

A Brief Introduction to UNIX

Lindsay Kubasik, Geoffrey Lawler, Andrew Hilton

Version 1.0

Duke University Computer Science, January 2012

Contents

1	Fundamentals of UNIX	2
1.1	What is UNIX?	2
1.2	Why UNIX?	2
2	Accessing a UNIX Shell	2
2.1	From Mac OS X	2
2.2	From Windows	3
2.3	Connecting to the Duke Linux Cluster	3
2.4	Logging Out	8
3	Directory Structure	8
3.1	The UNIX Directory Structure	8
3.2	Navigating UNIX Directories	9
4	UNIX Commands	10
4.1	Control Symbols	10
4.2	Commands for Directory Navigation	11
4.3	Copying, Moving, and Deleting	12
4.4	Secure Copy	13
4.5	Viewing, Comparing, and Searching Files	14
4.6	Flags	15
5	Coding and Compiling	15
5.1	Text Editors	15
5.2	Compiling	16
5.3	Debugging	16
6	Other Operations	17
6.1	clear	17
6.2	Subversioning	17
6.3	Scripting	17

1 Fundamentals of UNIX

1.1 What is UNIX?

UNIX is an operating system originally developed in the late 1960s at AT&T's Bell Labs. Today, many common operating systems are based on UNIX including Mac OS X, Solaris, all variants of Linux, Android, and iOS.

User interaction with any UNIX system is facilitated by either a shell or, more recently, a full graphical user interface (GUI) such as that found on Windows or Mac OS X. A shell is a simple command line prompt that allows a user to interact with the computer and run programs. UNIX was built to support multiple simultaneous users, so a single UNIX workstation can be accessed by multiple local or remote users at any given time.

Using the UNIX shell is not very different in concept than using any other operating system, but rather than having a visual interface, you use commands typed into the prompt to navigate or run programs. Similar to the file system of your operating system, you can organize files by making new folders, or directories, navigating between different directories, and seeing the contents of these directories. You can also use various programs to open files that exist in any of these directories, edit them, and save them.

1.2 Why UNIX?

There are also merits to UNIX that make it particularly useful. For the purposes of this class, it is important to use UNIX because it allows remote access to another computer. This means that using your own laptop you can connect to a computer in the Duke Computer Science Linux Cluster in the LSRC. Because your files will be stored remotely, you will be able to access them on any other computer as well by remotely connecting using your CS login. In addition to having access to your files, you will be able to compile and run your files on the CS Linux cluster rather than trying to compile and run your files on your own computer. You will also be able to turn in your projects directly using UNIX, and because they are all being compiled on the same machines with the same compilers, any discrepancies resulting from different compilers can be avoided.

Beyond all of the class-specific reasons to use UNIX, it is important for any computer science major to understand how to navigate and operate using the command line. Most "real world" server environments are based on some flavor of UNIX, and knowing how to work in/with them is an essential job skill.

2 Accessing a UNIX Shell

2.1 From Mac OS X

Mac OS X is based on UNIX, meaning all commands described will work within an OS X shell. Mac provides a shell application called Terminal which can be found in /Applications/Utilities. The easiest way to open this is to click on the spotlight magnifying glass in the right hand corner of your screen and type Terminal and then enter. Terminal is also available by opening Finder, looking in your Applications folder, opening the Utilities

sub-folder, and then clicking on Terminal. This shell is capable of executing commands on the host machine or connecting to a remote server such as via SSH as described later.

2.2 From Windows

Windows is not based on UNIX and the MS-DOS command prompt cannot execute UNIX commands. In order to remotely access a UNIX machine Windows users must download a terminal emulator such as PuTTY. PuTTY is available from the OIT software download website,

<http://oit.duke.edu/comp-print/software/>

where you can then click on the "Browse and order software" link. After downloading, installing, and running PuTTY, you will be brought to a PuTTY Configuration screen. From here, you will be able to connect to a remote server. In addition to PuTTY, you will also need X-Win32 (also available through OIT) to allow you to have external graphical windows pop up. X-Win32 will allow you to use programs outside of the command line prompt while connected remotely. There are a few options when you login to OIT, and "X-Win 32 2012, build 67 with SSH" is the one you should use. There is also a PDF file that is a part of the two files you can download from OIT when installing the software called Activation Instructions (2011 & 2012). The PDF file explains how to get X-Win 2012 licensed. Note: if you are off-campus, you must use the Activation License and not the License Server. Go to the last page of the instructions.

2.3 Connecting to the Duke Linux Cluster

For any of the work done in this class, it is important to understand how to remotely connect to the Duke Linux Cluster. Before doing this, you will need to have a CS login and password. Once you have this information, you will be able to connect by using either PuTTY on Windows or Terminal on Mac OS X.

2.3.1 Windows

You should start by running X-Win32 (in your start menu you want to pick the one called X-Win32 2012). You only need to open X-Win32 and have it running in the background. Now, you should open PuTTY. After you are on the PuTTY Configuration screen, you will need to enter a Host Name. For this class, the host will be

`linux.cs.duke.edu`

The connection type should by default be set to SSH, and the port to 22. Your screen should look like the one in Figure 1 below.

Next, you need to enable X11 forwarding, which allows PuTTY to use X-Win32 effectively. To do this, click the "+" next to SSH on the left panel of the PuTTY configuration screen, then click "X11". Click the box next to "Enable X11 forwarding" (shown below in Figure 2) and then click on "Session" in the menu on the left to get back to the screen with the host.

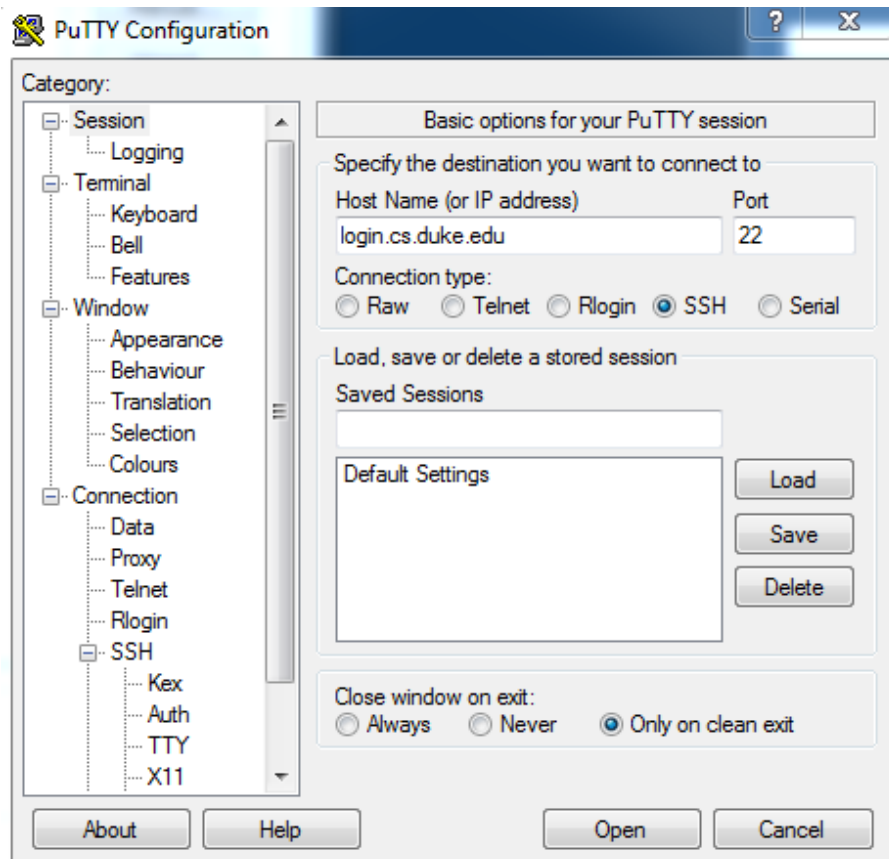


Figure 1: Putty Config Screen

Now, it makes sense to save this session so that you do not need to redo these steps every time. You can give a name to this session in the saved sessions block and hit the save button. In the future, this session will be listed and you can simply click on it and then hit load instead of rewriting the host name and enabling X11 forwarding. Your screen should now look similar to the one in Figure 3 below.

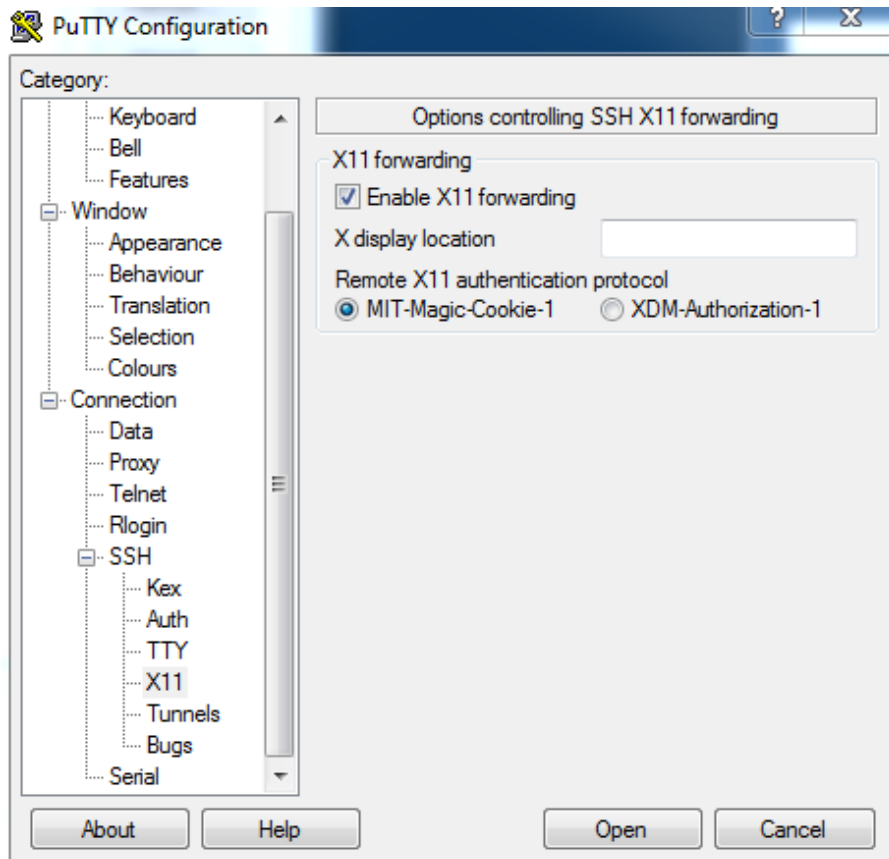


Figure 2: Enabling X11 Forwarding

At this point, you should hit open. You may get a security alert message, and unless you believe there is a security problem, you should click yes to add the host to your list of trusted hosts. Next, you will need to provide your CS login on the screen shown in Figure 4.

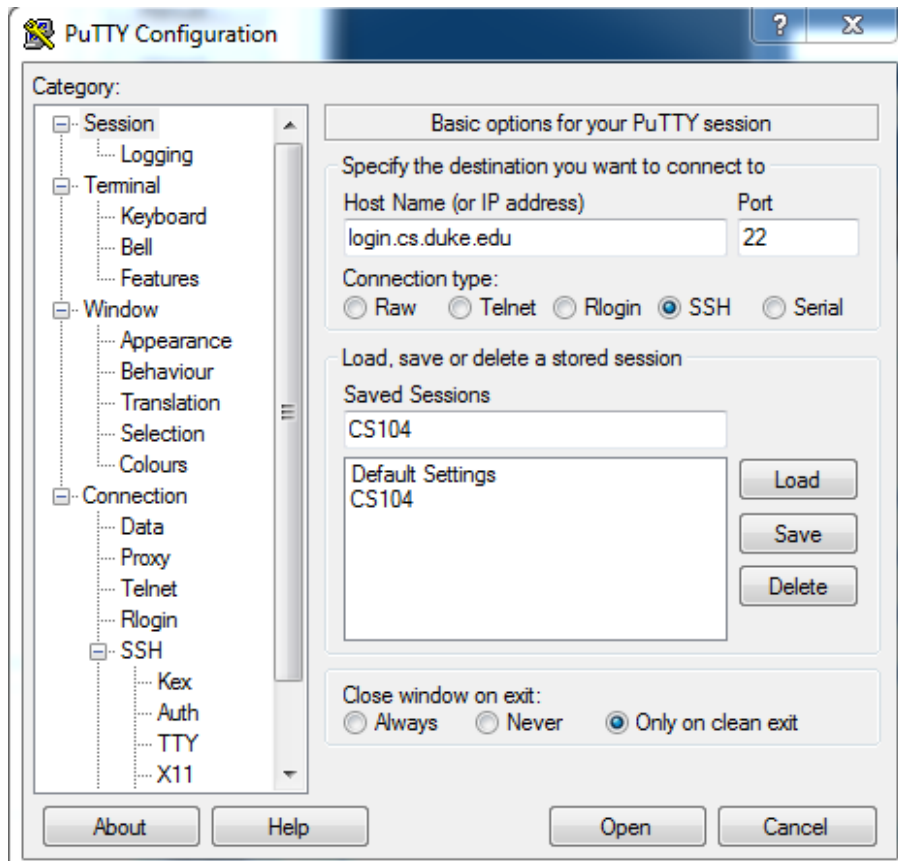


Figure 3: Saved Session

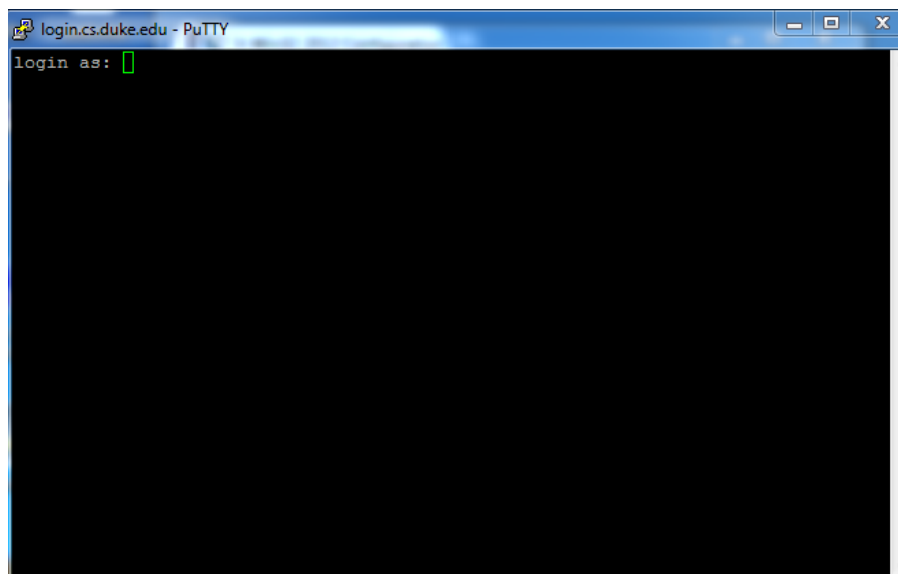


Figure 4: Login Screen

After entering your login, you will need to enter your password. If it connects successfully, you will now be in your CS home directory, and you should have a prompt similar to the one below in Figure 5.

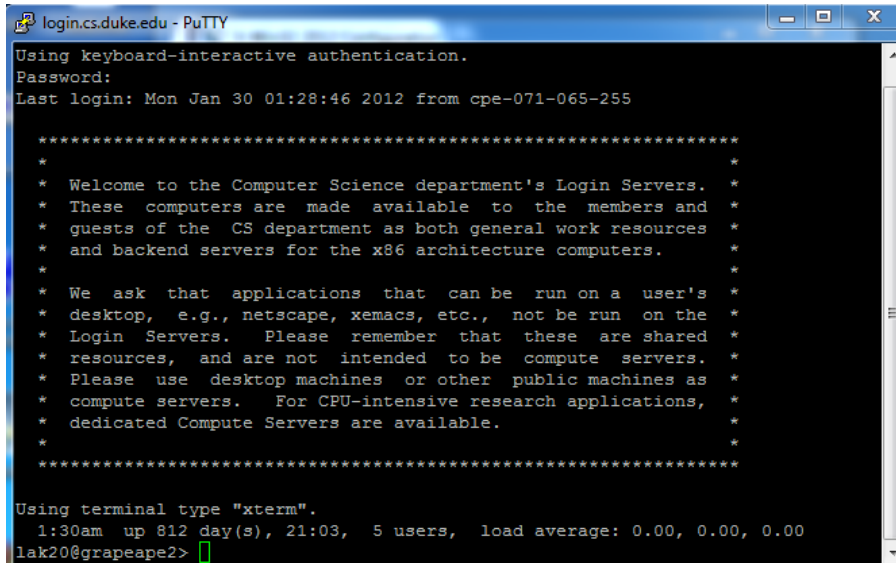


Figure 5: Successful Login

2.3.2 Mac OS X

You will want to start by opening Terminal, as described above. After launching Terminal, you will type the command below (replace CSLOGIN with your username):

```
ssh -XY CSLOGIN@linux.cs.duke.edu
```

This is also shown in Figure 6 below.

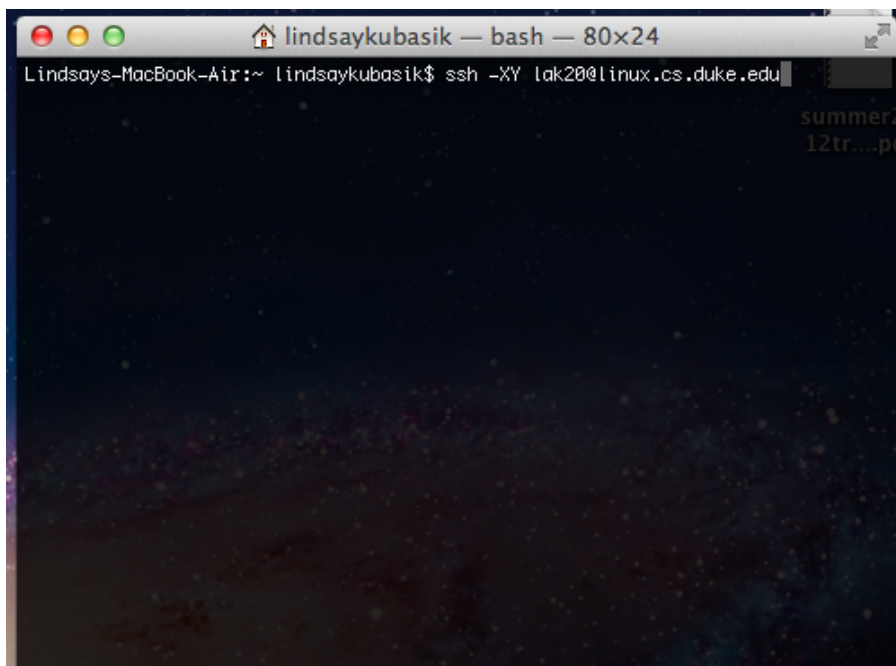
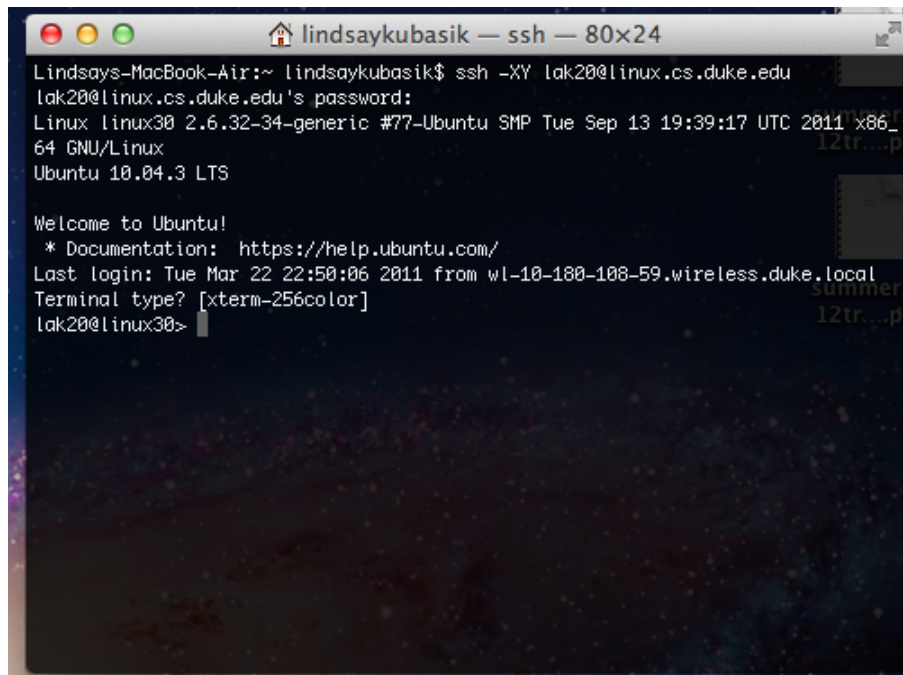


Figure 6: SSH Command in Terminal

The -XY is only necessary if you plan to use any graphical programs during your session. If you plan only to use programs in the command line you can omit this part

of the command. At this point you may be asked to add the connection to your trusted networks by typing yes/no, at which point you should type yes unless you believe there are security problems. After this, you will need to enter your CS account password. If you are then welcomed to Ubuntu, you have successfully connected. When prompted for terminal type, you can simply hit enter. You should now be connected and in your CS home directory, and you should have a prompt with your CS login, similar to the one below in Figure 7.

A screenshot of a terminal window on a Mac. The window title is "lindsaykubasik — ssh — 80x24". The terminal shows the following text:

```
Lindsays-MacBook-Air:~ lindsaykubasik$ ssh -XY lak20@linux.cs.duke.edu
lak20@linux.cs.duke.edu's password:
Linux linux30 2.6.32-34-generic #77-Ubuntu SMP Tue Sep 13 19:39:17 UTC 2011 x86_
64 GNU/Linux
Ubuntu 10.04.3 LTS

Welcome to Ubuntu!
 * Documentation: https://help.ubuntu.com/
Last login: Tue Mar 22 22:50:06 2011 from wl-10-100-108-59.wireless.duke.local
Terminal type? [xterm-256color]
lak20@linux30>
```

Figure 7: Successful Login

2.4 Logging Out

When you are finished using an UNIX machine that you have connected to remotely, you may terminate your connection simply by typing:

```
logout
```

at the command prompt.

3 Directory Structure

3.1 The UNIX Directory Structure

When using a UNIX or UNIX-like system, one of the most important things to understand is the directory structure. Understanding the directory structure is similar to understanding how folders and files on your computer are organized. UNIX filesystems, like most filesystems, are hierarchical, that is the directory structure can be represented

as a tree. The root node of any UNIX filesystem is referred to simply as **root** and is denoted as a slash (/). All directories are direct or indirect subdirectories of root.

Each user in a UNIX system is given their own directory for storage referred to as a home directory. In file paths, a home directory is written `~username` or simply `~` for the current user. UNIX ties a user-based permission scheme to the file system to protect system files and preserve a user's privacy. For example files in a user's home directory usually may only be viewed and edited by that user. System files are often only editable by an administrative user, referred to as the *root user*.

3.2 Navigating UNIX Directories

Being able to navigate the file system of a UNIX machine from the shell is essential for almost any use of a UNIX machine. Directories in UNIX can be accessed by their path, which is the chain of directories separating two locations separated by forward slashes (/) just like on the web. Any file or directory is addressed by its path relative to some location. For example the location of user Bob's source file `code.c` could be referenced from Bob's home directory at

```
~/cps104/project1/src/code.c
```

or from the root of the filesystem

```
/home/home1/cps104/project1/src/code.c
```

A shell will also have a current or "working" directory. You can find what a shell's working directory is at any point by typing **pwd** (print working directory) at the command prompt. In addition to referencing locations from the home directory or root, file paths can be referenced from the current working directory. In fact, if a path does not begin with `~` or `/` then it is assumed that the path begins at the working directory. In the above example if the working directory is

```
/home/home1/cps104/project1
```

then `code.c` may be referred to simply as

```
src/code.c
```

The two most commonly used commands when navigating a UNIX filesystem are **cd** (change directory) and **ls** (list stuff). **cd** is used to change the working directory to the path supplied as an argument. **ls** lists all non-hidden contents in the working directory. These commands are explained in somewhat more detail in their respective sections below.

Every directory in UNIX contains two special directories `."` and `..`, which are not real directories but links to other directories. The single dot `."` refers to the current directory and the double dot `..` refers to the directory one level above the current directory. In the previous example, another way to access `code.c` would be to use

```
./src/code.c
```

as the path. One could change to the `~/bob/cps104/` directory by typing

```
cd ..
```

You may wonder why the `ls` command doesn't display `."` or `.."`. This is because all filenames in UNIX which begin with a dot (`.`) are considered hidden and are excluded from most commands unless explicitly specified.

4 UNIX Commands

4.1 Control Symbols

4.1.1 Control C

Control-C is done by hitting the control key and then `C`—the same as the keyboard shortcut for copy. In Mac OS X this is actually Control-C, not Command-C. In the command line this is not copy, and instead it is an abort command to abort a running process. On Mac OSX, you can still copy with Command-C. On Windows, you typically need to use the menu to copy. Note that if you are in Emacs (described in Section 5.1.1), you can copy with Control-w.

4.1.2 Control Z

Control-Z is done by hitting the control key and then `Z`—the same as the keyboard shortcut for undo. In the command line this is not undo, and instead it suspends the current process. This returns you to the command line, allowing you to execute other commands while leaving your first program “frozen”. You can bring a suspended program back with the `fg` command (short for foreground), or let it run in the background with the `bg` command. The `jobs` command shows you all background and suspended jobs. You can give a job number (from the output of `jobs`) to `fg` or `bg` as a command line argument to affect a particular job.

4.1.3 Input/Output

In UNIX you can use the characters `<` and `>` to redirect input and/or output. One common case for this would be to redirect the output of a command—which would normally show up directly in the terminal—to a text file. This can be done by using the `>` symbol. As an example, typing

```
ls > fileList
```

would put the list of files that would normally appear in the shell into a file named `fileList`. This can be useful when you want to save the output of something running, for example the error list when compiling code or the output of running code. Input redirection is less common, but is useful if you want to run a program or command that takes a file as input. This is done the same as above, but with the `<` symbol instead. Note that you are encouraged to use both of these in homework 1 to help you test your program.

4.2 Commands for Directory Navigation

Directory navigation in UNIX is not much different than your folders on your current operating system, but you navigate using the command line rather than by clicking on visual folders. The file system is discussed in more detail in later sections.

4.2.1 Finding Current Directory

The command for finding your current directory is

```
pwd
```

which prints your current working directory. This is useful when you need to figure out the path of your current location.

4.2.2 List Stuff

The command for listing everything in the directory is

```
ls
```

This command is similar to looking into a folder to see everything that is inside. As discussed in the directory section above, some things in a directory could be hidden, but these can be revealed using flags, which are also discussed below.

4.2.3 Change Directory

The command for changing directories is

```
cd
```

This is similar to clicking on a folder to move into that folder. The change directory command can be used with absolute or relative paths. Different ways to express paths are discussed above, and the general form for using `cd` is

```
cd Path/TargetDirectory
```

In this example, we want to get to `TargetDirectory`. The `Path` could be nothing (as in we do only `cd TargetDirectory`), or it could be a path of folders we need to move through to get to the location of `TargetDirectory`.

4.2.4 Make Directory

The command for making a new directory is

```
mkdir
```

This is similar to clicking "new folder" in a GUI environment. This command can be used to make a new directory by typing

```
mkdir NewDirectoryName
```

This will make a new directory called `NewDirectoryName`.

4.3 Copying, Moving, and Deleting

4.3.1 Copying

The command for copying a file is

```
cp
```

Files can be copied to the same directory, or to another directory. To copy a file to another file you can use

```
cp SourceFile TargetFile
```

where SourceFile is the file you want to copy, and TargetFile is the name of the new file you want it to be copied into. Files can be copied into a different directory using

```
cp SourceFile TargetPath
```

where SourceFile is the file you want to copy, and TargetPath is the path to where you want the file to be copied.

4.3.2 Moving

The command for moving a file is

```
mv
```

The move command moves files or directories from one place to another. As a result, this can be used to rename files as well. To rename a file, you can use

```
mv OldFile NewFile
```

This will rename OldFile to NewFile. You may also want to use this command to move the file to a new location. This can be done using

```
mv OldFile SomePath/OldFile
```

In this case we are moving something called OldFile to a new location, SomePath, and keeping the name as OldFile. After the path we could have changed the name of the file, like below.

```
mv OldFile SomePath/NewFile
```

In this case we are moving something called OldFile to a new location, SomePath, and keeping the name as NewFile.

4.3.3 Removing

The command for removing (or deleting) a file is

```
rm
```

In order to remove a file from your current directory, you can use

```
rm FileName
```

where FileName is the name of the file that you want to remove. Directories can be removed by adding the -r flag to the rm command, such as:

```
rm -r aDirectory
```

Be careful when using the -r flag as directories once removed cannot be recovered.

4.4 Secure Copy

Secure copy is used for transferring files between your local computer and a remote host, or two remote hosts. In this class, you can use this command to transfer files from your personal computer into your CS account space on the CS Linux cluster.

4.4.1 Mac OS X

Once you have terminal open, in order to copy a file from your computer to the remote host, you can use

```
scp SourceFile CSACCOUNT@linux.cs.duke.edu:path/TargetFile
```

where SourceFile is the file you want to copy, path is the path from your CS home directory to where you want the file copied, and TargetFile is what you want the file to be named when it is on the remote host. You can copy files from the remote host to your computer by using

```
scp CSACCOUNT@linux.cs.duke.edu:path/SourceFile TargetFile
```

where SourceFile is the file you want to copy, path is the path from your CS home directory to where the source file is located, and TargetFile is what you want the file to be named when it is on your home computer. The file would be copied to your current location on your home computer when you use this command.

4.4.2 Windows

In order to use secure copy on Windows you will need to open another program. When you download PuTTY from OIT, a program called psftp will also be downloaded and put into the PuTTY folder in your start menu. When you want to use secure copy, you should open this program, which should open a terminal window that says on top that no host is specified. When you see this you should type

```
open linux.cs.duke.edu
```

and then enter your CS account name and password as you do when connecting through PuTTY. Once this connects you will be in you CS directory and you can navigate to wherever you want to copy files to or from. Now you can transfer files from the remote host to your personal computer using

```
get myfile
```

which will fetch myfile from the server and save it on your personal computer. You can also put files from your personal computer onto the server by using

```
put myfile
```

which will put the file myfile from your personal computer onto the server.

4.5 Viewing, Comparing, and Searching Files

4.5.1 less

Less is used to view but not change the contents of a text file within the shell. By typing

```
less file_name
```

you can use less to open the file called file_name and view it in the shell, one screen at a time.

4.5.2 diff

The diff command can be used to compare the contents of two files, as you have seen in homework 1. This command is useful in telling if there have been changes made to a file, or for comparing outputs of a program to see if anything has changed. This command is used by typing

```
diff original_file new_file
```

where the output of the command is the changes that would have to be made to original_file to make it into new_file. If there is no output then the files are the same.

4.5.3 grep

This command is used for searching in the command line. One example of how this can be used is

```
grep dog animals.txt
```

where the command would return any line containing the character sequence "dog". This command can also be used more generically with *, like in the command

```
grep dog *.txt
```

where any file ending in .txt will be searched for the character sequence "dog". This command becomes more powerful with the use of regular expressions, as you can read more about from other resources.

4.5.4 find

Find is used in the command line to search and locate files based on criteria specified by the user, and then applies an action to these files as specified by the user. This could be used to find and copy or find and remove a bunch of files with a certain characteristic. The find command can be used in many different ways, and the wikipedia article

```
http://en.wikipedia.org/wiki/Find
```

has many examples of how to take advantage of the command.

4.6 Flags

Flags can be used in conjunction with various UNIX commands in order to alter or enhance the commands. Flags are inserted in-line with a UNIX command and consist of a "-" followed by a letter. Different letters have different meanings and are relevant to different commands. As an example, in the command

```
rm -v foo
```

the `-v` will cause `rm` to detail any successful removals that happen. In the case of `rm`, `-i` could also be used to require yes/no confirmation before a file is deleted. Each command has different flags that can be useful, they are typically described in the `man` page for the command. Type `man whateverCommand` to get information about any commands flags and arguments.

5 Coding and Compiling

5.1 Text Editors

There are a variety of text editors that you can use when logged in to your CS account. Text editors are what you will use to actually view, edit, and write your code. This is a brief discussion of two popular options.

5.1.1 Emacs

In order to run `emacs` once you are logged into you CS account you can simply type

```
emacs &
```

The default mode `emacs` opens in is a graphical version. The `&` causes `emacs` to open in the background (as if you did `ctrl-Z`, then `bg`), so that you can use your terminal for other commands while `emacs` is open. If you do not have graphical support on your connection (e.g., you did not install/setup X support above), you will need to leave off the `&`, as you will get the terminal version which is not useful in the background. You can also get the terminal version by supplying the `-nw` argument to `emacs` (e.g., if you are on a very slow connection).

When you are first starting, using `emacs` in a graphical version is probably the easiest option. This will feel similar to using any other program when you can use the file menu or other commands at the top of the screen so save and open files. As you frequently use commands, you can learn their keyboard shortcuts from the menus, and transition to solely using the keyboard. Once you are familiar with the keyboard shortcuts, you can use the graphical or terminal version as you prefer. Using the terminal version is typically faster over a remote connection, as there is less data to transfer.

5.1.2 VIM

Another option for text editing is `VIM`. `VIM` is a command line text editor, which means that you will work within the shell instead of in an external window. You can start `VIM` by typing

vim

VIM has many useful features, but it is more complicated than emacs, so if you are interested in this option, you may want to look at a tutorial, such as

http://blog.interlinked.org/tutorials/vim_tutorial.html

5.2 Compiling

For this class, we will be using the gcc compiler. This is available on the CS Linux clusters when you are remotely logged on. When you are ready to try compiling your code, you can try a command such as

```
gcc myprog.c
```

where myprog.c is a c file that you are trying to compile. This is the simplest way to compile your program, but there are many flags that will be useful in trying to actually ensure your code is working properly. For example, compiling as

```
gcc myprog.c -o myexe
```

will create an executable program called myexe that would be run by typing ./myexe into the command line. Because you are running a program and not just giving a path you must explicitly type the ./ when in the same directory as your program. The -o followed by myexe is used to name your executable file myexe, otherwise a default name will be used. Another useful way to compile would be to use

```
gcc -ggdb3 myprog.c -o myexe
```

where the -ggdb3 adds debugging information that can be used by gdb in the case that your code does not run properly (in particular, it adds the maximum debugging information possible). There are many other useful flags in compiling, but these are the most useful for this class.

5.3 Debugging

In this class, debugging will be done using gdb in the command line. This debugger is a very powerful tool, but there is some overhead in learning how to use it. gdb will be covered in lecture, but here is a tutorial with more information and some examples on how you can use gdb to examine your code. Remember that in order to use gdb to debug your code you must include the -ggdb3 flag while compiling.

<http://www.cs.cmu.edu/~gilpin/tutorial/>

6 Other Operations

6.1 clear

Clear is a useful command when your shell gets crowded with lots of text. At any point you can use

```
clear
```

to clear your screen of the shell. This does not delete all your previous commands, but it inserts a large blank space so that you can start with a clean screen.

6.2 Subversioning

This document will not go into the details of subversioning (SVN) but it is a useful tool that allows you to track different iterations of a document. This could be useful as you make changes to your code so that you can revert to previous versions or see what you changed between different versions. Feel free to read more information on this topic if it is something you want to use while coding.

6.3 Scripting

Scripting is also not covered in this document, but is a tool that could be useful to those who are interested. One advantage of using the command line is the ability to write scripts to perform certain tasks. For example, a script could be used to run many different test files on a program you have written. Scripts can be very useful and powerful for anyone who is interested in learning how to write them. Feel free to read more information on this topic if it is something you want to use while coding.