selecting a text-insertion mode, you can begin typing your document. It is important to realize that the document that you are editing exists only in memory as a text-buffer, it won't be placed on disk unless you explicitly put it there. This feature allows you to completely screw up a document, then back out of the problem by exiting without saving the document. If you have made editing changes that you don't wish to save, then you can type `:q!` to throw the current buffer away, leaving the document on disk untouched. If for some reason the system crashes while you are in **vi**, you will receive mail from the system explaining that if you issue `vi -r < filename >` you can recover your edit session, with generally no loss at all.

5.5 VI User Variables

vi has a series of variables that can be set to customize it. To see these variables while editing a document, give the command `:set all'`. This will cause a list of the variables to be displayed on the screen. Variables are set with the `:set'` command followed by the variable you wish to change. Some variables are set equal to a value, as with the `marginlength` setting below, while some, like the `autoindent` feature, are either on or off.

To change the default word wrap margin:

```
:set wm=< marginlength >< CR >
```

To change the default setting for autoindentation (a useful feature to enable while programming):

```
:set ai< CR >
```

To turn off a boolean option, set the variable again as *no*< *variablename* >.

To change the shift-width variable, which is used with the <<, >> commands:

```
:set sw=< shiftwidth >< CR >
```

These commands can be performed every time you start **vi** by setting the variables within the **vi** initialization file *.exrc* in your home directory. A typical setup as written in the *.exrc* file is shown below:

```
set autoindent
set nowrapscan
set shiftwidth=4
set modeline
set showmode
set report=1
set novice
set noterse
set wrapmargin=10
```

For more information on these, and other **vi** variables, issue *man vi*.

5.6 VI quirks

One of the first things you will notice about **vi** is that it likes to beep a lot and it tends to switch modes unexpectedly. **vi** will produce an error beep every time you do something that is not an option, such as paging past the end of the text buffer. There is nothing particularly wrong with that, except that **vi** often does more than just beep. As an example, if you try to move the cursor past the invisible end-of-line characters using the arrow keys, **vi** will beep and then switch to command mode. This wouldn't be that much of a problem if the mode indicator in the lower right corner always worked correctly, but it doesn't. Sometimes it works impeccably, but often it will say that it is in one of the text entry modes when it is actually in command mode. The danger here is that command mode, as mentioned before, interprets all keystrokes as commands. So if you are unexpectedly thrown into command mode and you keep typing, a

series of powerful editing commands may destroy the document you are working on. This possibility reinforces the importance of backing up your documents frequently, so that you have the option of quitting without saving the buffer to disk. If `vi` beeps at you when you are in a text-entry mode, assume that you've been put into command mode, and reselect the text-entry mode by pressing the appropriate character. At worst, that character will be inserted into your document, where it can be deleted later. The consequences of one keystroke while in command mode can be far more destructive and time-consuming to undo.

Another quirk you may notice is that the up and down arrow keys do not work in text-entry mode; rather, they insert a line above where you are typing and put a capital letter on that line. You will need to exit text-entry mode, and delete the lines (with `'dd'`) to get rid of them, but otherwise your document will be unharmed.

6 Creating And Editing Files Using EMACS

EMACS is different from VI in that it is not a mode-based editor. Instead, it relies on control and escape-key sequences to perform the editing functions that VI performs within command mode.

6.1 Starting emacs

EMACS is invoked by giving the command:

```
emacs [pathtofile] < filename >
```

If *pathtofile* isn't included then the current directory is assumed.

When EMACS begins, a text window will appear that has a mode line on the bottom identifying what file is currently being edited. There is also a mode listed on the lower right which depends on the extension of the file you are editing. If you are editing a file with a standard extension such as `.c`, which refers to a text file of the programming language C, then EMACS provides enhancements such as bracket indentation to make program structuring easier. Files with no standard extension have the default mode 'Fundamental'.

There is a command line called a 'minibuffer' which pops up below the mode line to prompt for input after an escape or control sequence that requires additional information, such as a filename.

Unlike VI, EMACS is always in text-entry mode. To begin writing your document, just start typing. Editing is done with control or escape sequences.

6.2 Escape and Control Sequences

EMACS commands are either escape sequences or control sequences. To enter an escape sequence, press and release the $\langle ESC \rangle$ key, then enter the character that represents the command. An escape sequence to convert a word to uppercase would be:

$\langle ESC \rangle u$

Control sequences are issued by pressing and holding the control key, then entering a character command. A control sequence to delete a character under the cursor is:

$\langle ctrl - d \rangle$

A command can be executed a number of times in succession by specifying a repeat count. As an example, to delete 10 characters in a document, issue the following command:

$\langle ctrl - u \rangle 10 \langle ctrl - d \rangle$

The ' $\langle ctrl - u \rangle \langle num \rangle$ ' specifies the number of times the following command is to be repeated.

6.3 Character Operations

- $\langle ctrl - b \rangle$
Move back one character.
- $\langle ctrl - f \rangle$
Move forward one character.
- $\langle ctrl - d \rangle$
Delete the character underneath the cursor.

6.4 Word Operations

- $\langle ESC \rangle b$
Move back one word.
- $\langle ESC \rangle f$
Move forward one word.
- $\langle ESC \rangle u$
Convert word to uppercase.
- $\langle ESC \rangle l$
Convert a word to lowercase.

- `< ESC >d`
Delete one word to the right.
- `< ESC >< delete >`
Delete one word to the left.

6.5 Line Operations

- `< ctrl-a >`
Move the cursor to the beginning of the line.
- `< ctrl-e >`
Move the cursor to the end of the line.
- `< ctrl-p >`
Move to the previous line.
- `< ctrl-n >`
Move to the next line.
- `< ctrl-o >`
Insert a blank line into the buffer (at cursor pos.).
- `< ctrl-k >`
Kill (Delete) a line of text.

6.6 Screen Operators

- `< ctrl-v >`
Page down one screenful.
- `< ESC >v`
Page up one screenful.
- `< ESC ><`
Go to the beginning of the current document.
- `< ESC >>`
Go to the end of the current document.

6.7 File Operations

- `< ctrl-x >< ctrl-s >`
Write the current document to the file listed in the mode line.
- `< ctrl-x >< ctrl-w >`
Write the current document to a file. You will be prompted in a minibuffer for a filename.

- $\langle \text{ctrl-x} \rangle \langle \text{ctrl-c} \rangle$
Quits EMACS. If you haven't saved your document, EMACS will ask you if you wish to save the document before exiting.

6.8 Regions

Blocks of text called *regions* can be selected and manipulated using a pair of control sequences. Move the cursor to the place in the document where you want the block to begin, then issue the first control sequence $\langle \text{ctrl-} \rangle$. Move the cursor to the point where you wish the block to end, then give the command $\langle \text{ctrl-w} \rangle$. The block you have selected will disappear and will be held in what is called a *killbuffer*. To copy a block held in the killbuffer to another location, issue the $\langle \text{ctrl-y} \rangle$ command ('y' stands for "yank" back) to retrieve the text at the cursor location. The text remains in the killbuffer until another region is selected so multiple copies can be made.

6.9 Searching

EMACS provided two commands to search for strings of characters within a document:

- $\langle \text{ctrl-s} \rangle$
Search from cursor to end of document.
- $\langle \text{ctrl-r} \rangle$
Search from cursor to the beginning of document.

When either command is issued, the message 'I-search:' appears on the minibuffer command line. Type the pattern you wish to search for after the colon. If you want to repeat the search, type either $\langle \text{ctrl-s} \rangle$ or $\langle \text{ctrl-r} \rangle$, depending on the direction of the search. Enter $\langle \text{ESC} \rangle$ to exit the search process.

6.10 Multiple Buffers

EMACS allows you to edit more than one file buffer simultaneously. To start editing another buffer, issue the control sequence $\langle \text{ctrl-x} \rangle b$. You will be prompted for the name of the new edit buffer. To load a file into this buffer, enter $\langle \text{ctrl-x} \rangle \langle \text{ctrl-v} \rangle$ before entering the name of the file that you wish to load.

To alternate between edit buffers, issue the $\langle \text{ctrl-x} \rangle b$ command again and specify the buffername. To see all of the buffers that you are currently editing, type $\langle \text{ctrl-x} \rangle \langle \text{ctrl-b} \rangle$. To get rid of this information, make sure your text cursor is in the edit window (not the bufferlist window) and enter $\langle \text{ctrl-x} \rangle 1$ to focus on the window that contains the text cursor (Press left mouse button when mouse cursor is in other window or enter $\langle \text{ctrl-x} \rangle o$ to

change window focus).

To edit two buffers simultaneously, type `< ctrl - x > 2` to split the edit window into two separate windows. Use the `< ctrl - x > o` command to move the cursor between windows, then type `< ctrl - x > b` to name the buffer that should appear in the window that contains the text cursor. If the buffer exists it will be loaded into that window, otherwise a new buffer will be created. You can return to editing a single window by entering `< ctrl - x > 1` when the text cursor is over the window you wish to continue editing.

Killing a buffer, which removes a buffer without saving it to a file, is accomplished with the `< ctrl - x > k` command. The buffer that is killed is the buffer currently being edited.

Copying text between buffers is done by selecting a region of text (as detailed in the section on Emacs Regions), then switching windows with `< ctrl - x > o` to copy it where you wish in the second buffer.

7 Document Preparation

7.1 `nroff` and `troff`

A document preparation package that comes with most UNIX systems is *nroff-troff*. The command **troff** is used to process a document for viewing on a graphics terminal or a laser printer, while the **nroff** command is used to produce output on a lineprinter or a character-mapped screen. Input documents for these commands consist of text with formatting commands that indicate the typestyle and organization of the output document. A UNIX reference book or the online manual pages are places where you can learn the syntax of *nroff-troff* documents.

There is a considerable amount of documentation written using *nroff*. Even if you choose to use a more powerful text preparation system, you need to be familiar with *nroff* conversions so that the documentation provided with a program can be viewed in a readable form.

A *nroff* formatted document is recognizable by the formatting statements that start with a period, such as `'PP'` and `'TH'`.

To get an ascii approximation of a *nroff* formatted file, use the **nroff** command with output redirected to another file:

```
nroff < filename > > < outfilename >
```

This output file is a printable ascii file. View the document with the **more** command before sending it to a printer to make sure the document looks the way you want it to

If the document was formatted for use as an online manual page, use the *-man* argument for the best results:

```
nroff -man < filename > > < outfile >
*or*
nroff -man < filename > | more
```

Do not attempt to use **troff** to print documents on the laser printers here. Although the **troff** command's sole purpose is to produce documents printable on a laser printer, it is not supported on any of the Computer Science Department's printers. Attempts to use **troff** even on small 'test' documents can generate hundreds of pages of output, with only a few meaningless characters on each page, so please don't try.

7.2 L^AT_EX

LaTeX is not available on the SGI's, xtex is not available on Linux.

L^AT_EX, pronounced "Lah-Tech", is a document preparation system that is much more powerful than *nroff-troff*. To use it, you will definitely need a L^AT_EX manual. You create/edit your L^AT_EX document with either VI or EMACS, following the document syntax as described in the L^AT_EX manual, and name the file with a *.tex* extension. To view or print the document, you must first 'compile' the file with the *.tex* extension using the **latex** command:

```
latex < documentname >
```

Successful compilation will produce a file with the same name as your original document, but with a *.dvi* extension. This stands for 'device independent file' and is the form required for viewing the document with a program called **xtex**. To use **xtex** with the dvi file enter:

```
xtex < documentname >
```

The *.dvi* extension will be assumed, and a document view window will be created which shows your document as it would appear if printed on a laser printer.

To print a L^AT_EX document one more step is required. The *.dvi* file created by **latex** must be converted to a PostScript file that can be output on a laser printer. To do this, use the **dvips** command:

```
dvips -o < outdocumentname.ps > < indocumentname.dvi >
```

The *outdocumentname.ps* file can then be printed using the printing procedures described later in this manual.

To view a PostScript file run **ghostview** < *document.ps* >.

On the SGI's use:

```
xpsview < filename >
```

8 Checking The Spelling Of Documents

Documents can be spell-checked with the **spell** command. To see a list of all the words that the spellchecker doesn't recognize,

```
spell < documentname >
```

To spell-check a L^AT_EX document, use the **texspell** command:

```
texspell < documentname.tex >
```

To check the spelling of a single word or group of words on the command line (instead of from a file), enter the **spell** command followed by a carriage return, then enter the word(s) followed by an *eof* character, < *ctrl-d*>.

9 Printing Text Files

To print a document, use the **lpr** (for 'lineprinter') command. A specific printer is selected with the **-P**< *printername* > argument as follows:

```
lpr -Proom106 < documentname >
```

We currently have 2 print queues that can be accessed directly from the Sun and Linux machines. (To print from the SGI's, see the info below on printing PostScript files.) Neither of these is a PostScript printer, so *don't* send them PostScript files:

```
room119      2 Epson FX1170 dot matrix      Room 119
room106      2 Panasonic KXP1124i dot matrix      Room 106
```

Room119 is the default printer, so if you don't specify a printer name with the **-P** option to any of the **lp** commands, room119 is assumed.

To see a status report of any printer available on the system use the **lpq** command:

```
lpq -Proom106
```

Here is what you might see as the result of an **lpq** command:

```
room106 is ready and printing
Rank  Owner      Job  Files                Total Size
active ARSENA42   256  DEPREERR.CPP        60 bytes
1st   ajalbert    25   ftp.note            607 bytes
```

To delete a print job, you type:

```
lprm -P{printer} {job number}
```

So typing **lprm -P**room106 25 will remove the file **ftp.note** from the queue.

To remove all of the print jobs that you own type:

```
lprm -P{printer} -
```

Make sure you attempt the **lprm** from the same machine from which you submitted the job. You will get a **permission denied** error message if you attempt it from another machine.

Printing PostScript

To print PostScript files you need to download the file to a PC and print it from the PC. This is inconvenient but necessary because of the way we quota laser printer usage.

Log into a PC and ftp to **gandalf.umcs.maine.edu**. You can then log into your account and transfer the files you want to print. FTPing is further detailed in a separate chapter. To print the files from the PC type:

```
COPY FILENAMEPRN
```

An alternative to printing PostScript files is to view them using:

```
ghostview < filename >
```

On the SGI's use:

```
xpsview < filename >
```

Please try to only print things that you absolutely need printed. Careless printing is a waste of resources.

10 Unix C Shell Interface

The shell is a command interpreter and it is your interface to the UNIX operating system. When you log in to the system, a shell is started to receive your commands. There are many shells to choose from, all with different features and enhancements. The default shell is the Enhanced C-Shell (**tcsh**).

csh is an enhanced version of the Bourne shell, providing many new features such as aliasing, command history, and job control, to name just a few.

tcsh is the same as the **csh**, except it provides command-line recall and editing based on Gnu EMACS editing commands. **tcsh** provides many more enhancements than just command-line editing; a discussion of the options can be found in the online manual pages for **tcsh**.

10.1 C Shell Initialization Files

When you log in to the system, a series of initialization files are used to set up the environment in the desired way. The three primary initialization files are the *.login*, the *.cshrc*, and the *.xsession*.

The *.login* file is used to set global environment variables such as terminal type and is executed only once during the login process. *.login* is called ONLY when logging in via a terminal device (non X windows device).

The *.cshrc* is the C-shell initialization; it is executed every time a C-shell is created. Typically, the *.cshrc* declares directory paths, aliases, default protection modes (via *umask*), and any other environment variables that affect the operation of the shell.

The *.xsession* file is used to set up your initial screen under X-windows on the Sun's. Programs to run, window placement, size, and color is specified in *.xsession* before the last line of the file, which calls the window manager (**twm**). The last command executed from your *.xsession* must always be *twm*.

Be careful about modifying your *.xsession* and *.cshrc* files. If you change something such that their execution causes an error message, your keyboard lights will flash and you will be logged out. If this happens then you must correct the faulty *.xsession* file by telneting in from a PC or after typing your password hit the F1 key instead of *< enter >* to bypass your *.xsession* and *.cshrc* files and just open an *xterm* for you.

Put Xwindow programs that you want to run on startup into your *.xsession* file, NOT your *.cshrc* file. This is because your *.cshrc* file is run even if you are telneting in from a PC, and you can't display an Xwindow application on a

terminal!!

Note that the initialization files in your directory may be redirected to an initialization file in another directory. For example, your *.login* file should contain the line:

```
source /usr/local/lib/UMCS.login
```

This will execute the commands in the */usr/local/lib/UMCS.login* file. If you wish to tailor your *.login*, do so after the source statement!

10.2 Shell Commands

alias

The **alias** command allows commands or commands with parameters to be aliased with a shorter, sometimes single-character version of the command. To use **alias** to redefine the **ls -FC** command to the single character **l**, enter the following:

```
alias l "ls -FC"
```

Giving the **alias** command without any parameters lists the aliases that are currently defined.

Place any aliases that you plan to use often in your *.cshrc* file so you don't have to reenter them every time you login. Another reason for placing the alias declaration in the *.cshrc* is that aliases are window-specific. If you declare an alias within one window, then it is only recognized within that window. Placing an alias declaration in the *.cshrc* assures that the alias will be available regardless of the window that you are in.

A list of aliases that you should consider placing in your *.cshrc* follows:

```
alias cp cp -i
alias mv mv -i
alias rm rm -i
```

The *-i* argument specifies interactive prompting if existing data might be overwritten, a precaution that might save you a considerable amount of work in the future.

history

Commands that have already been issued are 'remembered' and can easily be repeated with a couple of keystrokes if the **history** command is used. In your

`.cshrc` file, include the statement:

```
set history=20
```

The **set** command assigns the number of commands that **history** will be able to access. Include the statement `'alias h history'` in your `.cshrc` file so that you only need to enter 'h' to get the command history list. Note that the history list is window-specific in that commands entered in one window don't show up on the history list of another.

To use the history list after following the above steps, enter 'h' after you have issued a few commands. You will see a numbered list showing the last twenty commands that you have entered. To reexecute a command, enter a bang character (an exclamation point) followed by the number of the command. Another way of reexecuting a command is to type a bang character followed by enough of a previous command to uniquely distinguish that command from others in the history list. If a previous command had been:

```
grep strcmp 3dlife.c
```

then entering **!g** would reexecute that command if no other command in the history list began with 'g'. If more than one command in the history list is described by the character or string after the bang character, then the most recent command in the list fitting that description is reexecuted.

If you just want to reexecute the last command typed, type two bang characters followed by a carriage return. In addition, by putting text after the bang characters you can modify the previous command. As an example, if you forgot to type the destination while copying a file you would get an error message telling you the proper syntax of the copy command:

```
cp /backup/*.tex <CR>          (Missing destination for copy)
```

To reexecute this command specifying the destination:

```
!!latex                        (latex is destination directory)
```

The text following the bang characters is concatenated with the previous command string and then the command is executed.

10.3 Default File Handles

The C shell maintains three files, **stdin**, **stdout**, and **stderr** that do not have to be explicitly opened by you. These files are assigned to certain devices by default. **stdin** (standard-input) corresponds to the terminal keyboard while

stderr (standard-error) and **stdout** (standard-output) correspond to the terminal screen.

10.4 Redirection of Output

Many commands assume that input will come from `stdin`, and the output (and error messages) will go to `stdout`. There are many times when you will want to change these defaults and have a command take input from a file, or send output to a file. The shell's capability for redirection allows this. The C-shell supports six different command-line operators for redirection:

<code><</code>	Redirect <code>stdin</code> to a file.
<code>></code>	Redirect <code>stdout</code> to a file.
<code>> &</code>	Redirect both <code>stdout</code> and <code>stderr</code> to a file.
<code>>></code>	Append <code>stdout</code> to a file.
<code>>> &</code>	Append <code>stdout</code> and <code>stderr</code> to a file.
<code><<< word ></code>	Reads standard input until <code>< word ></code> is encountered,

The redirection commands will create a file if it doesn't already exist; if the file does exist, and the environment variable **noclobber** is *not* set, then it will be overwritten.

As an example, suppose you want to concatenate three files and send the resulting output to a fourth file:

```
cat file1 file2 file3 > file4
```

A command that normally expects input to come from `stdin`, and normally sends output to `stdout` can be redirected to read from an input file and write to an output file as follows:

10.5 Using Pipes

Redirection works fine if you only need to redirect streams to and from files, but if you need to chain commands together that read and write to the default file handles, then you need to use *pipes*. Pipes connect the `stdout` of one command to the `stdin` of the command that is next in the chain. As an example, suppose that you wish to concatenate three files, sort the contents of the files, then display the output a page at a time on the terminal screen:

```
cat file1 file2 file3 | sort | more
```

In this example, **cat**'s stdout is piped into **sort**'s stdin, and **sort**'s stdout is piped into **more**'s stdin. Since no pipe exists after the last command, output goes to the default stdout.

If you add an ampersand to the pipe operator (`|&`), stderr will also be redirected to the next command's stdin.

10.6 Unix Filters

10.6.1 grep

The name *grep* is short for "generalized regular expression parser." The command *grep* is a UNIX filter that allows searches for regular expressions and fixed strings within ascii documents.

Regular expressions are patterns or templates that are defined by a combination of ascii strings and metacharacters. The metacharacters, characters that represent something other than their literal meaning, allow you to specify search tasks such as "Find all strings that start at the beginning of a line that contain a character sequence with three g's in it".

There is actually a family of *grep* commands, each command designed for a different task:

1. *grep* Searches for limited regular expressions.
2. *egrep* (Extended *grep*) Searches for full regular expressions.
3. *fgrep* (Fixed string *grep*) Searches for fixed strings.

Which *grep* command you need to use depends on the complexity of the search task. **fgrep** is the command to use for strings that contain no wildcards, or other metacharacters, just a single text pattern that must be matched exactly. **grep** is the command to use for general purpose searching that requires the use of wildcards and other metacharacters, specifying string position, string size, character class, or closure. **egrep** is the extended version of **grep**. It can handle expressions just like the **grep** command, but it allows more variability in the search by allowing the search pattern to be "string1 *or* string2, followed by string3 *or* string4".

The general syntax for *grep* commands that search for regular expressions is:

```
grep < expression > < filename > [< filename > ...]
```

The **fgrep** command is similar, except that it searches only for fixed strings:

```
fgrep < string(s) > < filename > [< filename > ...]
```

Grep commands can be restricted to a single filename, or can be told to search a series of files, either by listing them in order, or by using wildcard characters.

For specific details on the specific metacharacters used and options available for the *grep* commands, see the online manual page for **grep**.

10.6.2 sort

A filter for sorting alpha-numeric text fields is appropriately called **sort**. **sort** accepts input from stdin by default, so it can be used in a chain of commands, or it can accept input from a file:

```
cat < file1 > < file2 > | sort | more
```

or the equivalent:

```
sort < file1 > < file2 > | more
```

By default, the sort is done according to the character or numeric value in the leftmost column of a field. Fields are separated by tab or space characters by default, but any other field separators can be used by defining them on the command line.

Useful command-line options for **sort** are as follows:

- b Ignore leading space characters in the starting and ending positions of a field.
- d Dictionary order. Only letters, digits, space, and tab are significant in the sort.
- f Treat upper and lower case characters as equivalent.
- n Numeric sort. Sort by arithmetic value.
- r Reverse the order of the sort.
- t< c > Use the character < c > as the field delimiter.
- +sp.o sp is the starting position for the sort. +0 is leftmost field. .o is the optional character offset into a field which indicates where the sort should begin.
- ep.o ep specifies the field number before which the sort is ended. .o is optional; it specifies that the sort will end at the character just prior to the .o offset into the ep field.

EXAMPLES:

```
sort -d +0 -1 file1 | more
```

The sort will start at the first field, and end before the second.

```
sort -d +0.1 -0.2 file1 | more
```

The sort starts at the first character of the first field, and ends after that same character.

10.7 Job Control: Background and Foreground Processes

Commands that are non-interactive can be run in the background while another task is run in the foreground by appending an ampersand character to the command string. To run an X-window's based clock program (for example) in the background, while you work on something else in the foreground, the following command would be entered to display the clock:

```
xclock &
```

While you are using the current window for other purposes, the clock will continue to operate.

A list of all jobs you currently have, with information on their state (either running or stopped) is available by entering the **jobs** command.

If you wish to bring a background process into the foreground again, enter:

```
fg % < jobnumber >
```

To stop a foreground job, hit the **Control-Z** keys at the same time.

Only one job can run in the foreground at any time, while many jobs can be running in the background. To make stopped jobs run in the background, enter:

```
bg % < jobnumber >
```

To see all of the processes that you have running, use the **ps** command with no arguments. A list will be displayed showing the process id, the status of the process, and the command that started the process.

If you start a process that you need to stop, use the process id in conjunction with the **kill** command:

```
kill -KILL < processid >
```

The **kill** command terminates the process immediately. If you wish to terminate a process in the foreground the **kill** command doesn't necessarily need

to be used. Entering `< ctrl - c >` will normally kill a foreground process.

10.8 C Shell Programming

The C-Shell is more than just a command-interpretter and user interface. The shell is a programming language which can be used to create useful programs known as *shell-scripts*. Shell-scripts are analogous to batch files in MSDOS machines, but are far more flexible and powerful. Shell-scripts allow for the creation of variables and arrays, as well as providing conditional statements and a means for statement repetition. An example of a shell script follows:

```
#!/bin/csh
#
# Find any core files in current directory, or subdirectories of the
# current directory
#
find . -name core -print
```

The first line of a shell script gives the path to the shell that will interpret the program. The pound sign (#) followed immediately by an exclamation point indicates that a new shell will be created. Any text following the sequence of a pound sign and a blank space is a comment, and will be ignored by the shell. The **find** command is used to do the dirty work of identifying any directories that contain files named 'core'.

After writing this code to a file, name the file something descriptive and easy to type, like **corefind**. In order to run the script, you'll need to make the file executable, with **chmod**:

```
chmod u+x corefind
```

When running a shell script written as above, you will notice a distinct lag before the command does it's job. The reason for this lag is that the shell you are creating is performing all the initializations specified in your *.cshrc* file. You can add the *-f* option after the name of the shell to force the shell to start up with default options, thereby speeding up the execution of the program.

You can find out more about writing shell-scripts either with one of the many UNIX references, or by perusing the man pages under *csh* or *tcsh*.

11 X-Windows

11.1 Changing Default .xsession

Rarely does a default X-Windows environment keep a user satisfied for very long. A look around at other terminals will show you that people love to cus-

customize their X environments with all sorts of colors, windows, and icons. These customization options are placed in the *.xsession* file in your directory. For information on customization options, see the online manual page for *xterm*, or pick up a good X-window's reference manual. Be forewarned that as you fool around with your *.xsession* file, any statement which is not syntactically legal will prevent you from accessing your Sun account from the Sun terminals. You will log in as usual, the lights on the upper-right of the keyboard will flash, then you will be presented with the login screen once again. If you experience this after modifying your *.xsession*, you will have to telnet to your account from a dumb terminal and fix the error from there. Always make a backup of your *.xsession* before modifying it so you can quickly restore things to normal.

11.2 Using the X-Windows Environment

When X-Windows is invoked you are usually presented with at least one X-terminal window. By moving your mouse pointer over this window you select that window to receive the 'input focus' of the terminal. All keyboard input is directed to the window with the input focus. After the window is put in focus, you can enter commands at the shell prompt.

An important use of the **setenv** command is to redefine the DISPLAY variable when logging in to a remote machine. If you log in to a remote machine, then run an X-windows-based program from that remote machine, the X server needs to know where to display the window for program output. The DISPLAY variable is used for this purpose. When you login to a Xwindow based machine, the default startup files set the DISPLAY environment variable for you to the machine you have just logged into. If you rlogin into another machine or run a program remotely via rsh, then the DISPLAY environment variable should be set correctly. However you may run into some instances where this doesn't work properly and you need to set it manually. If you get an error like:

Can't open display:

then enter the following from the remote machine to set the DISPLAY environment variable:

```
setenv DISPLAY machine - you - want - to - display - on:0
```

After setting the DISPLAY variable, add the name of the remote machine to the list of those machines allowed to make connections with the X server with the command **xhost**. If remotely connected to *gandalf* the following would add *gandalf* to the list:

```
xhost gandalf
```

You can now run your X application on the remote host.

If you get an error that looks like:

```
Xlib: connection to "dori:0.0" refused by server Xlib: Client is not
authorized to connect to Server xterm Xt error: Can't open display:
dori:0.0
```

then you need to run xhost.

Modifying the current configuration of windows is done by moving the pointer out of all X-terminal windows (onto the background screen) and pressing the right mouse button. A window entitled 'System' will pop up, and by continuing to hold the mouse button down, and dragging the mouse towards you, you will highlight the various options that are in the window. Moving the mouse slightly to the right after highlighting any one of these options will show any further options under a given heading if further options exist.

Note that by pressing the right mouse button when the pointer is over the title bar on the top of a window, a window entitled 'Title Bar Window' appears. This window has the most of the options that are in the 'Window Options' sub-menu of the System window, but the options act specifically on the window whose title bar was selected.

The System window options:

1. Window Options

- Lower
Places the selected window on the bottom of the stack of windows.
- Raise
Places a selected window on the top of the window stack.
- Resize
Selecting this option allows you to change the size of any X-terminal window that you are using. After selecting this option, move the pointer over the window you wish to change and press the left mouse button. While holding the button down, drag the mouse in any direction to alter the frame size of the window. Release the left button when the window is the size that you prefer. This option also exists as the fourth icon (from the left) on the title bar. Follow the same procedure for resizing when using the icon.
- Zoom/Unzoom
This option expands a window to the maximum screen size, and back to normal if selected again.

- Half Zoom/Unzoom
Same as Zoom, except that window is expanded to half the screen size.
 - Move
Used to change the position of an X-terminal window. Follow the procedure for Resize (above) to drag window to new location. This option also exists as the second icon (from the left) on the title bar.
 - Deiconify
Not used. To deiconify a window, click the left button while the pointer is on the appropriate icon box in the upper right corner of your terminal screen.
 - Iconify
Removes a window leaving only its icon on the upper right part of the screen. Icon is marked with an 'X' if window it represents is not active. This option also exists as the first icon (from the left) on the title bar.
 - Identify
Display a message showing info about window selected.
 - Focus
Force input focus to be in a selected window irrespective of pointer position.
 - Unfocus
Unfocus previously Focused window.
 - Kill
Skull and Crossbones pointer that destroys the window beneath it if the left mouse button is pressed. This option also exists as the third icon (from the left) on the title bar.
2. Active Windows
Shows a list of all windows you currently have active when you move the pointer to the right on this option. Also provides an alternate way of deiconifying windows by clicking on the window names.
 3. OpenWindows
This menu contains a series of utility programs that you can start by dragging the highlight bar over the program name, then releasing the right mouse button.
 4. Button Help
Details the mapping of mouse buttons and meta/control key combinations to functions.
 5. Open New XTERM
Opens a new default X-terminal window.

6. Show Icon Box
Displays icon box if hidden behind other windows, or if explicitly removed via 'Hide Icon Box'
7. Hide Icon Box
Removes icon box from view. Can be brought back into view with 'Show icon Box'
8. TMW-Version
Version information on window manager.
9. Show system MOTD
Displays a window with any Messages Of The Day from the system. This message will come up automatically at login, but you can look at it again with this option.
10. Redraw all
Redraws all windows on the screen. Helpful if garbage is written on your screen for some reason. This option won't get rid of "ieback reset " and other system error messages, you'll need to log out and log back in again to get rid of these.
11. Restart Window Mgr
Restarts the Window Manager. Probably not something you will ever need to do.
12. Lock Screen
Locks your screen while displaying a moving image. Something you should do every time you leave your terminal, to prevent unauthorized access to your account. Hit a key to get the password entry screen, and enter your password to get back into your account.
13. Logout
Logs you out if you are on a Sun Terminal. Causes a lot of trouble if you are on an RT. The proper way of logging out on an RT is to press CTRL-ALT-BACKSPACE to kill the X-Windows process, then entering 'logout' on the command line.

The Tools window options can be seen by moving the pointer away from all other windows and pressing the left mouse button. The options are a series of utility programs:

1. Calculator
A standard calculator that you operate with the mouse.
2. Editors
Another menu is here; move the mouse to the right to access it. Several editors (including EMACS) are listed for your use.

3. Mail
A utility for sending mail. Many more options than command-line version.
4. XMail
Another utility for handling your mail. Some people like this better than the previous one.
5. Biff
Produces a mailbox icon that announces the arrival of mail by putting up the mailbox's flag and reversing icon's video.
6. Manual Pages
A man page reader which is prettier and more powerful than command-line version.
7. Magnify
A utility that magnifies a square region of the screen that you select.
8. Load Monitor
Displays the CPU load on your host terminal in graphical form.
9. Notepad
Notes can be written to this window to help you keep track of things during a login session. Notes cannot be saved to a file.
10. Clock
Brings up a standard analog clock.
11. Digital Clock
Brings up a standard digital clock.
12. XArchie
A utility that lets you search through for files available by anonymous ftp.
13. Xforecast
A utility that lets you get weather reports from around the country.
14. XRn
A window based form of read-news (rn). Much fancier than the command-line version.
15. Xdtm
A utility that gives you a pseudo- MS-Windows/Macintosh view of your files.
16. Xgopher
A information retrieval system, Gopher lets you quickly find information about the University of Maine and many other sites and topics.

12 Network Commands

12.1 Who is using the Network?

To determine who is using the network, enter the command **finger**. **finger** displays a list of all the users on the system, their hosts, and the date and time that they logged in. Users are listed by their logon id's, not their real names. To identify an individual (and show some other useful information about that user) enter:

```
finger -l < loginId >
```

Finger can also be used to locate another users account name. For example, the command

```
finger -l foo
```

Will show something users who have 'Foo' in their real names like:

```
Foobius Barus (foobar) Home: /sulu/foobar Last login on ttyp7 from boromir,  
on Fri Sep 6 11:41:49 1991 No mail. No plan.
```

12.2 MAIL

To communicate with another user via network mail, you need to know that user's logon id. With it, and with one of the many mail programs that exist, you can send letters to another user.

Elm and Pine are two popular electronic mail programs that are menu driven and fairly self explanatory. To run them type 'elm' or 'pine'. Pine works well with telnet and is similar to the editor Pico.

12.2.1 Berkeley Mailer

The Berkeley mailer is an effective and easy to use mailer. The mail program is invoked by entering:

```
Mail < logonid >
```

Note the capital 'M' in Mail. Using lowercase 'mail' will work, but that mail program is not as functional as Mail.

The logonid belongs to the person that you wish to send to. Once you enter Mail, you are prompted for a subject and then you are allowed to edit your

letter. Basic editing is very limited; you can't go back to edit previous lines once you've gone on to a new line. If you wish more extensive editing, issue `v` on a new line. You will then be in `vi` and able to edit the text of your mail file. Once done editing, write the file to disk and quit (via `ZZ` or `:wq`). You will then be placed back into the limited Mail editor.

Once you are finished writing your letter, sending it is done by entering `< ctrl - d >` or by hitting a period on a line by itself. If you change your mind about sending a letter, you can cancel it by entering `< ctrl - c >` twice consecutively; the letter you were about to send is placed in a file called 'dead.letter' in your home directory.

If you have already created a file you wish to mail someone, issue:

```
Mail < logonId > < < letterfile >
```

If you have mail in your mailbox when you log in, the system will inform you with the message 'you have mail'. If mail is received while you are logged in you will not be told you have mail unless you select the `XBiff` tools-window option (discussed previously) or run the `biff` command. To see your mail headers, and to read your mail, simply enter `Mail`. The header list shows the letter number, the sender, the date and time of sending, and a excerpt from the subject line. Reading mail is done by selecting the number of the letter you wish to read at the prompt. Viewing the mail list can be done again after reading a letter by entering 'h'. After you have read the letters that you wished to read, you can exit Mail by entering 'q'. All letters that have been read are appended to a file called 'mbox' in your home directory. If you have questions about mail while you are using it, enter 'help' at the prompt; a complete list of commands will be displayed.

12.3 TALK

An alternative to mail is a real-time conversation with another user. The `talk` command requests a conversational link to another user by specifying that user's logonid:

```
talk < logonid >< hostname >
```

You will see a message "Waiting for your party to respond" as the talk daemon waits for the message recipient to enter the "talk < yourlogonid >< yourhostname >" confirmation string. Once a connection is established, a horizontally split window is created which separates your editing space (the top section) from the recipient's editing space. What you type is displayed simultaneously on the top of your talk window, and on the bottom of the recipient's window. To end the conversation, indicate to the recipient that you are leaving, then enter the interrupt sequence `< ctrl - c >` to break the connection.

12.4 Remote login

Once you have logged in to the system, you can log in to other machines on the network or on other networks with the **telnet** and **rlogin** commands.

telnet

The format for the **telnet** command is:

```
telnet < hostname >
```

After connecting with the remote host, you will be prompted for your login id and your password just as if you were logging on via a terminal on that system.

rlogin

The command **rlogin** has the advantage over **telnet** in that you can specify a username on the command line, and can set it up so that when you rlogin to a host you will not be prompted for a password. Rlogin assumes you wish to login to an account with the same name as the account you are presently logged in as. If you wish to change this, use the **-l** option.

To have a remote system not ask for your password, edit your `.rhosts` file and put a line similar to the following in:

```
remotehostname userid
```

Where remotehostname is the name of the system you will be rlogin from, and userid is the name of the account you will be on.

The format for the rlogin command is:

```
rlogin < hostname >
```

To temporarily suspend the connection to the remote host is done by entering the sequence:

```
~< ctrl - z >
```

This returns you to the local host. To bring the remote host's job back into the foreground, use the **fg** command as described earlier.

12.5 File Transfer Program (ftp)

The letters 'ftp' represent both the name of the program, File Transport Program, and the ARPANET File Transfer Protocol. The command **ftp** is used to

access remote systems for both uploads and downloads of files. The format for the command is:

ftp < *systemname* >

The name of the host is either a name of an Internet node, such as 'zurich.ai.mit.edu', or an address of the form '26.2.0.74'. After issuing the **ftp** command with either form, you will receive the message 'Connected to < *systemname* >' followed by a request for a username and password. If you have an account on that system, you enter your logonid and password and you will have access to your remote account; if you don't have an account, type 'ftp' or 'anonymous' in response to the 'username' prompt to indicate that you wish to access the remote system anonymously. In response to the 'password' prompt, enter either your logonid, or your full E-mail address (< *logonid* >gandalf.umcs.maine.edu), depending on what the remote system requires. After a successful login, you will have the 'ftp>' prompt, and be able to enter commands to move around the remote system.

FTP commands

FTP commands differ slightly from system to system. Usually, if you enter 'help' or the '?' character, a list of commands effective on the remote system will be displayed. Useful commands available on most systems include:

cd < <i>dirname</i> >	Change directory.
lcd < <i>dirname</i> >	Change the <i>local</i> directory, where files will be downloaded.
ls	List the files in the current directory.
pwd	Print working directory on remote machine.
binary	Switch to binary mode.
ascii	Switch to ascii mode.
get < <i>fname</i> >	Copy a file from the remote computer to yours. If you are FTPing from a PC, < <i>fname</i> > may include a drive letter, e.g. a:mystuff.doc
mget < <i>fname1fname2</i> >	Copy multiple files from the remote computer to yours.
bye or quit	Logs out at remote host.

Note that in order to download binary files, you must explicitly type 'binary' at the ftp prompt or the binary file will be corrupted. Ascii mode is the default mode, and should only be used for sending and receiving text files. Downloaded files are placed in the directory that was current when **ftp** was called, unless the **lcd** command was used to change the local directory.

Files in remote sites are often compressed by one of many compression schemes. Files intended for use on UNIX systems are most commonly archived

and compressed using **tar** and **compress**. Remember that compressed files are binary files, and must be transferred using *binary* mode.

You are free to use **ftp** to download files from remote networks, but keep in mind that the downloading of large files (or sometimes of *any* files, depending on the site) should be done during the off-hours to reduce ftp traffic. Off-hours are often defined as between 1900-0600 hours local time, but the peak hours may vary from site to site. Specific site usage rules are detailed in messages that are presented after you log in. The local time is usually displayed right after the ftp policy message, so you don't have to figure out the local time in Denmark or elsewhere. Please pay attention to any messages concerning ftp traffic that the remote site sends to you. Please be considerate of the networks that allow you anonymous ftp access; anonymous ftp is a privilege, not a right. Systems that are overloaded due to inconsiderate people clogging the networks during prime-time hours often disallow anonymous access to free up the system for the purposes that they were originally intended for.

12.6 Network News

Netnews is a world-wide electronic bulletin board. There are newsgroups for almost every subject you can think of (and many more that you can't, or wouldn't want to think of!). To access netnews, you can use one of two programs, **rn** or **xrn**. The first is a command-line version which is started by entering **rn** (for 'readnews'). The second is **xrn**, an X-window's news reader that has many more enhancements over the command-line version. The **xrn** version can be invoked by either typing the command **xrn**, or by selecting the **XRn** option in the 'Tools' window (which comes up when you press the left mouse button away from any X-terminal windows). The first difference you will notice is that the command-line version comes up very quickly; the X-window's version takes its time. The X-window's version has all of the controls up front, in a collection of labeled buttons, while the command-line version relies upon character commands that are listed within a context-sensitive help menu that is available by entering 'h'. The best way to determine which version you will want to use, and the best way to learn how to use the bulletin boards, is to spend time using them.

In addition to reading the news, you are able to post replies to actually get involved in the discussions yourself. Issue `man rn` or `man xrn` for more information on netnews and posting to newsgroups. When you first use netnews you will be subscribed to a group called `news.announce.newusers`. Please read the articles regarding network policy before you post articles. What you send from these systems will be seen by hundreds of thousands of people worldwide and will help others form opinions of us here.

13 Compilers

Here is a partial listing of compilers available on the Computer Science Department UNIX systems:

cc	SUN C compiler
gcc	Gnu C compiler
g++	Gnu C++ compiler
pc	SUN Pascal compiler
f77	SUN Fortran 77 compiler
lisp-3-0	SUN Common Lisp interpreter

The path to the directory that contains the compilers and other executables should be in a path statement in your *.cshrc* file. The path statement allows you to invoke the compiler and other programs from your home directory simply by typing the name of the executable, the C-shell does the work of finding the binary file to execute. A typical path statement follows:

```
set path=(. /usr/local/bin /usr/bin/X11 /bin /usr/ucb /usr/bin /usr/lang)
```

Any other compilers on the system that are not listed above can be found in */user/lang* or */usr/local/bin*.

Disclaimer:

All compilers, and in fact most programs on the system are distributed on an as-is basis. There are no guarantees that the software will perform as described in the documentation.