ATSC 212

UNIX

An operating system (OS) is a resource manager. It's task is to schedule resources and make available resources to system processes or programs. There have been hundreds of operating systems implemented for computers since the 1800's.

UNIX is an operating system written around 1970 by Dennis Ritchie and Kenneth Thompson at AT&T. In 1973, UNIX was re-written into C code making it the first portable OS in the world.

Linux is an operating system that was initially created as a hobby by Linus Torvalds at the University of Helsinki in Finland. Linux conforms to the basic UNIX standard and contains many of the applications available on UNIX systems. Version 1.0 of Linux was released in 1994 under the GNU General Public License and its source code is freely available to everyone.

ATSC 212 – UNIX

Most people think an OS includes the graphical user interface (GUI) and built-in applications (such as web browsers). Operating systems are often packaged with these features to make them more accessible and user friendly.

In truth, an operating system is just the kernel (resource manager) and drivers (code that help the kernel understand system resources). Basic operating systems run on the command line. In other words, there are no fancy interfaces, no special applications, just a prompt that allows the user to input commands directly to the system kernel.

UNIX is a simple, modular and very portable operating system, and in its pure form, runs on the command line. However, to make it more user friendly, UNIX also has many GUIs and applications often packaged with it to make it appear more like the well known Windows systems. However, to take advantage of the power of UNIX, you have to work at the command line.

3

ATSC 212 – UNIX

To access the command line, we need a shell. A shell is an operating environment that allows us to interact with the UNIX kernel. On a UNIX system without a GUI, you would work entirely from a shell. Nowadays, most versions of UNIX come with a GUI similar to the Windows desktop (ie KDE or GNOME). If you are working in a GUI, you need to invoke a shell to get command line access. This can be done by running terminal, xterm, XDarwin, or another shell program. This will automatically create a shell with your default environment to work in.

Once you have a shell you can issue commands to the kernel to do things. Typically, commands take the form

```
<command> [options]  [arguments]
```

Options are special flags or selections you can make to alter the way a command is executed. Arguments are the things the command will execute on.

4

ATSC 212 – UNIX

MAN

To get help from UNIX, use the man command. man is short for manual. Manual pages are help pages for the commands UNIX recognizes. To use man, type man [command]. You can even "man man" to get help on using man.

Manual pages (often referred to as man pages) can be browsed using the arrow keys, and quit out of using q.

You can search man pages for specific keywords using the -k option followed by keywords. This will return a list of man pages containing the keywords.

- man ls
- man -k files

5

ATSC 212 – UNIX

PASSWD

One of the most important responsibilities of a user is to keep their account secure. This begins with having a secure password. To change your password, you simply type passwd at the prompt.

passwd may restrict the types of passwords you can have on certain systems (ie must be longer than 8 characters, cannot be a dictionary word, etc). Try to pick passwords you can remember, so that you don't have to write them down. Also try to use numbers in your passwords. Passwords are case sensitive, so including capitals in odd places can also be effective. But most of all, change your password often!

- passwd

6

ATSC 212 – UNIX

UNIX FILESYSTEM

In any operating system, we need a place to store data, programs, even the operating system itself, this is called the filesystem. The UNIX filesystem is a hierarchical structure that allows users to store information by name. At the top of the hierarchy is the root directory, which always has the name /.

The location of a file in the file system is called its path. Since the root directory is at the top of the hierarchy, all paths start from /. The '/' character is also used to indicate directories.

In UNIX, everything in the filesystem is a file, however, there are three kinds of files. Basic files are used to contain data. They could be a report, or an image, or even a program. Directories are files that are used to describe the hierarchy of the filesystem. Finally, special files are used to identify filesystem devices, the actual hardware that stores the data. You don't need to know about special files, and probably won't ever have access to them.

7

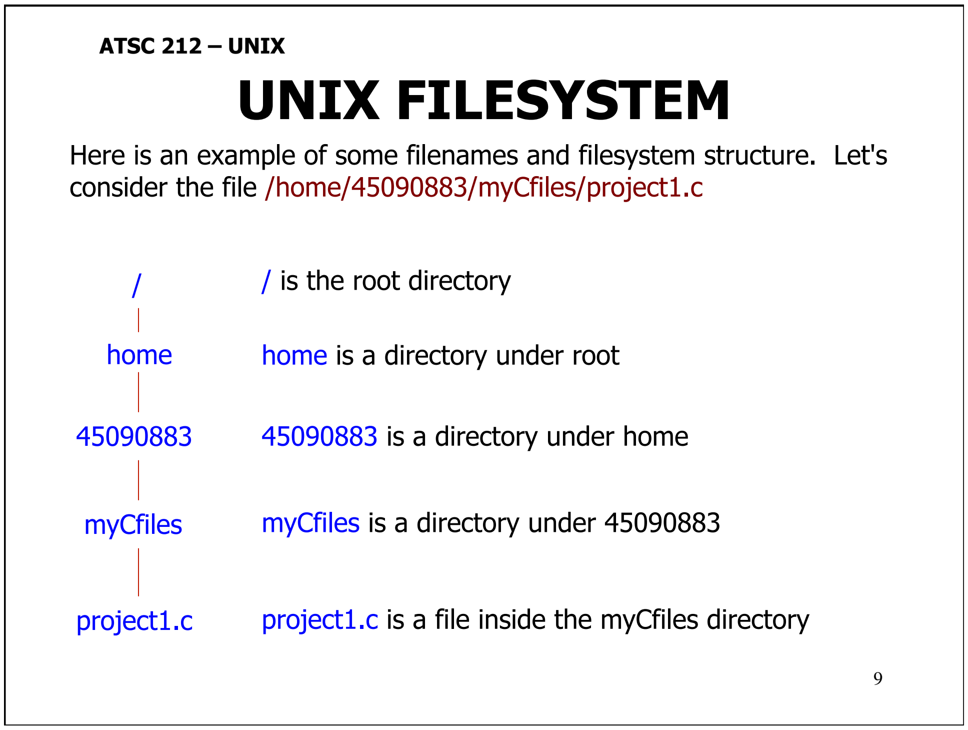
ATSC 212 – UNIX

UNIX FILESYSTEM

The name given to a file or directory can contain almost any character that can be typed on the keyboard and be of almost any length. It is common to use only letters and digits for the first character of a filename, and use only letters, digits, periods . , hyphens - , and underscores _ for the remainder of the filename. However, quotes ‵ ″ , spaces, question marks ? , asterisks * , slashes / \ , and greater than or less than signs < > should not be used in filenames as they all have special meaning to UNIX.

UNIX is also case sensitive. For example, the files ab, aB, Ab, and AB represent four different files in UNIX. Lastly, files that begin with a period are automatically hidden by the filesystem. These files are usually intended to be secure configuration files only used by the operating system or special programs.

8



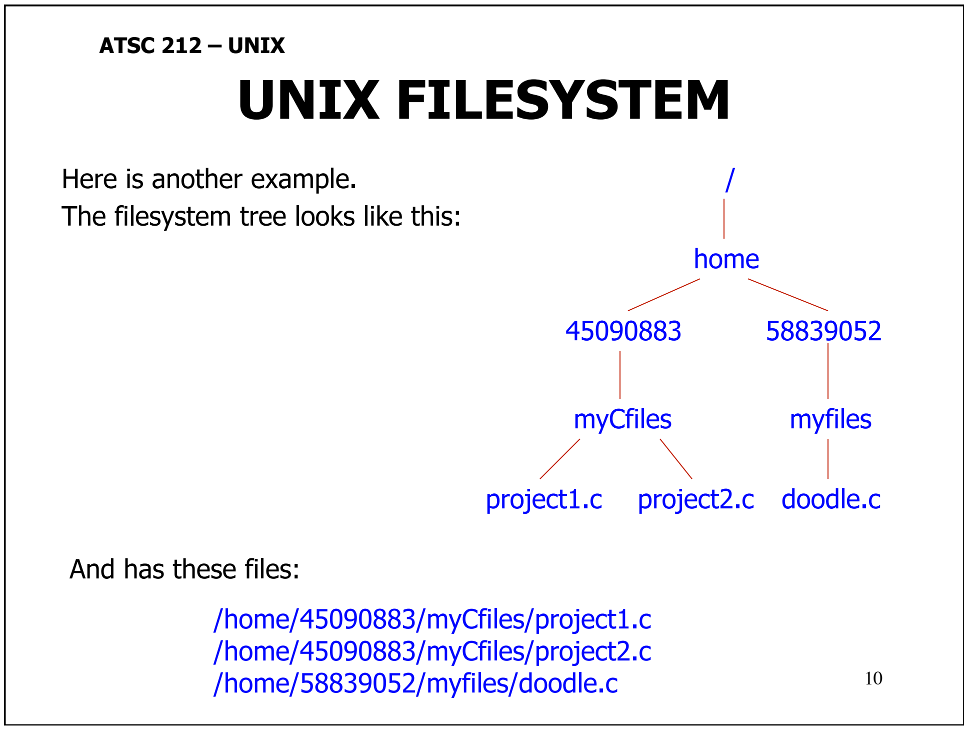
ATSC 212 – UNIX

UNIX FILESYSTEM

Here is an example of some filenames and filesystem structure. Let's consider the file /home/45090883/myCfiles/project1.c

/	/ is the root directory
home	home is a directory under root
45090883	45090883 is a directory under home
myCfiles	myCfiles is a directory under 45090883
project1.c	project1.c is a file inside the myCfiles directory

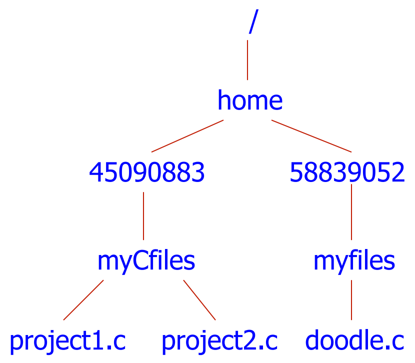
9



ATSC 212 – UNIX

UNIX FILESYSTEM

Here is another example.
The filesystem tree looks like this:



And has these files:

/home/45090883/myCfiles/project1.c
/home/45090883/myCfiles/project2.c
/home/58839052/myfiles/doodle.c

10

UNIX FILESYSTEM

There are special shorthand characters for files which include: / the root directory, ~ your home directory, . the current directory, .. the parent directory of the current directory. These special shorthands can be used with UNIX commands to identify files or directories in particular locations without specifying the entire path.

UNIX also uses special characters called wildcards to help identify files, or groups of files with particular names. * will match any set of characters (ie /* will match all the files in the root directory). ? will match any single character (ie 2?.txt would match 2a.txt, 2b.txt, 2c.txt...29.txt, etc) Square brackets [] can also be used to match characters between the brackets. This can even be specified as a range of numbers or letters (ie [x-z] will match x, y, or z; 2[a-c].txt would match 2a.txt, 2b.txt, or 2c.txt; 2[abc].txt will match the same files.)

UNIX FILESYSTEM

In Windows and OSX, files are automatically given an extension (ie .doc, .jpg) by the application that is writing the file. In UNIX, files are not given extensions automatically although it is common practice for the user to include them in filenames to identify the type of file.

.doc	a document file (typically Word document)
.txt	a text file (typically ASCII plain text)
.exe	an executable binary file
.pl	a perl script
.pdf	a Portable Document Format file
.gif	a GIF format image
.jpg	a JPEG format image

ATSC 212 – UNIX

COMMANDS

Now that we understand some basics about the filesystem, it is time to look at commands. Most commands we use in UNIX are going to affect the filesystem in some way. Practically speaking, everything we do with a computer is about controlling the flow of data. The commands we will be covering are:

ls	cat
pwd	cd
mkdir	rmdir
cp	rm
mv	ln
chown	chmod

grep

13

ATSC 212 – UNIX

LS

The ls command (short for list) can be used to see what files are in a given location in the filesystem. Some common options for ls include –l to get file permissions and ownerships, -h to get human readable filesizes, and -a to show all files in a directory including hidden files. If you supply a directory as an argument, it will list the contents of the directory. If you supply a file, it will list only the file. Wildcards can be used in filenames to get more comprehensive or specific lists.

- ls
- ls -lh
- ls -a /home/45090883/myCfiles
- ls /home/45090883/myCiles

14

ATSC 212 – UNIX

CAT

The cat <file> command (short for concatenate) can be used to see the contents of files. The command prints the contents of the file on the screen. These contents will only be readable if the file is in ASCII format. Binary files will output machine code which will appear as a bunch of gobbledy-gunk.

```
➢cat project1.c

/* project1.c */
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[])
{
...
}
```

15

ATSC 212 – UNIX

PWD

The pwd command (short for print working directory) can be used to show what directory you are in.

```
➢pwd
/home/45090883/myCfiles
```

16

ATSC 212 – UNIX

CD

To change to a different directory, use the cd command (short for change directory. This command allows you to navigate filesystems. You can only cd into a directory you have execute permission on (more on permissions in a moment).

```
➢cd /home
➢cd 45090883
➢cd myCfiles
➢pwd
/home/45090883/myCfiles
```

17

ATSC 212 – UNIX

MKDIR

To make a new directory, use the mkdir <directory name> command (short for make directory. If you do not specify the full path of the directory you wish to make, it will make the directory within the current directory. You must have write permission on the location where you wish to make the directory, or mkdir will not work.

```
➢mkdir /home/45090883/newProject
➢cd /home/45090883/newProject
➢mkdir files
➢ls
files
```

18

RMDIR

To remove an existing directory, use the rmdir <directory name> command (short for remove directory). If the directory is not empty, rmdir (usually) will not let you remove the directory.

```
➢cd /home/45090883
➢rmdir newProject
Cannot remove newProject, directory is not empty
➢cd newProject
➢rmdir files
➢cd /home/45090883
➢rmdir newProject
```

CP

To copy files, use the cp <original file> <new file> command (short for copy). If you attempt to copy a file to a filename that already exists, you will be prompted about whether or not you wish to overwrite the existing file. If you choose yes, the existing file will be destroyed. Wildcards can be used in filenames to copy multiple files at once.

```
➢cd /home/45090883/myCfiles
➢ls
project1.c
➢cp project1.c project1.c.bak
➢ls
project1.c project1.c.bak
```

ATSC 212 – UNIX

RM

To remove files, use the rm <filename> command (short for remove). It is important to understand that files that are removed with rm cannot be retrieved (unless they have been backed up). Also, rm will remove all files listed after it on the command line (ie rm report.txt myreport.txt will remove both report.txt and myreport.txt).

Normally, rm does not prompt the user to ensure that the file should be removed before removing it. Some systems do set this up as a default. However, if you want to be prompted to ensure you are removing the correct files, you can use the -i option. If you do not want to be prompted, even on systems where it is the default, you can use the -f option. If you want to remove all the files within a directory tree (subdirectories and all) you can use the -r option. This will recursively go through all the subdirectories removing files. This can be particularly dangerous.

21

ATSC 212 – UNIX

RM

Wildcards can be used in filenames to remove multiple files at once, however, this is often very dangerous and should probably be used with the -i option.

Some examples:

```
➢cd /home/45090883/myCfiles
➢ls
project1.c  project1.c.bak  project2.c  project3.c  readme.txt
➢rm -i project1.c
rm: remove regular file `project1.c'? y
➢ls
project1.c.bak  project2.c  project3.c  readme.txt
➢rm -f project*.c
➢ls
project1.c.bak  readme.txt
```

22

ATSC 212 – UNIX

MV

To move or rename files, use the mv <original file> <new file> command (short for move). If you attempt to move a file to a filename that already exists, you will be prompted about whether or not you wish to overwrite the existing file. If you choose yes, the existing file will be destroyed. It is important to keep in mind that unlike cp, mv will not retain the original file in the original location

```
➢cd /home/45090883/myCfiles
➢ls
project1.c
➢mv project1.c ../project2.c
➢ls ../
myCfiles  project2.c
```

23

ATSC 212 – UNIX

LN

One important thing we can do on a UNIX filesystem is create links to files. Links are good for allowing files on one filesystem to be accessed on another, and for allowing multiple applications to access the same data at the same time. There are two kinds of links, hard and soft. Both are created with the ln <original file> <link> command (ln short for link). Hard links are created by default. Soft links are created using the -s option.

Hard links are direct pointers to file data. As such, hard links cannot be made across different filesystems. However, they allow two different files to access the same data (this can be dangerous if two processes are writing to the file at the same time). They also allow access to the data even if the link or original file is deleted.

Soft links are aliases for filenames. As such, they can point across filesystems. However, they do not allow simultaneous access, and cannot access data for a deleted file.

24

LN

Here is an example:

```
➢cd /home/45090883/myCfiles
➢ln project1.c new.c
➢ls –l
-rwxr-xr-x    new.c
-rwxr-xr-x    project1.c
➢ln -s new.c verynew.c
➢ls –l
-rwxr-xr-x     new.c
-rwxr-xr-x     project1.c
lrwxrwxrwx   verynew.c -> new.c
```

FILE PERMISSIONS

Most filesystems contain information about which users own files and which users can access files. UNIX has three levels of file ownership; owner, group, and other (which is essentially everyone), and three types of permissions; read, write, and execute.

You can see permissions for a file by using the -l flag with ls. This will display ten characters to the left of the file listing. The first character shows what type of file it is (- means a basic file, d means a directory, l means a link). The next nine characters show the permissions in groups of three characters. The first three characters are the permissions for owner, the next three are the permissions for group, and the last three are the permissions for other. Each group of three characters is specifically r or - in the first position, w or - in the second position, and x or - in the third position.

ATSC 212 – UNIX

FILE PERMISSIONS

If r is present, the file is readable by that user/group. If w is present, the file is writeable by that user/group. If x is present, the file is executable by that user/group. If a - is present in any of the positions, then that permission is not given.

Directories need to be executable for users to run commands on them (like ls or cd). Directories normally need to be writeable to allow a user to create, move, rename, or delete files inside that directory. Some examples:

drwxr-x---	would indicate the file was a directory that the owner could read, write, and execute over, and that members of the group could read or execute over. All other users would not have access to the directory.

ATSC 212 – UNIX

FILE PERMISSIONS

---x--x--x	would be a basic file that everyone has execute privileges for, but nobody could read or write.
lrwxrwxrwx	would be a link to a file that everyone could read, write, and execute.

A file's owner is usually the user that created the file. Groups are collections of users. Having a group permission level allows certain users more access to the file while restricting all others.

28

CHOWN

To change the ownership of a file, use the chown <owner:group> <file> command (chown short for change owner). Owner must be a valid user and group must be a valid group on the system. You can change just the owner by leaving out the group and colon, or you can change just the group by leaving out the owner in the command call. Be careful with this command. Once you change the owner to another user, you no longer own the file.

```
➢cd /home/45090883/myCfiles
➢ls –l
-rwxr-xr-x   me    friends   project1.c
➢chown you:us project1.c
➢ls –l
-rwxr-xr-x   you    us         project1.c
```

CHMOD

File permissions can be changed by using the chmod <permission string> <filename> command (chmod short for change mode). There are two ways to give a permission string, characters and octal.

In octal format, the permissions for each level are encoded from binary into an octal number. If a permission exists, it is given a value of 1 and if not, then it is given a value of 0. For example, rwx would be treated as 111 which converts from 111 in binary to 7 in octal. r-x would be 101 which is 5 in octal. --x would be 001 which is 1 in octal. The permission string is then formed by three numbers indicating the permissions for each level (ie 755 would be rwxr-xr-x).

ATSC 212 – UNIX

CHMOD

If this seems somewhat confusing, there is another way using characters that most people find more intuitive. The characters are r, w, x, +, -, u, g, o, and a. r, w, and x stand for the permissions mentioned already. + indicates to add the permission. - indicates to remove the permission. u means apply the change to the owner level. g means apply the change to the group level. o means apply the change to the other level, and a means apply the change to all levels.

For example, u+r would add read permission to the owner. a-x would remove execute permission from everyone. o+w would add write permission for other.

31

ATSC 212 – UNIX

CHMOD

To apply permissions, we make permission sets out of these characters, separated by commas. For example, a+x,o+r,g+w would add execute permission to all levels, read permission to the other level and write permission to the group level. The octal format is a more compact and precise way of specifying permissions, however, the character method is more intuitive.

```
➢cd /home/45090883/myCfiles
➢ls –l
-rwxr-xr-x  me  friends  project1.c
➢chmod a-x,g+w project1.c
➢ls –l
-rw-rw-r--  me  friends  project1.c
```

32

ATSC 212 – UNIX

GREP

In addition to managing files, you will occasionally want to search files for important information. Although you could use cat to look through the contents of a file, it can be tedious to find specific information from a file particularly if the file is quite large. grep is a tool that can pull lines from files that match particular patterns.

Typically, you would invoke grep as follows

```
grep [options] <pattern> <file> [<files>]
```

If you examine the man page for grep you will find that it has a lot of options for refining how it searches for matching lines. The two most useful options are -r (which will search recursively on a given directory) and -c (which will print the number of lines matching the pattern instead of the lines themselves).

33

ATSC 212 – UNIX

GREP

Normally, the output of grep is all lines of the file(s) that match <pattern>. <pattern> can be simple text or a regular expression. Pure regular expressions are beyond the scope of this course, however, you can find out more about them online if you want to use them. We will be touching upon regular expressions when we come to the section on PERL. They are particularly powerful for finding particular lines. Normally, <pattern> will simple text (ie 2006-10-31, Patrick).

34

ATSC 212 – UNIX

GREP

It is usually a good idea to put <pattern> in quotes. If <pattern> contains a space, this is essential, otherwise grep will treat only the part before the first space as <pattern> and the remainder as files. You can have grep work on multiple files by listing them all after <pattern> or by using wildcards. You can also have grep search over all the files in a directory and its subdirectories by using the -r option.

```
➢grep main project1.c
int main(int argc, char* argv[])
/* This is the main part of the program */
    /* All that remains in this function
```

35

ATSC 212 – UNIX

RUNNING PROGRAMS

Any file that has executable permission can be run. What that means though, depends upon the file. If the file is a script, UNIX will invoke a shell and execute the commands of the script. If the file is a binary (ie a compiled program), it will copy your current environment to a new part of memory and run the binary starting from line 1.

When you run a program or script, it is often as easy as typing in the name of the program or script. However, UNIX needs to know where the file is exactly before it can run it. If the file is in your PATH, a special environment variable that contains the locations of all system executables, then you need do nothing more than type out the program name. If the file is not in your PATH, you need to give the location of the file as part of the command.

36

RUNNING PROGRAMS

Here are some examples:

```
➢cd /home/45090883/myCfiles
➢ls –l
-rwxr-xr-x   me   friends   project1
➢project1
project1: command not found
➢/home/45090883/myCfiles/project1
➢./project1
```

The commands we have been talking about are also binary programs that run by simply typing in their name because they are located in a place that is always in your PATH by default.

THE ENVIRONMENT

Every operating system needs information about users to customize their GUI, specify programs they can run, even to set limits on resources. In UNIX, this information is referred to as the environment. Your environment consists of a set of variables specific to you that the kernel knows about. You can see what those variables are, and what they are set to, by using the env command.

Each user has their own specialized environment. Anytime you invoke a shell (such as when you login), the kernel loads your environment for that shell, so it is also possible to have multiple copies of your environment loaded for different shells, and these environments could differ from each other if you change them.

ATSC 212 – UNIX

THE ENVIRONMENT

There are two ways to change your environment. The first is to change the active environment dynamically. This is done by setting or changing environment variables within a shell. In tcsh, you can set a variable in the environment of the current shell using

```
setenv <variable name> <value>
```

In bash, you can set a variable in the environment of the current shell using

```
export <variable name>=<value>
```

It is important to note that bash requires the = in its syntax and that tcsh does not.

39

ATSC 212 – UNIX

THE ENVIRONMENT

The other way to change your environment is to change the file that the kernel uses to load your environment whenever you invoke a shell. For tcsh, this file is called .tcshrc or .cshrc. For bash, it is .bashrc. Because the filenames begin with a period, these files are hidden in your home directory. You need to use the -a flag with ls to see them.

These files are scripts that contain commands to initialize the environment (usually variable setting commands such as export or setenv). They can also include commands to run other scripts such as

```
. <file>
```

or

```
source <file>
```

40

PIPES

You may find yourself in a situation where you need the output of one command as input to another command. Usually when you need to do this, there is no easy way to get the output of the first command into the second. For this reason, UNIX has a special character called a pipe, | (shift \ on the keyboard). Pipes can be used to direct the standard output from one command into the standard input of another. Depending on the commands you combine, you can have spectacularly disastrous results. For that reason, it is usually not a good idea to use pipes with mv, rm, chmod, and chown.

```
➢ls -lrh /home/45090883/myCfiles | grep "rwx------"
-rwx------   me   us   mydata.txt
-rwx------   me   us   project2
```

REDIRECTION

So far whenever you have executed a command or program, the command took data either from the command line, or from a location known to its binary. If the program only knows how to take data from the command line, and you want to give it an entire file, it would be tedious to type it all out on the command line. Another way to do it would be to redirect the contents of the file through standard input. You do this using the < character.

```
➢cd /home/45090883/myCfiles
➢ls
mydata.txt  project1
➢./project1 < mydata.txt
```

ATSC 212 – UNIX

REDIRECTION

We can also redirect the standard output of a command/program to a file so that we can retain it. There are two ways to redirect the standard output to a file, > and >>. > overwrites the redirected file. >> appends to the redirected file. Both versions will create the redirected file if it does not exist. It is also worth noting that you can use both input and output redirection at the same time.

```
➢cd /home/45090883/myCfiles
➢ls
mydata.txt  project1
➢./project1 > output.txt
➢./project1 < mydata.txt >> output.txt
```

43

ATSC 212 – UNIX

VI

Now that you understand basic commands and are ready to start writing scripts and code, you need to know how to edit files. There are two common ASCII text editors in UNIX, VI and emacs.

VI is a line editor, but it uses the full terminal screen, so you can view more of the file than just the current line. VI can create, edit, and save files in ASCII text format. Most VI installations also recognize different files extensions (ie .c, .php, .pl) and will colour code keywords and syntax appropriate to the type of file. To start VI, type vi <filename>.

VI has two modes of operation, command mode and edit mode. In edit mode, you change the text in the file. In command mode, you can utilize commands to change the contents of the file (such as copy and paste), save the file, or even create macros.

44

ATSC 212 – UNIX

VI

When you first open a file with VI, you will be in command mode. You can return to command mode from edit mode by hitting esc. To execute a command in command mode, type

[count] <command> [where]

[count] is an optional number that tells VI how many times to execute the command. If you do not supply a [count] then VI assumes it should only do the command once. <command> is the command name. Most commands are single letters in VI. [where] is additional optional parameters that only apply to certain commands. Common commands include:

a	enter edit mode and insert text after the current position. [where] does not apply to this command. [count] specifies how many times inputted characters should be repeated.

45

ATSC 212 – UNIX

VI

i	enter edit mode. [where] does not apply to this command. [count] specifies how many times inputted characters should be repeated. This is similar to the a command but it inserts text before the cursor position.
h	move the cursor to the left.
j	move the cursor down one line.
k	move the cursor up one line.
l	move the cursor to the right.
u	undo the last change to the file. Re-issuing the command will re-do the change.
G	goto the line number specified by [count].
x	delete a character.
dd	delete a line.
d^	delete from the current position to the beginning of the line.

46

ATSC 212 – UNIX

VI

dw	deletes the current word.
yy	copies the current line. [count] will copy that many lines.
y^	copies from the cursor to the beginning of the line.
y$	copies from the cursor to the end of the line.
yw	copies the current word.
p	pastes deleted or copied text after the cursor. [count] increases the number of copies pasted.
P	pastes deleted or copied text before the cursor. [count] increases the number of copies pasted.
ZZ	save the current file and quit VI.
:q!	quit VI without saving the current file.
:w	save the current file without quitting.
:wq	save the current file and quit.

In edit mode, you can move about the file using the arrow keys, and can insert text by typing it. In most versions of VI, you can delete text by using the del and/or backspace keys. Anything typed in this mode ends up being inserted into the file.

47

ATSC 212 – UNIX

EMACS

The other commonly used editor for UNIX is emacs. Emacs is not as widely supported as vi, and may not be installed on all UNIX systems. To start emacs, type emacs. By default, emacs runs in an X-windows graphical mode which looks somewhat like MS Word. If you want to run emacs within your current terminal window, use the -nw option when invoking emacs.

Emacs functions very similarly to editors like Word, utilizing keyboard controls and mouse to cut and paste text. It also has pull-down menus with standard cut, paste, file creation, save, and quit features. It can also be set by the options menu to colour code text depending on file type much in the same way vi does.

48

ATSC 212 – UNIX

REMOTE COMPUTING

Most of the work you do on UNIX systems will be done remotely. You can use your Windows or Mac machines to login to UNIX systems and do work on them as if you were sitting at those machines. Most research systems are large scale computers housed in special facilities and are never interacted with physically except by systems administrators.

In this course, you will be remotely utilizing a UNIX server called eidolon using PuTTY and/or NX. Both PuTTY and NX provide a basic interface that will allow you to login to the UNIX server securely. This is commonly available freeware for Windows although there are other commercial products that do the same thing (such as SecureCRT). PuTTY gives a terminal interface to a computer while NX provides a virtual desktop, allowing you to interact with eidolon as if you were sitting at it.

49

ATSC 212 – UNIX

REMOTE COMPUTING

There are many different ways to connect to other computers and most depend upon the OS you are using. Within UNIX, there are two common ways to connect to other machines through a terminal, telnet and ssh.

Telnet creates a simple, unencrypted channel to another computer. To do this, type telnet <address>. <address> should be the name or IP of the machine you wish to access (ie eidolon.eos.ubc.ca or 137.82.23.188).

Although telnet is usually installed on most systems, it is typically blocked or disabled by systems administrators because of its lack of security. All information, including user passwords, is transferred in plain text over the network. Telnet should not be used except in protected, private networks.

50

ATSC 212 – UNIX

REMOTE COMPUTING

The secure way to access a system is to use secure shell, ssh. To do this, type ssh [user@]<address>. <address> is a valid name or IP, same as telnet. [user@] is optional and specifies a valid user on that system that you wish to connect as. If you do not supply [user@], ssh will assume you want to login as your current user.

SSH connections are encrypted using public-key encryption methods similar to PGP (Pretty Good Privacy). Thus all communication made between computers is encrypted before it is sent over the network. The encryption is very difficult to break which gives a great deal of security to valid users (particularly in protecting passwords and private data).

SSH can also be configured to deny certain users and networks from accessing machines, making it more difficult for hackers to access the computer. For example, the server we will use, eidolon, is not accessible from networks outside of UBCNet.

51

ATSC 212 – UNIX

FILE TRANSFERS

Besides logging into other systems, sometimes you will want to transfer files between systems. As with remote computing, the tools available vary considerably by OS. However, the manner in which files are transferred, regardless of system, is almost always by file-transfer-protocol (FTP).

In UNIX, there are two utilities that allow file transfers, ftp and sftp. sftp is the secure (encrypted) version of ftp. To use these tools, type (s)ftp [user@]<address>. In ftp, if you do not specify the [user@], then it assumes you want to perform anonymous ftp. if the system you are ftp'ing to accepts anonymous logins, you will be asked to input a password which should be your email address. In sftp, if [user@] is not specified then it is assumed to be your current user. In both cases, the address must point to a computer that runs an ftp server or accepts ftp logins, otherwise (s)ftp will fail.

52

ATSC 212 – UNIX

FILE TRANSFERS

Once a connection is established, you can use the ls, pwd, and cd commands to move about the remote filesystem. lls, lpwd, and lcd can be used to perform the same actions on your local computer (although it is usually a good idea to enter the local directory you want to move things to or from before invoking (s)ftp).

To download a file from the remote system to the local one, use get <filename>. To upload a file from the local system to the remote one, use put <filename>. You will need the appropriate write permissions to upload or download files. These will be based upon the user you logged in as, or in the case of anonymous login, the server configuration. To end a (s)ftp session, simply type quit.

53

ATSC 212 – UNIX

AND THAT IS IT…

Well, not really. Refer to your quickstudy guides for more information on all the topics covered. *The UNIX Programming Environment* is also good for expanding on the topics covered and giving more advanced information.

You can also utilize the man pages and the web. www.google.ca and www.wikipedia.com will lead you to links and information on all the topics covered.

54