Linux Lab

Part 1

If you install a Linux virtual machine (or install dual-boot or single-boot Linux onto your machine) then you will get a full graphical windows-based interface in a similar manner to Microsoft Windows or Mac OS. You can use Linux to do most of your day to day jobs just through this mouse-driven desktop interface, but this isn't where the strength of Linux lies. In this lab we will not be using the menus and icons on the desktop, but we will be looking at using the Linux command line and some of the powerful commands that automate tasks compared to the graphical interface.

Knowing the Linux command line is also useful since you may find yourself interacting with a cloud-based computer using just the Linux command line. Or possibly a server machine within a computer rack using just console based command line input. But even if you have a graphical interface (like on Ubuntu or a Mac) – using the terminal command line can sometimes be much more efficient than using the GUI interface (particularly when handling lots of files).

Logging into Ubuntu cloud virtual machine

We will connect with a University Ubuntu virtual machine using the command line – similar to how you may connect to a remote server or cloud machines - using the "ssh" command.

One way to connect to the Ubuntu virtual machine is via the "Glasgow Anywhere Desktop". Double clicking the Penguin on this desktop will take you to the "Glasgow Anywhere Linux Desktop".

From within the lab, you may be able to connect to the Linux virtual desktop by typing the following into a terminal window:

ssh *your_GUID_username@10.224.160.71*

(This will only work if you are on the University network – whereas "Glasgow Anywhere Desktop" works anywhere !)

This is connecting to a Ubuntu virtual machine at the University of Glasgow. You will get a Unix console window which gives you command line access to the remote computer.

Opening the Command Line

We will use the Linux Terminal, a powerful Command Line Interface that once learned offers more functionality and flexibility than a point and click interface.

Today, we will start our exploration with three commands:

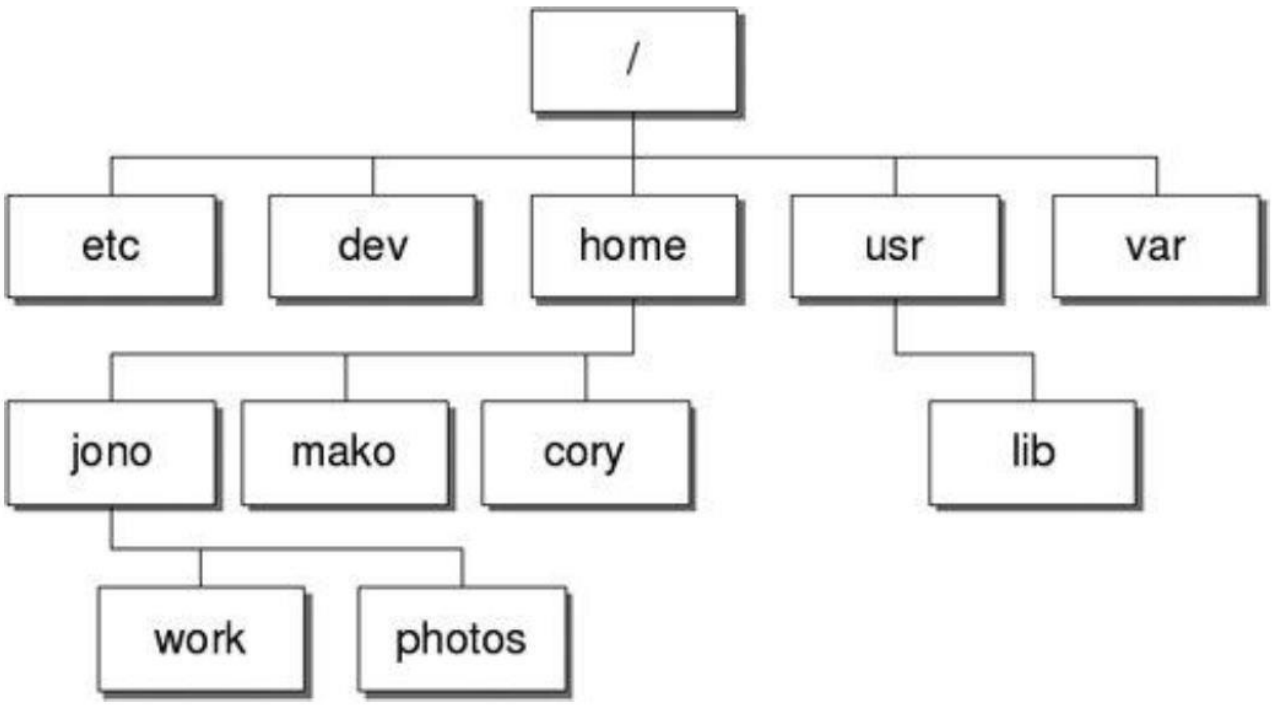
- **pwd** - print the current working directory
- ls – list the contents of the given directory
- cd – change the current directory

1. Type in to the terminal **pwd** and press return to execute the command. This will show you your current working directory. The terminal by default opens in your home directory. Try typing in **PWD**. What happens? _____

The Linux command line is case sensitive so when typing filenames and commands, you must use the correct case.

The Linux File Structure

There are two different types of objects that we will be working with: Files and directories. Files contain user or program information, like applications or word documents, music files, videos etc. Directories are folders that contain files and other directories. The Linux file system is usually represented as an inverted tree with **a special directory called the root at the top**. We reference the root using the backslash key '**/**'.



2. We're now going to change the current directory to the root. By typing **cd /** Remember to add a space between the command **cd** and the directory **/.**
3. Try checking the current working directory as before to check things have changed
4. We should now be in the root directory in the file system. We are now going to see what's in there by typing **ls** then **<return>**
5. ls lists the contents of the current directory. Compare what is in the root directory in your system with the root directory in the diagram. What is in your root directory that isn't in the diagram?

_____ _____

_____ _____

6. The diagram shows a hierarchical file structure. We're now going to navigate back through the hierarchy to get to our home directory again. We do this using cd by typing **cd home**
7. This takes us into the "home" subdirectory. If you now type in **ls** you can see all the directories in this home directory. What are all these directories? _____

8. If you type **ls -l** (that is minus then an "l" letter) then it does a "long listing" with more information about the directories. Can you access these other directories? (Recall how the file permissions work.) _____
9. Find your own GUID and **cd** into this directory.
10. Again use **pwd** to check that we're in the right place.
11. Navigate back to root. We're going to try using the tab key to help speed things up a bit. **Try typing cd /h then press the 'tab' key**. Tab can help auto complete file and directory names which is really useful for long file paths both to save time typing and make sure that the path is right.

12. Tab can also be used to make suggestions. In the root directory **try typing cd /m then tab tab**. What suggestions does it make? _____

13. Try navigating through a few different directories using tab to make suggestions
14. We're going to go back to root and try typing **cd ~** What does this command do? (Use **pwd** to see what directory you have gone back into.) _____

15. We want to see whether the file passwd exists in the directory /etc
16. Navigate to /etc and try ls

This returns a whole load of results and what would be helpful would be some way to filter them. We're going to use a special character to help filter our results. The * character is called a wildcard, and we can use it to stand for any character, or any group of characters. For example **b*d** could stand for **bad**, **bed**, **bud**, **band**, **ballad**, **broad**, or even just **bd**

17. Try **ls p*** This will list all the files starting with p and will list all the directories (and contents of the directories) that start with p. This can get a bit confusing.

18. To tell it to stop listing everything in the directories, we need to use a command line option. Type **ls –d p*** The –d for ls option tells Linux to list the names of the directories but not their contents. We should now be able to see our passwd file. (Try **man ls** to read about the "-d" command line option used for the ls command.)

19. We can also narrow things down a bit further in a number of ways. Using **ls –d** how many files or directories can we see using
    a. pa* _____
    b. pas* _____
    c. *d _____
    d. p*d _____

20. How would you list all files starting with p and containing the letter d? _____

To help you reflect on what you have just covered please fill in the blanks in the table below

Command	Description
...	Allows user to move around the file hierarchy
cd ~	...
...	Allows user to move to the root of the file structure
	Autocompletes the current path or suggests possibilities
ls -d	...
...	List all files or directories starting with 'b' and ending with 'd'

Part 2

Part 1 Review Questions

1. Open a terminal and write down the current working directory _____
2. Switch to the root directory. Which command did you use _____
3. How many files and directories in /etc end with the letter 'p' _____
4. How many files and directories in /etc end with the letter 'f' and contain a 'q' _____
5. Switch back to your home directory. What command did you use _____

Relative Addressing

In the previous section, we practiced switch between directories in the file structure using cd to change directory, pwd to show the current directory and ls to show the contents of the current directory. In this part we're going to extend this and start looking at the concept of addressing. In particular, relative addressing. Anytime we start a file path from the root directory, we call it absolute addressing. We know that we're using absolute addressing when our file path starts with **/.** For example

cd /home/mireilla

gets us to mireilla's home directory wherever we are in the file system.
If we don't start with the **/** Linux uses the path for the current working directory.

6) Let's type in **cd /home** to get to the directory home in the root directory. List the contents of the directory. What do you find in this directory _____

7) We can now relative addressing to go straight to the mireilla directory without having to address it from the root directory. Type **cd *your_GUID* <tab>** (note the lack of the **/** that we use for absolute addressing. Without this **/** we use relative addressing and we work from the current working directory.

8) Go to **/usr** and list the contents.

9) Now go straight to **sbin** and list the contents using relative addressing. What two commands did you use_____

We now know how to step one step downwards through our file structure but how do we move back up? We're now going to introduce two more special symbols that like **/** and **~** allow us to navigate the file system in different ways.

10) Go to your home directory.

11) Type in **cd ..** Where are you in the file system now and what do you think **..** stands for?

_____

12) Now try **cd .** Where are you in the file system now?

_____

13) Move to a different directory and try again. what do you think **.** stands for?

_____

.. allows you to reference the parent directory whereas **.** allows you to reference the current directory. A lot of the time we can reference the current directory without using . We will see examples later of when we need it, but for now it is enough to know that if we were in the **/home** directory, the command **cd mireilla** would be equivalent to **cd ./mireilla** and the command **ls .** is equivalent to **ls**

14) Go to your home directory. Using relative addressing only and in one command only, how would you move to the root directory? _____

15) We can use both absolute and relative addressing with commands other than **cd**. Go to the home directory and type in **ls /dev** to list all the files and folders in the /dev directory (note that you haven't used cd so are still in your home directory).

16) Now using relative addressing only, and in one command, list the contents of **/etc**. What command did you use? _____

17) We can also combine file paths with the ***** wildcard we saw in the previous lab. Using relative addressing only, and in one command, how would you list all the files in the **/etc** directory that start with the letter **d** and contain an **n**? _____

18) Practice, practice practice !!!! The best way to get use to navigating the file system is to practice. Try as many examples as you need moving up and down the file system until you feel that you are comfortable.

Creating Files & Directories

There are lots of different commands we can use to create a new file in linux. To just create an empty file we can use the touch command.

19) Make sure that you're in your home directory (you can actually go to your home directory by just typing **cd** without any arguments). Try typing **touch touchtest.txt** and check that the file has been created. To create multiple files at the one time, we can use touch with many parameters with a space in between (e.g. touch file1 file2 file3).

20) Check that it's empty by typing **cat touchtest.txt** . The command cat prints out the contents of the file that you give it. This confirms the file is empty.

21) Alternatively try opening it from the command line by typing **vi touchtest.txt** or **nano touchtest.txt**. (Remember you can use <tab> completion for the filename to quickly type it.)

(Also note that you get out of the **vi** editor using the commands ":q!" to quit without saving. The nano editor provides information about what keys to use on screen.)

We can now create an empty file and edit it with **vi** / **vim** / **nano**. How about creating a non-empty file. Again we can do this in different ways, but today we're going to use **echo.**

22) Type **echo Hello World** What does this do? _____
echo writes text to the screen, but with Linux we can redirect the output to a file using the **>** key and then supplying a filename. If the file doesn't exist, it will create the file for us.

23) Try **echo Hello World > echotest.txt**. Use **ls** and **cat** commands to find out what this command does.
_____

24) We're now going to create a directory in the current directory. This is really easy to do. We just type **mkdir testdirectory** which creates a directory called test directory in the current directory. How many files now exist within testdirectory? _____

Moving, Copying & Deleting

The final things we are going to be looking at in this part are how to move, copy and delete files and folders. Move and copy are similar operations but have slight differences in their outcomes.

26) First of all we're going to move a file into our new folder. We do this using **mv file_to_copy destination_directory**. Move touchtest.txt into testdirectory. Explain in words what effect this command has _____

27) Similarly, we can copy files using **cp file_to_copy destination_directory**. Copy echotest.txt into test directory and explain in words what effect this has on the file system and how it differs from mv.
_____
___
_____

28) We can also use mv to rename files in our home directory, try **mv echotest.txt test2.txt** What effect does this have on the file system? _____

29) Similarly with cp, we can create a copy of the file with a different name using **cp test2.txt echotest.txt** Try this and confirm it works.

We're now finished with our new files and directories so we're now going to clean up. Firstly by deleting the file echotest.txt and then deleting our directory and all of its contents.

30) To delete files we use the **rm** command (short for remove). Type **rm echotest.txt** to remove our file and check that it's been deleted.

31) Try deleting testdirectory. What happens ? _____

32) We need an additional parameter to remove a directory and all of its contents. We do this using **rm –r directory_to_delete** Remove testdirectory and check that it has been deleted.

To help you reflect on what you have just covered please fill in the blanks in the table below.

Command	Description
...	Move to the parent directory
touch myfile1	...
...	Create a file myfile2 containing the text Hello World
gedit myfile &	...
	Copies file1 to the directory mydir
mv file1 file2	...
...	Moves file1 to directory1
rm file1	...
...	Deletes directory1 and all of its contents.

Part 3

Some Useful Programs

Here, we continue on from where we left off in the previous section. Make sure that you've finished everything in part 2 before starting here. Let's start with, a quick refresher.

Refresher

1. Open a terminal. What does the command **ls ../../etc** do?_____
2. Move to the directory /dev in one command, using relative addressing only. Which command did you use? _____
3. Go to your home directory, then create 3 empty files called test1, test2 and test3 with one command. Which command did you use?

_____

4. Rename test1 to become test4. Which command? _____
5. Make a directory called test_dir and copy in test2. Which two commands were required?

   _____
6. Delete all the test files that you created. Which commands did you use?

_____

7. Delete test_dir. Which command did you use?_____

Some More Useful Commands

You really don't need to know a lot of commands before you can do a lot of useful things with the command line. There are a large number of programs that you can combine in different ways to provide a huge amount of functionality, however in day to day usage, we usually only need a small subset. In this lab, we'll be looking at a few more useful commands. In particular, we will look at **cat, head, tail, ps**, **kill**, and **man**.

cat

cat is short for concatenate as it allows us to join two text file together. However, it has other uses which we will now explore.

8. Make sure that you're in your home directory and create a text file called file1.txt containing the words "Hello World". If you can't remember how to do this, look back to echo in part two.

9. Create a second text file called file2.txt containing the words "Goodbye there".

10. Firstly we'll try cat with one argument. Try **cat file1.txt** What does this command do ?
_____

11. Let's try it with a longer file. Use cat to show the contents of /etc/passwd. How would you do this without changing directory?_____

12. Now let's try cat with two arguments. In your home directory, try typing **cat file1.txt file2.txt** What does this do ? _____
With **cat**, we can use as many arguments as we want to join together the contents of as many files as we want. We can use the same trick as we used in the last lab (when using echo) by using **>** to send the output of **cat** to a file.

13. Now try the same command as before to concatenate file1.txt and file2.txt using the **>** character to redirect the output to file3.txt. What does this command look like
_____

14. Finally we try using cat without passing it a file. Try the command **cat > catfile.txt** Then type in **One Two Three Four <Return key> five six seven eight <return key> nine ten eleven twelve.** Finish by pressing the **control key** and **d** together. That's **Ctrl + d.** You should see a file catfile.txt has appeared in your current directory.

15. Use the command **cat catfile.txt** to find out what the command in step 14 did?
_____

We've now seen three uses of cat. To list the display of a text file (when we pass it one file), to join multiple files together (when we pass in multiple files), and to create and write to a text file (when we pass in no files). Cat gives us a low effort way to view and create text files. When combined with the redirect character **>** this provides us with a really quick and easy way to manipulate text files. We can use this to redirect the output of any command.

16. Try typing **ls /etc > test.txt**. What does this command do ?

head & tail

17. Using the technique in step 14, create a text file called testfile.txt, but this time with numbers 1 to 20 each on separate lines.

18. What does the command **head testfile.txt** do?
_____

19. Let's introduce an argument to head. Try **head –n 3 testfile.txt**. What does this do differently?
_____

20. Do the same two commands work with **tail**?_____

21. Explain how the **tail** command differs from the **head** command._____
_____

ps & kill

We don't have time for a full look through what is happening with linux, but to show you a little of what is going on under the hood and to learn a useful technique we are going to look at the commands **ps** and **kill**. **ps** allows you to see the current processes running on your machine. What is a process? When a program gets executed, an instance of that program is created and run by the operating system. This running instance of the program is called a process.

22. Run the following command (this is written in bash script which we will be looking at shortly):

    ```
    for i in {10..0}; do sleep 3; echo $i; done
    ```

    This will count down from 10 to 0 – essentially going around a "for loop" – sleeping for 3 second each time – before writing out the current number. If you get tired of how slow it is counting down … you can press Ctrl-C keys and terminate it since it is running in the foreground (i.e. connected to your keyboard input).

23. Running this command takes over the current shell while it is counting down from 10. If you type **ls** while this count is ongoing – nothing happens – until the end of the count when a file listing is produced (since this command is stored in the keyboard buffer and is executed once the previous command is finished). Essentially the shell executes commands in order and needs to finish one command before it can start another.

24. Now type:
    ```
    for i in {10..0}; do sleep 3; echo $i; done &
    ```

    The only difference is the "&" at the end which tells bash that this should be run in another shell in the background (essentially running concurrently or at the same time). If you now type **ls** into the console then this **ls** command will be run immediately while the counting continues ! (Since the two processes are running concurrently, or at the same time, with one in the background and one in the foregrounds.)

25. Type **ps -f** at the console when nothing else is happening. How many processes can you see running? What are their names? The columns show the UID (User ID), PID (Process ID), PPID (Parent Process ID), TTY (terminal or output console the process is connected to), and CMD gives the command. Can you

explain the processes that are running when the **ps -f** command is executed? Do the child-parent process relationships make sense?

_____

_____

_____

26. Run the loop in the background again and quickly do a **ps -f** command. Do the processes showing make sense? Can you see that the background loop is running in another bash shell process? What is the process ID for the shell that is running the loop in the background?
    _____

27. Isn't it annoying that this loop takes a very long time in the background and we cannot seem to stop it printing to the screen? If a program is misbehaving we need a way to deal with it. We can do this using the **kill** command. Try **kill processID** (where the processID is your answer to step 26). What does this command do?_____

28. Try ps again and check what processes are running. You should see a process bash. Try and kill the bash process using its Process ID. What happens?_____

29. Some important processes need an extra option to stop them. To stop these processes, we use the kill 9 option. Try killing the bash process with the -9 option. For example, if the bash process ID was 2165 type **kill -9 2165**. What happens and why?_____ _____

man

We have saved the most useful command for last. **man** is short for manual and it allows you to find out about other commands; both what they do and all the different ways that you can use them. We will finish off by using man to find out about some of the commands that we have been using. First we will use it with pwd.

Type **man pwd** What does the manual page for pwd say that it does? _____

_____

29. Press q to get out of the pwd manual page. Now let's try it with ls. Open the manual page for ls. We are going to look at some of the options available with ls. Use the up and down arrow keys on the keyboard to find the description of the –d option. What does it say?

_____

30. We are going to use man to find out how to list our current directory using long listing format. Which option do we need? _____

31. Use this option on the /home directory. The 6th column shows the size of each of the entries in bytes. What size are all the directories?_____

32. Use man with ps to find out how to list all processes currently running on your machine (includes system processes as well as your own). How many are there

Less than 10 Between 10 and 20 More than 20

33. What do you think **man man** will do? _____

When finished this lab, use the command and keyword sheet to work through everything that we have learned in these tasks.

Make sure that you are confident moving between directories and practice creating file and directories and copying and moving them to different places. Use man to explore all of the commands we have looked at so far and see if you find any interesting or useful options.

Command	Description
...	Show the whole contents of file test1.txt on the command line
cat file1 file2 > file3	...
...	Show the first 10 lines of the file text1.txt
tail –n 5 test1.txt	...
...	Show the current user's running processes
kill 2156	...
...	Show all processes running on the machine
man ls	...

Part 4 – Basic Shell Scripting

Here, we continue on from where we left off in the previous lab. Make sure that you've finished everything in lab 3 before starting here.

Refresher

1. Use the **cat** command to create a file called numbers.txt containing the numbers 1 to 3 on separate lines. Which command?_____

_____

2. Create a file containing "Hello World" and a second file called joinedfile that joins the contents of number.txt and your hello world file. Which two commands?

_____

3. Show the first 2 lines of numbers.txt Which command?

_____

4. Write out the last 2 lines of numbers.txt to the file lastnumbers.txt. Which Command?

_____

5. Show all of the processes currently running on the machine. Which Command?

_____

6. Show the manual page for mv. Which Command?_____
7. Find the option in the cp command that will interactively ask the user if they wish to overwrite a file if the destination file exists

_____

Shell Scripts

A script is a text file that contains a series of commands. This allows us to group together multiple commands and run them using one line of code. There will be no new commands to learn in this section (well maybe just the one or two to help with the scripting). Instead we're going to be using the commands that we already know and combining them into scripts.

Creating Our First Script

A script is just a text file so we can use any of the techniques that we have learned to create files (like **echo**, **cat** or **vi / vim / nano**) to create our script. The one that gives us the most flexibility in our current set up however is **vi / vim / nano**. This program allows us not just to create a text file but to easily modify it, an important fact if we're going to be developing programs that may have bugs in them.

8. In your home directory, ***create a directory called scripts*** and in it, create a file called *helloworld* script and open it in **vi / vim / nano**. (You may want to open two console windows connected to the Linux machine – one for editing the file and the other for running it.)

9. Our first script will be as simple as we can make it, just a single command. Enter the line **echo !!! HELLO WORLD !!!** and save the file.

10. Let's try running out script and see what happens. Make sure that we're in our scripts directory. Linux (in the default set up) doesn't look in the current working direct for scripts so we will need to be specifying directly the current directory using the **.** directory when we try and run the script. Try **./helloworld** Why doesn't it work ?

_____

Now there are different ways that we can run the script. Today we'll focus on one that avoids the rather messy business of Linux file permissions which I want to avoid completely given our short time with Linux. *If you are interested in looking at the other (more commonly used) method of running a script as we tried above, you need to look at the **chmod** command that will allow you to make your script executable.* In this section, we won't be doing this. We will use the command **source** which will take our script file and run it without the need for it being executable.

11. Now we use source to run our script. Type in **source helloworld**
What appears on the
screen?_____

12. We can run the script from whichever directory we are in. Go to the home directory. Which command lets us run the helloworld script without changing directory? (Remember it is in the scripts directory.)

_____

13. We're now going to modify our script to try a few things. First question. If we run a script from a different directory as above, what will the current directory be? That is, if we use the keyword **.** in our script, will it refer to our current working directory, or the directory that the script is in. Modify our script to run the command **pwd** command instead of echo (make sure that you save it). Run it from different directories. Is the current working directory in the script our current working directory or our scripts directory ?_____

We've now seen a script run, but why is scripting a useful tool? It allows us to do multiple jobs with the one command saving time typing the commands while also giving us a series of commands we can reuse whenever we want. We're now going to do a longer script with a sequence of commands to carry out with **one command on each line**. Remember, these are all commands that we've seen before.

14. Create a script in your scripts directory called *list_files* that will list all files in the **/dev** , **/etc** and **/bin** directories that start with the letter **t**. Use the echo command to label the output so we know what files come from what directories _____

_____ _____

_____ _____

_____

15. Now let's try something a bit more complicated. Create a script in your scripts directory that when run creates 2 files file1 and file2, the first containing "Hello World" and the second containing "Goodbye". Have your script create a directory called mydir and in that directory create a file called output that contains the contents of file1 joined up with the contents of file2. What does your script look like?

16. Now create a script in your script directory called cleanup that deletes all of the files and directories created by your previous script. What commands do we need?

User Input

If this was all we could do with scripting, that would mean every time we ran the script, the same things would happen to the same files. This would be quite restricted. What we need is a way of getting user input that will allow us to run the same series of commands with different data and with different files.

17. Create a script in the script directory called *names*. The script should contain the following commands

```
echo Enter three names
read FIRSTNAME SECONDNAME THIRDNAME
echo $THIRDNAME $SECONDNAME $FIRSTNAME
```

Describe what this script does.

_____

_____

_____

Here we are using variables that store values. FIRSTNAME, SECONDNAME, and THIRDNAME act as boxes where we can store user input. Then using the **$** keyword we can get the value back. We can give these variables any name we want.

18. Write a script called *create_file* that asks a user for a file name and contents of the file and creates the file with the given contents. What does your script look like?

_____ _____

Command Line Arguments

The final scripting tool that I'd like to look at allows us to use command line argument. What do I mean by that? Well if I wrote the command **`cp file1 directory1`** the first command line argument would be file1 and the second directory1. We can get access to any number of these using special variable names. The first is **`$1`**, the second **`$2`** and so on.

19. Let's write a script that simply list 3 command line arguments. Your script should contain only one line **`echo $1 $2 $3`** Confirm this works. What happens if you enter more than 3 command line arguments?

_____

20. What happens if you don't enter enough command line arguments?_____
    _____

21. Try rewriting the above script in 17 using command line arguments instead of reading the user's input. What does the script look like

    _____
    _____
    _____
    _____
    _____
    _____

    Try rewriting the above script in 18 using command line arguments instead of reading the user's input.

    _____
    _____
    _____
    _____
    _____
    _____

Option Extra

22. Write a script that asks the user for an output file name and for the name of three directories. Create a file with the appropriate output that contains the contents of all three directories.

Review the previous material by filling in the following blanks:

Command	Description
...	Reads keyboard input from the user into variables var1 & var2
source ~/scripts/script1	...
...	Show the first 10 lines of the file text1.txt
$var1	...
...	Keyword for the first command line argument

Command	What does it do?	How to use it
pwd	**Prints the current working directory**	**pwd** prints the current working directory
ls	**Lists the contents of the path specified. If no path specified we use the current directory**	**ls** lists the contents of the current directory **ls /etc** lists the contents of the /etc directory
cd	**Changes the current working directory**	**cd Documents** changes to the documents directory found in the current working directory **cd /etc** changes to the etc directory found in the root directory
~	**The tilde character is a keyword for your home directory**	**cd ~** changes the current working directory to your home directory (/home/brian in our labs) **cd ~/Documents** changes to the Documents directory in our home directory
/	**The slash is a keyword for the root directory of the file system**	**ls /** lists the contents of the root directory
*	**A wildcard that can stand for a string of characters of any length (including and empty string)**	**ls ../p*.txt** list all files in the parent directory that start with p and end in .txt **cp * dir1** Copy all files in the current directory into dir1
..	**Keyword for the parent directory**	**cd ..** Change directory to the parent directory
.	**Keyword for the current directory**	**cp Documents/test1.txt .** copy the file test1.txt in the Documents directory to the current directory
&	**To start a process as a background process**	**gedit file1 &** Opens file in gedit as a background process so that commands can still be run in the terminal while gedit is open.
cp	**Copies the file specified to the directory (or file given)**	**cp file1 dir1** Copy file1 into the directory dir1 **cp file1 file2** Create a copy of file1 with the name file2 **cp *.txt dir1** Copy all files ending in .txt to the directory dir1
mv	**Moves or renames a file**	**mv file1 dir1** Moves the file file1 into the directory dir1 **mv file1 file2** Renames file1 with the name file2 **mv *.pdf dir1** Moves all files ending in *.pdf to the directory dir1
mkdir	**Creates a directory**	**mkdir dir1** Makes a directory in the current directory called dir1
rm	**Deletes a file or directory**	**rm file1** Deletes file1 **rm *.txt** Deletes all files ending in .txt **rm –r dir1** Deletes the directory dir1 and all of its contents
touch	**Creates an empty text file (can also update the last modified date on a file if the file already exists)**	**touch file1 file2** creates two empty files called file1 and file2 (as long as they don't already exist)
echo	**Writes out a string of text**	**echo Hello World** Writes "Hello World" to the screen **echo Hello World > file1.txt** Creates a file called file1.txt with the text Hello World in it
>	**A keyword to redirect the output of a command**	**ls > file1.txt** lists the contents of the current directory and writes the output to the file file1.txt
cat	**Used to display, concatenate of create files**	**cat myfile.txt** Displays the contents of myfile.txt **cat file1.txt file2.txt > file3.txt** write the contents of files1.txt and file2.txt to file3.txt **cat > file1.txt** Allows a user to type a string with ctrl+d to finish. File1.txt is created with the contents of the string typed
head	**Displays the first n lines of a file (n by default is 10)**	**head file1.txt** displays the first 10 lines of file1.txt **head –n 5 file1.txt** displays the first 5 lines of file1.txt
tail	**Displays the last n lines of a file (n by default is 10)**	**tail file1.txt** displays the last 10 lines of file1.txt **tail –n 5 file1.txt** displays the last 5 lines of file1.txt
ps	**Displays the processes running on the machine**	**ps** display my running processes **ps –A** display all running processes
kill	**Terminates the given process**	**kill 2456** terminates the process with PID 2456 **kill -9 2456** terminates forcefully the process with PID 2456
man	**Displays the manual page of the given command**	**man ls** Display the manual page for the command ls (q is used to exit the manual page