A Quick Introduction to Working in the Unix Environment

Conventions used in this document

- Type that will be input at the command line or type that you can expect to see on the screen will be in a mono spaced font. For example:

  ```
  chmod +x perlprogram.pl

  myusername> ps
      PID TTY      TIME CMD
     8152 pts/1    0:00 bash
     8203 pts/1    0:00 ps
  ```

- URLs are written like this: `www.ncbi.nlm.gov`.
- Things to be mindful of are written in italics: *Be careful when using this command.*

The Unix operating system and logging on

Originally developed in the late 1960s, Unix is a multi-user, multi-tasking operating system popular in server environments and academia. The popular open source Linux operating system is one of the many varieties of Unix. Because of its popularity in academic circles and its ability to handle large data processing tasks, most of the the major genome assembly/analysis software programs are designed to run under Unix.

As mentioned above, Unix is a multi-user operating system. Multiple users can log into the one computer and run programs. The Melbourne School of Engineering has three UNIX servers: `nutmeg.eng.unimelb.edu.au`,`dimefox.eng.unimelb.edu.au`, and `digitalis.eng.unimelb.edu.au`. These machines all access the same home directory with the same files, no matter which computer you log into.

From outside the university, you must use the VPN to access these machines. See the instructions on the LMS on accessing the MSE machines.

The university's MS Windows computers have a program installed entitled `MobaXterm`. It can be found on the laboratory machines under:

Start→MobaXterm Personal Edition→MobaXterm Personal Edition

As `MobaXterm` starts, a window will pop up requesting changes to your firewall. Select cancel and continue. A window with a command line prompt (probably `'$'` will appear on the right.

To connect to the Melbourne School of Engineering server `digitalis`, type at the prompt:

`ssh username@digitalis.eng.unimelb.edu.au`, where `username` is your `unimelb` student email username. If you are asked a question about keys, type 'yes' and press Enter.

You should now be on `digitalis`. If you are uncertain, type `hostname` at the prompt and look at the answer.

The window on the left shows the files in your home directory (or in whatever directory you `cd` to.

The `MobaXterm` program allows you to not only connect to a Unix computer via the command line, but also to transfer files back and forth from your Windows computer to the Unix machine. Because the software also includes an X Window server, it allows you to run graphical programs on the Unix computer and display them on your local MS Windows desktop. You can install `MobaXterm` on your own computer and logon to a Unix computer whenever you are connected to the internet (`http://mobaxterm.mobatek.net`). There is an excellent demonstration of `MobaXterm's` features at the software home page.

All Apple computers come with the `Terminal` application which can be found in *Go→Utilities*. Apple machines by default can run graphical Unix applications.

Click the `MobaXterm` application icon to launch the software. You should be presented with a black tabbed commandline interface window. You can launch multiple commandline sessions under different tabs. The login instructions are the same for both `MobaXterm` and the Apple `Terminal` application. You will be using the program **ssh** (secure shell) to login.

At the command prompt type:

```
ssh username@computer.csse.unimelb.edu.au
```

For example

```
ssh rshall@nutmeg.end.unimelb.edu.au
```

If you are asked about caching a key, type "yes" and press enter. You should then be presented with a unix prompt from the computer you have connected to.

Once you have logged into a computer, you will have a command line prompt similar to this:

```
 /storage1/alpha/users/rshall>
```

You can easily move to another computer by using the `ssh` command. For example, if you are on `nutmeg` and wish to move to `dimefox`

```
ssh dimefox.eng.unimelb.edu.au
```

will get you to `dimefox`. The command `hostname` will tell you which machine you are on.

The Unix prompt

The prompt is central to the Unix system command line interface. Commands are typed at the prompt, pressing the `Enter` key to complete the command. Output from the command may be information such as a list of files in the current directory, for the commans `ls` or it could be progress messages from a particular program which is saving its output to a file or files. The prompt can take a number of different forms but is usually a `$`, `>` or `%` and may be accompanied by other data such as your username or the server name etc. The Unix system is *case sensitive*, so all commands, file path names etc. must be typed with this in mind. The file path `/home/username/` is *not* the same as `/Home/username/`.

The Unix file system

The Unix file system is a hierarchical system similar to an inverted tree. The top level directory is designated the `root` or `/` and all other directories are below it. When you first login to the system your location in the file system will be your own home dirctory.

As Unix is a multi-user environment, provision must be made in the file system to allow the coexistence of many users at the same time. This achieved by the use of *home* directories. A typical file system configuration is for each of the users home directories to be within another directory called `home`, which itself is within the `users` directory and the `users` directory is itself within the *root* directory, `/`. A *file path* is the address of the file within the file system and is made up of a list of the directories traversed from the root to the file in question, separated by the symbol `/`. For example in the file system described above, a file name `mytext.txt` in the home directory of `myusername` might have a path of `/storage1/gamma/users/myusername/mytext.txt` (depending on where exactly the systems administrators put your home directory). File paths can be *absolute* or *relative*. The example file path used above (`/users/home/myusername/mytext.txt`) is an absolute path and can be used anywhere on the system as it contains the full address of the file. But if you are the user `myusername` and are currently in your home directory, the absolute path is unnecessary and all that is required is the relative path i.e. `mytext.txt`.

Unix often use file suffixes to indicate the type of file. While this is not necessary for executable files (unlike MS DOS and MS Windows with the `exe` extension), it is mostly used for data and program source files. Some software expects to see particular file suffixes. These include program compilers, image editors, multimedia players, typesetting software such as LaTeXand web browsers. When using text editors with syntax colouring, the file extension identifies the text file's type and will therefore colour the programming code depending on the source code language.

To reduce the amount of typing, a number of shortcuts are available. The tilde symbol (∼) always refers to your home directory file path. For example the absolute file path `/users/home/myusername/mytext.txt` may also be typed as `~/mytext.txt`, wherever you are on the system. Two periods `..` always refer to the directory above your current directory. If you are in the directory `documents` in your home directory, *i.e.* `~/documents` the `..` would refer to your home directory. From this location, you could access the file `homefile.txt` in your home directory with the path `../homefile.txt`. A single period `.` always refers to the current directory.

File redirection and *pipes* are another useful features of the operating system. File redirection refers to the assignment of output data, `standard out (stdout)` from a program to a file. Normally the output stream of data goes to the screen. A program such as `ls`, the file listing command can have its output sent to a file instead of to the screen. For example:

```
ls -l > mylisting.txt
```

Here the output from the detailed file listing is redirected from the screen to a file called `mylisting.txt`. The `>` is the redirection operator. The file is created and then the output is sent to the file. If the file already exists, the new output overwrites the file that was there.

To append data to a file, the `>>` operator is used. If the file doesn't exist, the operating system creates it, otherwise it adds the data to the end of the current file. It is often used for data or error logging. For example:

```
myprogram run2 >> alldataruns.dat
```

The program `myprogram` is executed with the parameter `run2`, maybe the second run of an analysis program, and the output is *appended* to the data file `alldataruns.dat`.

Unix file administration commands

The Unix operating system has a myriad of commands to manipulate and administer files. Here are a few with examples.

- `pwd` Prints out your the current directory as a absolute file path.

- `cd` This is the change directory command. If the command is given on its own, it changes to your your home directory. It is normally followed by a parameter, a filepath separated from the command by a `space`.

  `$ cd` Change to your home directory

  `$ cd documents` Change to the documents directory below your current directory

  `$ cd ..` Change to the directory directly above the current directory

  `$ cd /usr/local/bin` Change to the directory with the absolute path (note the path begins with the root character) given by the parameter.

  `$ cd -` Another short cut. Change to the directory you were last in. Very handy if you were formerly in a directory with a long file path.

- `ls` List a directory's contents. This command has many options that can be applied to provide file listings different sort orders etc.

  `$ ls` Will list all of the files and directories in the current directory sort alphabetically in column on the screen.

  `$ ls -l` Will list all of the files and directories in the current directory as above but showing their file sizes, when they were modified last, who owns the files, and their permissions (more on that later).

  `$ ls -lt` Same as above, but this time sorted by modification date.

  `$ ls -R` List the contents of the current directory and then the contents of all of the directories below.

  `$ ls -a` List *all* files in the directory. Without the `-a` option, `ls` does not list the dot files (files prefixed with a `.`, such as `.bashrc`). The files beginning with a dot are configuration files so are normally hidden for convenience.

- *Regular expressions* Across the Unix environment *regular expressions* are commonly used for text searching, translation and substitution. While regular expressions can be very complex, some of the simpler elements are useful with file administration. The asterisk `*` or *wildcard* will match any number of any character. A question mark `?` will match a single ocurrence of any character. Some examples:

  `$ ls *` As the `*` finds all characters of any length this command will list all files in the current directory.

`$ ls *.txt` This finds all all files in the current directory with the suffix `.txt`.

`$ ls testdata*` This finds all files in the current directory beginning with the letters `testdata`.

`$ ls datarun?` Finds all files in the current directory beginning with `datarun` and ending with a single character, such as `datarun0`, `datarun1` . . . `datarun9` or `datarauna`, `datarunb` . . . `datarunz`

- `mkdir` Create a directory.

  `$ mkdir newdirectory` Creates the directory `newdirectory` in the current directory.

  `$ mkdir documents/newdirectory` Creates the directory `newdirectory` in the directory `documents` below the current directory.

- `rm` Remove or delete a file. The `rm` command *can* cause a lot of damage as it will delete a file without warning or asking for confirmation. It has the ability to delete entire directory trees when used with wildcards and the recursive option. It is best used with the `-i` (interactive) option (see below). *The recovery of deleted files on a Unix system is not a trivial task, necessitates the intervention of a systems administrator, may not actually be possible, depending on when the file was created and deleted.and may take some time. Obviously accidental deletion is best avoided.*

  `$ rm badfile` Delete the file `badfile` without asking for confirmation.

  `$ rm -i badfile` Delete the file `badfile` after prompting the user to confirm the deletion.

  `$ rm -i *.jpg` Delete all JPEG files in this directory after prompting the user to confirm the deletion.

- `rmdir` Remove or delete a directory. Note that the directory *must* be empty before it can be removed.

  `$ rmdir tempdir` Delete the directory `tempdir`.

  `$ rmdir /users/home/myusername/documents/tempdocuments` Delete the directory `tempdocuments`.

- `mv` Move or rename a file. The `mv` command is used to transfer a file from one directory to another. It can also be used rename a file by moving a file from one name to another.

  `$ mv mydoc.txt documents` Move the file `mydoc.txt` from the current directory to the documents directory which is in the current directory.

`$ mv myolddoc.txt mynewdoc.txt` Rename `myolddoc.txt` to `mynewdoc.txt`.

- `cp` Copy a file to a new directory, keeping the original file.

  `$ cp firstcopy.txt secondcopy.txt` Makes a copy of `firstcopy.txt` and calls it `secondcopy.txt`.

  `$ cp importantdoc.txt backupdir` Makes a copy of `importantdoc.txt` and places it in the directory `backupdir`.

Unix data extraction and sorting

Most bioinformatic data files are plain text. These text files can be sequence files such as `fasta` files, or annotation files, for example a `genbank` file. Using Unix commands you are able to do such things as extract lines that contain particular information or sort files on a delimited field. Text data files can be very large, hundreds of megabytes to gigabytes, the standard Unix commands handle these files with ease.

- `less` Print a text document to the screen. The `space` key advances through the file, one screenfull (page) at a time. The `enter` key advances one line at a time. The `f` key advances forward one page, while `b` goes back one page. The `h` key when typed gives many options to use. The `q` key exits. Example useage: `$ less mytext.txt`.

- `head` Print the *first* 10 lines of a text file to the screen. For example `head textfile.txt`. The `head` command is useful if a file is very large (eg `fasta` data file) and you only wish to see the first few lines of the file. If you wish to see a different number of lines use you can use `head -30 textfile.txt` for example. This will show the first 30 lines of the file and `head -5 textfile.txt` will show the first 5 lines.

- `tail` Print the *last* 10 lines of a text file to the screen. For example `tail textfile.txt`. As with the `head` cpmmand, the `tail` command is useful if a file is very large (eg `fasta` data file) and you only wish to see the last few lines of the file. It has the same syntax as `head` with for example `tail -30 textfile.txt` printing the last 30 lines in the file to the screen.

- `grep` Print lines matching a pattern. This is a program which utilises regular expressions to search and filter textual data. The patterns do not have to be complex and with even simple search patterns is an extremely handy utility. Some examples:

  `$ grep apple fruitlist.txt` Searches through the file `fruitlist.txt` and outputs to sceen any lines containing the word `apple`.

`$ grep -i apple fruitlist.txt` The `-i` option makes the search non case sensitive. The words `apple`, `Apple`, or `APPLE` will all be found.

`$ grep app *.txt` Will find all lines containing the string `app` in any `.txt` file in the current directory.

`$ grep '[Ss]mug' document.txt` Search for `Smug` or `smug`

`$ grep '[0-9][0-9]' document.txt` Search for pairs of numeric digits

- `sort` The Unix sort command has a number of options and can be complex to use. It is always best to test your output from sort to confirm that is is sorting the way you wish.

  Some examples using the test file `capitals.txt` printed below:

  ```
  Melbourne
  Brisbane
  Adelaide
  Darwin
  Perth
  Hobart
  Sydney
  ```

  To sort the capitals file, simply use:

  ```
  sort capitals.txt
  ```

  The sorted list will print to the screen. To place the sorted list in a file use the redirection operator `>` (see page 4).

  For example:

  ```
  sort capitals.txt > sorted_capitals.txt
  ```

  To reverse the order of the sort, use the `-r` parameter:

  ```
  sort -r capitals.txt > reverse_sorted_capitals.txt
  ```

  If the data we are sorting is numerical use the `-n` parameter. Again the `-r` can be used to reverse the order of the sort.

  If our capitals text file had numbers as the first field:

```
4.3 Melbourne
1.8 Brisbane
1.6 Adelaide
0.4 Darwin
1.9 Perth
0.8 Hobart
4.5 Sydney
```

The `sort` command always operates on the first space-delimited field in the line. To sort on the numbers use:

```
sort capitals.txt -n > sorted_capitals.txt
```

To reverse sort:

```
sort capitals.txt -rn > sorted_capitals.txt
```

Note that in most cases numbers will appear to sort correctly without using the `-n` parameter. However, this is because string-ordering largely coincides with numeric ordering. However, this is definitely not always case, so the `-n` option is preferred when sorting numbers.

We can also sort on more complex lines of data. For example the contig header lines generated by the `Velvet` genome assembler, have a number of fields seperated by the "_" character. Here is a small sample called `nodes.txt`:

```
>NODE_20_length_586_cov_3.607509
>NODE_21_length_745_cov_5.076510
>NODE_22_length_202_cov_4.297030
>NODE_31_length_237_cov_5.172996
>NODE_36_length_305_cov_4.055738
>NODE_37_length_550_cov_4.149091
>NODE_38_length_24_cov_15.416667
>NODE_42_length_629_cov_4.146264
>NODE_46_length_1163_cov_4.441960
>NODE_59_length_1032_cov_4.525194
>NODE_60_length_145_cov_4.600000
>NODE_61_length_155_cov_5.625806
>NODE_62_length_1583_cov_4.566646
```

The fields are delimited by the "_" and are; ">NODE", the node number,"length", node length, "cov", contig coverage (the number of reads which make up this contig). We can sort on any of the fields by using the `-t` and `-k` parameters.

For example, the file is currently sorted on the node number, if we required the file sorted on the node length use:

```
sort -n -t '_' -k 4  nodes.txt > sorted_lengths.txt
```

The command uses `-n` to sort on numbers, `-t '_'` to specify the field delimiter and `-k 4` to select the field to sort on (in this case the 4th field, length).

To sort on the coverage in reverse order use:

```
sort -rn -t '_' -k 6  nodes.txt > reverse_sorted_cov.txt
```

- `Unix pipes` The Unix pipe command allows you to direct the output from one command immediately into another for processing (the data is *piped* from one command to another). This reduces the need for intermediate files and produces a smooth logical, workflow.

  For example, the `nodes.text` file was produced from the `Velvet` genome assembler's output contig file. From the contig `fasta` file we need to only extract the header lines. As the headers all begin with the ">" character, we can use grep

  ```
  grep ">" config.fa > nodes.txt
  ```

  This produces the `nodes.text` file - a small section is shown above. But we can eliminate the `nodes.text` file and use a pipe to sent is strainght to the sort command.

  ```
  grep ">" config.fa | sort -n -t '_' -k 4  nodes.txt > sorted_lengths.txt
  ```

  To check the top 10 lines of the output, you can use:

  ```
  grep ">" config.fa | sort -n -t '_' -k 4  nodes.txt | head
  ```

  The `head` program will read the data being fed into it (called `standard input`) as if it were reading a file.

  To check the output using the `less` command:

  ```
  grep ">" config.fa | sort -n -t '_' -k 4  nodes.txt | less
  ```

When building a piped workflow such as this, it is good practice to check the output from each section separately with `head`, to confirm that the component commands are working correctly, before stringing it all together with pipes

```
grep ">" config.fa | head
grep ">" config.fa | sort -n -t '_' -k 4  nodes.txt | head
```

This confirms that grep is only finding the header lines, and then the second line is the full piped workflow.

Unix file permissions

As mentioned above, the Unix operating system is designed for multiple users and the file permissions central to this. All users have a home directory where they can store their own files. The default file permissions allow a user to read and write their own files while preventing them from reading and writing other users files. It also protects the myriad of Unix system files from being altered or deleted by persons other than the appropriate system administrator. When the file listing command `ls -l` is entered at the prompt along with the file's dates and sizes is also their permissions. An example listing is below.

```
/path/myusername $ ls -l
drwx------   3 myusername   pgrad       1024 Aug 13  2003 RNA
-rw-r--r--   1 myusername   pgrad     962932 May 22  2003 mal2.fna
-rw-r--r--   1 myusername   pgrad       1872 May 22  2003 genome.cc
-rwxr-xr-x   1 myusername   pgrad       1698 Feb 14 19:41 biotest4.pl
```

The file permissions are the first grouping of letters in the file details listing. There are three types of access to a file; *user*, *group* and *all.* The first letter, `d` indicates whether a file is a directory or not. The character `-` in the first position designates this is a an ordinary file. In this example the only directory is one called `RNA`. The next nine letters are broken into three sets of three. The first three letter set gives the files owner's or *user* permissions. If this is your home directory, these are your file permissions. The letter `r` means that you can read the file, `w` that you can may write to the file, edit it or delete it, and `x` indicates whether the file is executable. When writing programs in a scripting language such as `Python`, generally the program file is only executable when its permissions are set to executable. The second group of the same three letters give the same permission but this time for the *group.* If a number of people are editing the same file, they can all have read/write/execute access to a file while still restricting it from users who are not in the group. The final set of permissions refers to *others* or everyone else who has access to the system. If you have a file which is to be available for all users to read, the read permission must be set for *other.*

To set or change the permissions for a file, the `chmod` command is used. This command takes two parameters, the permissions to be set and the names of the files on which to set them. Abbreviations are used for each of the permission sets with o = owner, g = group and o = other. Wildcards can be used to set the permissions for a number of files in one operation.

- `chmod +x myscript.sh` To make the file `myscript.sh` *executable* by everyone; user, group and other. The executable permission `x` is added to each of the three permission groups.
- `chmod go-w mydata.dat` Takes away the write permission for *writing* from the group and others.
- `chmod g+rw mydata.dat` Gives the group *reading* and *writing* permission.
- `chmod o+x myscript.sh` Make the file *executable* for the owner only.
- `chmod +x *.sh` Make all files with the extension `.sh` *executable* to everyone.

The `chmod` command may also be used with a numerical parameter to set file permissions. The nine letters after the *directory* flag may be represented as three *octal* numbers. The permissions can then be applied by setting the bits for each of the octal permission sets.

```
0 = ---
1 = --x
2 = -w-
3 = -wx
4 = r--
5 = r-x
6 = rw-
7 = rwx
```

Some examples:

- `chmod 644 mydata.dat` gives `rw- r-- r--` Makes the file `mydata.dat` readable, writable by the owner, but only readable to everyone else.
- `chmod 755 myscript.sh` gives `rwx r-x r-x` The file can be edited by the owner only, but read and executed by everyone.
- `chmod 777 myscript.sh` gives `rwx rwx rwx` The file can be edited, read and executed by everyone.

The graphical file editor `gedit`

A good graphical text editor to use on the server is `gedit`. It is installed already on the UNIX servers. Just type `gedit` or `gedit filename` or `gedit &` at the prompt, and an editor window will appear. If it doesn't appear, check behind the `MobaXterm` window, as sometimes it pops up behind this window.

Unexpected errors about not being able to find a file when you run `gedit` can be stopped by typing

`mkdir -p  /.local/share` ¡Enter¿

at the command line. Be sure to include the `'.'` before `local`.

The file editor `nedit` - suitable for the X Windows desktop

When using a terminal in the X Windows desktop environment, the text editor `nedit` is easy to use, and unlike some of the more traditional Unix editors, has common key commands to those from MS Windows.

To start `nedit` with a file to be edited, enter `nedit filename`. If the filename is omitted the editor will display a blank screen. If you append the `&` character to the command *i.e.* `nedit filename &`, `nedit` will launch, but it will run in the background. You will have a commandline cursor ready for another command, along with `nedit`. The program has a menu bar along the top with the familiar File, Edit, Search etc. headings. If you are editing a file and open another, the default action is to add a tab to the top of the window. The new file is under the tab. Most of the key commands are the same as Windows. For example `Ctrl-s` (Ctrl is the `control` key) saves a file, `Ctrl-n` to open a new enpty tab, `Ctrl-c` to copy some highlighted text, `Ctrl-v` to paste it and `Ctrl-f` to search for a word in a file. Of course you can use the menu options too. Under the `Preferences` menu option you can turn on syntax highlighting (great when editing programs) and also line numbers; this helps with locating programming errors (if there is an error in a script, it will give the line number of the line containing the error). Note that the `Help` menu option is on the far right of the menu bar. There is an extensive help system.

Nearly all sequence files are plain text. A program such as nedit will happily open a one million line text file containing a 50 million base genome.

The file editor `nano` - for the Unix terminal

A terminal editor has no graphical user interface features like mouse control or variable fonts, but is still suitable for creating programming and text files. An editor which is found on most Unix machines is `nano`. Start the editor by entering `nano filename`. If the filename is omitted the editor will display a blank screen. Use the arrow keys on the keyboard to move the cursor around the screen. Along the bottom of the screen you will see all of the key commands to perform common tasks. For example to save a file use `Ctrl-o` (write Out) and to read a file `Ctrl-r`. To remove or cut a line of text, place the curson on the line you wish to remove and press `Ctrl-k`. If you move the cursor to another blank line you can paste the same line (uncut) using `Ctrl-u`. To exit the edit use `Ctrl-x`.

R. Hall March 2014
revised, L. Stern Feb 2015

Tasks

Simple tasks to accustom yourself to working with Unix and some of the common commands.

1. Make a new directory in your home directory. For example, `mkdir dir01`
2. Make a second directory in your home directory. Example, `mkdir dir02`
3. Look at a directory listing for your home directory. `ls`
4. Look at a detailed directory listing for your home directory. `ls -l`
5. Start up the text editor nedit by typing `nedit document1.txt &`. This launches the text editor with the file document1.txt ready to be edited. As the file doesn't exist, the editor creates it for you. The `.txt` extension to the file name alerts us to the fact that it is a text file, but is not really essential.
6. Type in a few lines of random type. Treat it as any text editor.
7. Now save your new file and exit.
8. Get a detailed directory listing of the directory. `ls -l`
9. Copy the document to a different location. `cp document1.txt dir1`
10. Get a detailed listing of directory `dir1`. `ls -l dir1`
11. Change directory to `dir02`. `cd dir02`
12. Check where we are now by using the `pwd` command. This stands for *Present Working Directory.* It should confirm that we are in directory `dir02`.
13. Start up the text editor again; this time with the filename `python01.py` as a parameter. `nedit python01.py &`
14. Type in the following lines:

    ```
    #!/usr/local/bin/python

    # print Hello World
    print "Hello World"
    ```

Save the file and exit.

15. Show a detailed directory listing with `ls -l`, for the current directory. Note that the script `python01.py` does not show executable permissions - it is only a text file at the moment.

16. Use `grep -i 'hello' *.py` to find occurrences of the word `hello` regardless of case in all of the `.py` files.

17. Run `grep -i 'hello' *.py > python.txt` again, but this time redirect the input to the file `python.txt`.

18. Type `head python.txt` to see the first few lines of the file `python.txt`.

19. To now delete the file `python.txt`, use `rm python.txt`

20. Use the `chmod` command to convert `python01.py` to an executable file. `chmod +x python01.py`

21. Show a detailed directory listing with `ls -l`. The script is now executable for the owner, the group, and others.

22. Run the script with: `./python01.py`. Note the preceding `./`. In this instance, the full file path must be used to execute the file. As the `.` refers to the current directory these two characters give the required path. If your script won't run (sometimes `/usr/local/bin/python` will not be the correct path), try running `python python01.py`. This invokes the `python` interpretor directly.

23. If all goes well, you should get the output `'Hello World'`.

24. If we now want to delete the file `python01.py` we can use the `rm` command. `rm python01.py`

25. A directory listing will show the file gone. `ls`

26. Return to the directory above; your home directory - with either: `cd ..` or `cd`

27. Now remove the directory we were just in with `rmdir dir02`. If the directory is not empty, the command will fail.

28. And remove `dir01` with `rmdir dir01`

29. Finally delete our first example text file with: `rm document1.txt`