The *nix Command Line

Fionn Fitzmaurice `fionnf@maths.tcd.ie`

Most modern Unix-like operating systems (GNU/Linux, OSX, etc.) have a graphical way of interacting with the computer, but there is a much more efficient way of doing things: by using the command line. This should hopefully explain some basic commands.

First, look at the prompt. It should look something like `pwd`, `echo`

```
boole ~$.
```

I'll use `$` (instead of `boole ~$`) for the prompt from now on. `boole` is just one of the servers we have, but it doesn't really matter which one you use. The tilde tells you that you're in your home directory. To find out where that is, type `pwd` (print working directory) and press enter. If your username was `einstein`, you would see

```
$ pwd
/u3/maths/2017/einstein.
```

You could also type `echo $HOME`. The `echo` command prints its arguments to the terminal, in this case the value of the variable `$HOME` which is the path to your home directory.

To list the contents of the current directory, type `ls`. This command can take an argument, just like `echo`. If the argument is the name of a directory, `ls` will print its contents. There are also a number of options (called flags), such as `-l` which prints more information and `-a` which prints hidden files. More than one option can be used at a time, for example `ls -lah`. `ls`

To change directory, use `cd /path/to/directory` (for example). You can also use relative paths, so `cd ~/Downloads` and `cd Downloads` are equivalent if you're already in your home directory. A quick way to get home is by calling `cd` with no arguments, equivalent to `cd ~`. Some useful shorthand is `.` (which refers to the current directory) and `..` (which refers to the parent directory), so if you're in `/path/to/directory` and you type `cd ..`, you would change to `/path/to`. Notice that your prompt changes when you change directory. `cd`

To create an empty file use `touch filename` and to create an empty directory use `mkdir directoryname`. `touch`, `mkdir`

The command `mv` moves files and directories. `mv ~/photos/fun.jpg .` would move `fun.jpg` in the `~/photos` directory to the current directory. Then typing `mv fun.jpg badtimes.jpg` would rename `fun.jpg` to `badtimes.jpg`. You can copy with `cp`, which works identically except it leaves the original file alone. `mv`, `cp`

The `rm` command removes files. `rm badtimes.jpg` would remove `badtimes.jpg`. In general there will be no confirmation and the file will be permanently deleted, so be careful. If you're feeling unsentimental and want to delete the entire `photos` directory, you might try `rm`, `rmdir`

```
$ rm photos
rm: cannot remove ‘photos/’: Is a directory,
```

so `rm` cannot remove directories. `rmdir` can, but only if the directory is empty. To remove `photos` and all its content, the `-r` (recursive) flag can be used (i.e. `rm -r photos`).

Editors

To edit a file, for example when writing a program, you can use a text editor like `nano` (CLI) or `gedit` (GUI). If you want to be really cool, you might try `vim`.

`less`

Once you've written something in your text file, the `less` command will display its content on the terminal. Type `q` to exit.

`man`

Each of these commands can be thought of as a small program that you run from the command line. Programs that are installed generally come with a manual page which documents their usage and so on. This can be found using the `man` command, for example `man ls` or `man rm`. Of course, `man` is also a program and comes with a manual page; try typing `man man`.

`gcc`

In particular, `gcc` (the GNU C compiler) is a program used to compile C code. `gcc helloworld.c` would produce an executable program called `a.out` in the same directory. Generally, just typing `a.out` won't run the program (unlike the way typing `less` would run the `less` program). This is because the shell doesn't know where `a.out` is, so you have to tell it to look in the current directory by instead typing `./a.out`.

`>`, `>>`, `|`

Perhaps you might want to print the output of a program not to the command line but to a file. A simple way to do this is by redirecting the output using `>`. For example, `echo "Fionn" > favouritetutor.txt`. Running it again won't add another "Fionn" to the filename, but will erase the original and repeat the command. To append text to the file, use `>>`. There's also a way to pipe the output of one program into another as input, using the `|` character. Try `ls | less`, or `ls | grep o`.

If you want to kill a process, type Ctrl+C. Ctrl+D will ask a process to stop, but doesn't always work. Because Ctrl+C already has a meaning, copying from the terminal has the shortcut Ctrl+Shift+C and pasting Ctrl+Shift+V.

Some of the features that make the command line so powerful are history, tab-completion and wildcards. Try pressing ↑ a couple of times; this is a quick way of repeating tasks. Suppose you have a directory containing {`a.txt`, `b.txt`, `c.txt`, `x.cpp`, `y.cpp`, `z.cpp`}. If you want to read the contents of `a.txt`, rather than typing `less a.txt` it would be enough to type `less a` and then hit Tab which would autocomplete the command for you. If you don't give enough information for the shell to uniquely autocomplete, hitting Tab again will list all of the possibilities. This will also work for program names. Now, suppose you want to delete all `.cpp` files. We can do this quickly using the wildcard `*`, which stands for any string. `rm *.cpp` would do the trick. You now know enough to figure out what `rm -r *` would do; never run `rm -r *`.

Some very useful programs are `ssh`, `scp`, `grep`, `pdflatex` and `cat`. You will probably end up using these programs quite often; try looking up their manual pages.