Workshop Task – Command Line Familiarisation A

We will issue commands at the command prompt. The command may be followed by "arguments"…

```
( $ command [argument1] [argument2] [argument3] […] )
```

***Note**: In command line syntax, square brackets [ ] are used to indicate optional arguments*

We will denote the user's command prompt with a 'dollar' / 'string' sign ("`$`"). You do not need to type this character, you can see it on the command line!

Try each of these in the terminal, one by one:

```
$ whoami
$ pwd
```

But what are these commands showing you? We can look at their manual pages:

```
$ man whoami
$ man pwd
```

…you can scroll up or down using the arrow keys. Press the 'q' key to quit from the man[ual] pages.

`whoami` tells you your username, and `pwd` is 'print working directory'; it prints the name of the current working directory to screen.

Almost every command has a *man* page; even the '`man`' command has a manual page:

```
$ man man
```

…For now, quit ('q'), and you can look at some man pages later.

Now let's try some other commands, figure out what they do:

```
$ ls
$ ls -l    <-- that's a dash (minus sign) followed by a lowercase letter
           "L"
```

…here we can see a list of files and directories (folders).
We can see the file size as well as other information.
What's all that nonsense to the left? (`drwxr-xr-x.`)
Why does it say `cyber` twice? `<-- try but don't worry if you can't figure it out - we will cover it next week`

```
$ cd
```

…what did that last one do?
Check out its man page… …???

`cd` is short for 'change directory'. Let's practice with it. Change into one of the directories (folders) you saw a moment ago with the `ls` command:

```
$ cd <directory> <-- don't physically write "directory" but write one
                     of the directory names you saw earlier.
```

…have a look around – see what's in this directory. (Use `pwd` and `ls` commands.)
We can move down the tree of directories (towards the leaves / sub-folders), but how do we move back up to the parent directory?

```
$ cd ..     <-- that's cd followed by a space then two dots (full stops). The
            two dots is the first argument to the cd command.
```

You should be back to the parent directory. (Again use `pwd` and `ls` commands.)
You can go up the directory tree even further, or further down to the child/sub-folders.

Note: *Directory (in Linux) is the same thing as Folder (in Windows)*

Now, let's move to the home folder for the user (the username was given by the `whoami` command). The home folder is the one you started in when you opened the terminal. Simply type cd at the command line prompt. You should see the following full prompt…

```
[cyber@localhost ~]$
```

…notice the '~' (tilde). This means you are in your "home" directory for that user 'cyber'.

Let's make a directory. `mkdir` command is used to make a directory:

```
$ mkdir
```

…oh, it wants an argument (a parameter value that is passed to it after the command, just like we did with '`ls`' command earlier when we passed it the '`-l`' argument).

Make a new directory with a name of your choice, e.g. `my_folder`:

```
$ mkdir <my_folder>   <-- don't physically write "<my folder>" but write a
                      name of your choosing. (Don't use '<', '>', '$' or
                      other special characters in the folder name.)
```

Again, see that it is there (use `ls`, or `ls -l`), move into it (use `cd`, then `pwd`, `ls`).

Make some more directories; something like this tree structure (including its faults):

```
<my_folder>
+- Fruits
|     +- Lemon
|     +- Pear
|     +- TEMPORARY_FOLDER
|           +- JUNK
|     +- Orange
|     +- Doggggggggs
|           +- Hound
|           +- Shitsu
|           +- Whippet
+- Vegetables
|     +- Carrrrrot
|     +- Spud
|     +- Rocks
|           +- Granite
|           +- Marble
+- Animals
      +- Cat
      +- Zebra
```

Oh dear. Looks like we've made some mistakes. That was unforeseeable (/sarcasm). Let's correct it using as few commands as possible. To do this, we'll also need the `mv` command, which is short for "move". This command can not only move files and folders, but can also rename them when you do it. Finally, we'll need the `rm` ("remove") or `rmdir` ("remove directory") command to get rid of that 'TEMPORARY_FOLDER'.

Can you do it? We want this…

```
<my_folder>
+- Fruits
|     +- Lemon
|     +- Pear
|     +- Orange
+- Vegetables
|     +- Carrot
|     +- Spud
+- Rocks
|     +- Granite
|     +- Marble
+- Animals
      +- Cat
      +- Zebra
      +- Dogs
            +- Hound
            +- Shitsu
            +- Whippet
```

Did you try removing the 'TEMPORARY_FOLDER' with `rmdir`? It won't work if something is in the folder. Try '`rm -rf <folder_name>`'. Check out the man pages for what 'r' and 'f' **switches** do, or ask the tutor.

Once we've finished, we can make the whole tree of directories to be read only (non-writable). At the base of the tree list the r/w/x details of <my_folder>…

```
$ ls -l
```

…we can see that you can read and write the folder.

Let's make it non-writable by using the '`-w`' flag on the 'change mode' command…

```
$ chmod -w <my_folder> <-- you may also use a-w instead. Details next week
```

…and then check…

```
$ ls -l
```

If we look inside the directory though, the directories within it are still writable (and therefore changeable).

Let's recursively make every directory non-writable by also using the '`-R`' (Recursive) flag…

```
$ chmod -R -w <my_folder>
```

Again, we can check the r/w/x flags at any time by using `ls -l`.

An ‘x’ on a file means it is executable, but an ‘x’ on a directory means you can `cd` into that directory.

Move back to home (‘`cd`’ or ‘`cd ~`’) and make a directory; test, or some other name.

Let’s try printing to the screen…

```
$ echo
$ echo hello
$ echo "hello"
$ echo HELLO
```

…and some more…

```
$ echo ROAD
$ echo PATH
$ echo $PATH
```

…oh, that last one was a surprise!

PATH is one of many ‘**environment variables**’. You can access environment variables by putting a ‘dollar’ / ‘string’ sign (“`$`”) before the enviro variable name. We can see all of them, by typing

```
$ env
```

Make your own variable; leave no space either side of the assignment operator (“=”):

```
$ MEEP=<my_folder>
$ echo $MEEP
```

We are actually in a scripting language called “shell”. (Particularly BASH – Bourne Again SHell). We can use the value usefully…

```
$ cd $MEEP
```

…to change into the directory you created earlier (`<my_folder>`), or more complicated things.

The variable MEEP, which may hold a string value, is stored in the shell as a **local variable**. If you run another program from the shell (the terminal) then that program cannot access the variable.

If you want to pass the variable to other programs, then you have to export it to the “environment” (as we previously inspected with the `env` command). You can create and export a new environment variable MEEP with its value set to `<my_folder>` as follows…

```
$ export MEEP=<my_folder>
```

…the quotes are optional here, but are needed if you include whitespace in the string.

Now let’s make some text files and look at them:

```
$ echo “hello” > my_file <-- Write a target filename of your choice.
                             (Don’t use ‘<’, ‘>’, ‘$’ or other special
                             characters in the folder name.)
```

…this didn’t display on screen. Where did it go? Have a look with `ls`.
The ‘>’ after “hello” and before the file name means send it to the file `my_file`.

List the contents of files:

```
$ cat my_file
```

…cat is short for conCATenate, which joins files, or, if there is only one file given as an argument, then just prints it out to the 'standard output' (i.e. the screen).

What kind of file is that?

```
$ file my_file
```

…it's 'ASCII text', which is a standard text format that we can read with a text editor (and the cat command).

What about these – make your own notes and practice using them:

```
$ uname -a
```

…useful system information.

```
$ whereis firefox
```

…firefox is in the '`/usr/bin/`' directory. Explore it with `ls`; what other programs are in that directory?

```
$ ls -l /usr/bin/
```

What kind of file is firefox?

```
$ ls -l /usr/bin/firefox
$ file /usr/bin/firefox
```

Also…

```
$ which firefox

$ cat /proc/cpuinfo
$ cat /proc/meminfo

$ df

$ su
```

…become root/admin, if you have the password.

```
$ gcc
```

…the 'GNU C Compiler'

Other commands to explore. You may need to read about these, and to pass arguments to them.

```
$ touch
$ diff
$ chmod
$ chown
$ chgrp
$ find
$ grep
```

Did I miss any commands? Most likely, - we'll learn new ones as we need them, but don't hold back. Read some Linux command line tutorials.