Computational Practical 1 - Introduction to the Command Line and Unix

Module Developers: Dr. Stanford Kwenda, Mr. Collins Kigen and Mr Mishalan Moodley

Table Of Contents

Learning objectives

Learn how to use the Unix command-line.

Logging on/loading the virtual machine

(Instructions on how to do the basic logging-on etc will go here.)

This is where you will do all the practical exercises for this course. You will see a screen which looks a lot like this:

(base) ubuntu@student-1:~$

This is called the command prompt. It shows you the username you have used to connect to the server (in this case 'ubuntu') and the name of the machine (my machine is called 'student-1'), and the directory where you are (in this case, '~' - more about this later). The $ marks the end of the command prompt.

Files and directories

Whenever you connect to the server, you will start in your *home* directory. This is the area of the server where you can put your own files and directories. To see what is in our home directory, type the following into your window and press Enter:

ls

The output will look like this:

```
(base) ubuntu@student-1:~$ ls

anaconda2  data  Desktop ncbi  scripts

(base) ubuntu@student-1:~$
```

This gives you a list of the files and directories in the directory you are in, then puts you back at the command prompt ready to type another command.

The ls command is used to list the contents of any directory, not necessarily the one that you are currently in. Try the following:

```
(base) ubuntu@student-1:~$ ls /etc/perl

CPAN  Net  XML

(base) ubuntu@student-1:~$
```

Looking at directories from within a Unix terminal can often seem confusing. But bear in mind that these directories are exactly the same type of folders that you can see if you use any graphical file browser. From the root level (/) there are usually around

20 directories. You can treat the root directory like any other, e.g. you can list its contents with ls:

(base) ubuntu@student-1:~$ ls /

bin boot cvmfs dev etc galaxy home initrd.img initrd.img.old lib lib64 lost+found media mnt nonexistent opt proc root run sbin snap srv sys tmp usr var vmlinuz vmlinuz.old

(base) ubuntu@student-1:~$

You might see some of these names for files and folders appearing in different colours. Many Unix systems will display files and directories in different colours by default, or colours may be used for special types of files. When you log in to a computer you are working with your files in your home directory, and this is often inside a directory called ‘users’ or ‘home’.

There may be many hundreds of directories on any Unix machine, so how do you know which one you are in? The command pwd will ‘Print the Working Directory’ to show you what directory you are in:

(base) ubuntu@student-1:~$ pwd

/home/ubuntu

When you log in to a Unix computer, you typically start in your home directory. In this example, after we log in, we are placed in a directory called ‘ubuntu’ which itself is a subdirectory of another directory called ‘home’. Conversely, ‘home’ is the parent directory of ‘ubuntu’. The first forward slash that appears in a list of directory names always refers to the top level directory of the file system (known as the root directory). The remaining forward slash (between ‘home’ and ‘ubuntu’) delimits the various parts of the directory hierarchy. If you ever get ‘lost’ in Unix, remember the pwd command.

As you learn Unix you will frequently type commands that don’t seem to work. Most of the time this will be because you are in the wrong directory, so it’s a really good habit to get used to running the pwd command a lot.

Making and changing directories

If we want to make a new directory (e.g. to store some work related data), we can use the mkdir command:

(base) ubuntu@student-1:~$ mkdir learning_unix

(base) ubuntu@student-1:~$ ls

anaconda2 data Desktop galaxy-app galaxy-indices galaxy-tools learning_unix ncbi scripts

We are in the home directory on the computer but we want to work in the new learning_unix directory. To change directories in Unix, we use the cd command:

(base) ubuntu@student-1:~$ cd learning_unix/

(base) ubuntu@student-1:~/learning_unix$

Let's make two new subdirectories and navigate into them:

(base) ubuntu@student-1:~/learning_unix$ mkdir outer

(base) ubuntu@student-1:~/learning_unix$ cd outer

(base) ubuntu@student-1:~/learning_unix/outer$

(base) ubuntu@student-1:~/learning_unix/outer$ mkdir inner

(base) ubuntu@student-1:~/learning_unix/outer$ cd inner/

(base) ubuntu@student-1:~/learning_unix/outer/inner$

We created the two directories in separate steps, but it is possible to use the mkdir command to do this all in one step.

Like most Unix commands, mkdir supports command-line options which let you alter its behavior and functionality. Command-like options are - as the name suggests - optional arguments that are placed after the command name. They often take the form of single letters (following a dash). If we had used the -p option of the mkdir command we could have done this in one step. E.g.

mkdir -p outer/inner

Note the spaces either side of the -p!

Let's change directory to the root directory, and then into our home directory

(base) ubuntu@student-1:~/learning_unix/outer/inner$ cd /

(base) ubuntu@student-1:/$ cd home

(base) ubuntu@student-1:/home$ cd ubuntu

(base) ubuntu@student-1:~$

In this case, we may as well have just changed directory in one go:

cd /home/ubuntu/

The leading / is incredibly important. The following two commands are very different:

cd /home/ubuntu/

cd home/ubuntu/

The first command says 'go the ubuntu directory that is beneath the home directory that is at the top level (the root) of the file system'. There can only be one /home/ubuntu directory on any Unix system.

The second command says go to the ubuntu directory that is beneath the home directory that is located **wherever I am right now**. There can potentially be many home/ubuntu directories on a Unix system.

Learn and understand the difference between these two commands.

Frequently, you will find that you want to go ‘upwards’ one level in the directory hierarchy. Two dots .. are used in Unix to refer to the parent directory of wherever you are. Every directory has a parent except the root level of the computer. Let’s go into the learning_unix directory and then navigate up two levels:

(base) ubuntu@student-1:~$ cd learning_unix/

(base) ubuntu@student-1:~/learning_unix$ cd ..

(base) ubuntu@student-1:~$ cd ..

(base) ubuntu@student-1:/home$

What if you wanted to navigate up two levels in the file system in one go? It’s very simple, just use two sets of the .. operator, separated by a forward slash:

cd ../..

Using cd .. allows us to change directory relative to where we are now. You can also always change to a directory based on its absolute location. E.g. if you are working in the /home/ubuntu/learning_unix directory and you then want to change to the /tmp directory, then you could do either of the following:

(base) ubuntu@student-1:~/learning_unix$ cd ../../../tmp

(base) ubuntu@student-1:~/learning_unix$ cd /tmp

They both achieve the same thing, but the second example requires that you know about the full path from the root level of the computer to your directory of interest (the 'path' is an important concept in Unix). Sometimes it is quicker to change directories using the relative path, and other times it will be quicker to use the absolute path.

Remember that the command prompt shows you the name of the directory that you are currently in, and that when you are in your home directory it shows you a tilde character (~) instead? This is because Unix uses the tilde character as a short-hand way of specifying a home directory.

See what happens when you try the following commands (use the pwd command after each one to confirm the results if necessary):

cd /

cd ~

cd

Hopefully, you should find that cd and cd ~ do the same thing, i.e. they take you back to your home directory (from wherever you were). You will frequently want to jump straight back to your home directory, and typing cd is a very quick way to get there.

The .. operator that we saw earlier can also be used with the ls command, e.g. you can list directories that are 'above' you:

(base) ubuntu@student-1:~/learning_unix$ cd ~/learning_unix/outer/

(base) ubuntu@student-1:~/learning_unix/outer$ ls ../../

anaconda2 data Desktop galaxy-app galaxy-indices galaxy-tools learning_unix ncbi scripts

Time to learn another useful command-line option. If you add the letter 'l' to the ls command it will give you a longer output compared to the default:

(base) ubuntu@student-1:~/learning_unix$ ls -l /home

```
total 24
drwxr-xr-x  2 cloudgene  cloudgene 4096 Apr 10  2018 cloudgene
drwxr-xr-x  2 fuse       users     4096 Mar 12 12:41 fuse
drwxr-xr-x  5 galaxy     users     4096 Jan  2 04:08 galaxy
drwxr-xr-x  3 gvlwebuser webapps   4096 Apr 10  2018 gvlwebuser
drwxr-xr-x  3 ubuntu     ubuntu    4096 May  9  2017 linuxbrew
lrwxrwxrwx  1 root       root        27 Mar 12 12:41 researcher ->
/mnt/galaxy/home/researcher
drwxr-xr-x 23 ubuntu     ubuntu    4096 Mar 20 07:03 ubuntu
```

For each file or directory we now see more information (including file ownership and modification times). The ‘d’ at the start of each line indicates that these are directories. There are many, many different options for the ls command. Try out the following (against any directory of your choice) to see how the output changes.

```
ls -l
ls -R
ls -l -t -r
ls -lh
```

Note that the last example combine multiple options but only use one dash. This is a very common way of specifying multiple command-line options. You may be wondering what some of these options are doing. It’s time to learn about Unix documentation.

Documentation

If every Unix command has so many options, you might be wondering how you find out what they are and what they do.Every Unix command has an associated 'manual' that you can access by using the man command. E.g.

man ls

man man # yes even the man command has a manual page

When you are using the man command, press space to scroll down a page, b to go back a page, or q to quit. You can also use the up and down arrows to scroll a line at a time. The man command is actually using another Unix program, a text viewer called less, which we'll come to later on.

Removing Directories

We now have a few (empty) directories that we should remove. To do this use the rmdir command, this will only remove empty directories so it is quite safe to use. If you want to know more about this command (or any Unix command), then remember that you can just look at its man page.

(base) ubuntu@student-1:~/learning_unix$ cd ~/learning_unix/outer/

(base) ubuntu@student-1:~/learning_unix/outer$ rmdir inner/

(base) ubuntu@student-1:~/learning_unix/outer$ cd ..

(base) ubuntu@student-1:~/learning_unix$ rmdir outer/

(base) ubuntu@student-1:~/learning_unix$ ls

(base) ubuntu@student-1:~/learning_unix$

Note: you have to be outside a directory before you can remove it with rmdir!

Tab completion

Saving keystrokes may not seem important, but the longer that you spend typing in a terminal window, the happier you will be if you can reduce the time you spend at the keyboard. So the best Unix tip to learn early on is that you can tab complete the names of files and programs on most Unix systems. Type enough letters that uniquely identify the name of a file, directory or program and press tab…Unix will do the rest. E.g. if you type 'tou' and then press tab, Unix should autocomplete the word to 'touch' (this is a command which we will learn more about in a minute). In this case, tab completion will occur because there are no other Unix commands that start with 'tou'. If pressing tab doesn't do anything, then you have not have typed enough unique characters. In this case pressing tab twice will show you all possible completions. This trick can save you a LOT of typing!

Navigate to your home directory, and then use the cd command to change to the learning_unix directory. Use tab completion to complete directory name. If there are no other directories starting with 'l' in your home directory, then you should only need to type cd l and then press your tab key.

Tab completion will make your life easier and make you more productive!

Another great time-saver is that Unix stores a list of all the commands that you have typed in each login session. You can access this list by using the history command or more simply by using the up and down arrows to access anything from your history. So if you type a long command but make a mistake, press the up arrow and then you can use the left and right arrows to move the cursor in order to make a change.

Working with files

The following sections will deal with Unix commands that help us to work with files, i.e. copy files to/from places, move files, rename files, remove files, and most importantly, look at files. First, we need to have some files to play with. The Unix command touch will let us create a new, empty file. The touch command does other things too, but for now we just want a couple of files to work with.

(base) ubuntu@student-1:~/learning_unix$ touch heaven.txt

```
(base) ubuntu@student-1:~/learning_unix$ touch earth.txt

(base) ubuntu@student-1:~/learning_unix$ ls

earth.txt  heaven.txt
```

Now, let's assume that we want to move these files to a new directory ('temp'). We will do this using the Unix mv (move) command. Remember to use tab completion:

```
(base) ubuntu@student-1:~/learning_unix$ mkdir temp

(base) ubuntu@student-1:~/learning_unix$ mv heaven.txt temp/

(base) ubuntu@student-1:~/learning_unix$ mv earth.txt temp/

(base) ubuntu@student-1:~/learning_unix$ ls

temp

(base) ubuntu@student-1:~/learning_unix$ ls temp/

earth.txt  heaven.txt
```

For the mv command, we always have to specify a source file (or directory) that we want to move, and then specify a target location. If we had wanted to we could have moved both files in one go by typing any of the following commands:

```
mv *.txt temp/

mv *t temp/

mv *ea* temp/
```

The asterisk * acts as a wild-card character, essentially meaning 'match anything'. The second example works because there are no other files or directories in the directory that end with the letters 't' (if there was, then they would be moved too).

Likewise, the third example works because only those two files contain the letters 'ea' in their names. Using wild-card characters can save you a lot of typing.

The '?' character is also a wild-card but with a slightly different meaning. See if you can work out what it does.

Answer

The '?' matches any *single* character.

In the earlier example, the destination for the mv command was a directory name (temp). So we moved a file from its source location to a target location. Note that the target could have also been a (different) file name, rather than a directory, so mv can also allow us to rename a file. E.g. let's make a new file and move it whilst renaming it at the same time:

```
(base) ubuntu@student-1:~/learning_unix$ touch rags

(base) ubuntu@student-1:~/learning_unix$ ls

rags  temp

(base) ubuntu@student-1:~/learning_unix$ mv rags temp/riches

(base) ubuntu@student-1:~/learning_unix$ ls temp/

earth.txt  heaven.txt  riches
```

In this example we create a new file ('rags') and move it to a new location and in the process change the name (to 'riches'). So mv can rename a file as well as move it. The logical extension of this is using mv to rename a file without moving it (you have to use mv to do this as Unix does not have a separate 'rename' command):

```
(base) ubuntu@student-1:~/learning_unix$ mv temp/riches temp/rags
```

It is important to understand that as long as you have specified a 'source' and a 'target' location when you are moving a file, then it doesn't matter what your current

directory is. You can move or copy things within the same directory or between different directories regardless of whether you are in any of those directories. Moving directories is just like moving files:

(base) ubuntu@student-1:~/learning_unix$ mkdir temp2

(base) ubuntu@student-1:~/learning_unix$ mv temp2 temp

(base) ubuntu@student-1:~/learning_unix$ ls temp/

earth.txt heaven.txt rags temp2

This step moves the temp2 directory inside the temp directory. Try creating a 'temp3' directory inside learning_unix and then cd to /tmp. Can you move temp3 inside temp2 without changing directory? When you are finished, cd back to learning_unix.

Answer

(base) ubuntu@student-1:~/learning_unix$ mkdir temp3

(base) ubuntu@student-1:~/learning_unix$ cd /tmp

(base) ubuntu@student-1:/tmp$ mv ~/learning_unix/temp3 ~/learning_unix/temp/temp2

(base) ubuntu@student-1:/tmp$ cd ~/learning_unix/

(base) ubuntu@student-1:~/learning_unix$ ls temp/temp2/

temp3

Removing files

You've seen how to remove a directory with the rmdir command, but rmdir won't remove directories if they contain any files. So how can we remove the files we have created (inside learning_unix/temp)? In order to do this, we will have to use the rm (remove) command.

Please read the next section VERY carefully. Misuse of the rm command can lead to huge problems!

Potentially, rm is a very dangerous command; if you delete something with rm, you will not get it back! It is possible to delete everything in your home directory (all directories and subdirectories) with rm, that is why it is such a dangerous command.

Let me repeat that last part again. It is possible to delete EVERY file you have ever created with the rm command. Are you scared yet? You should be. Luckily there is a way of making rm a little bit safer. We can use it with the -i command-line option which will ask for confirmation before deleting anything (remember to use tab-completion). Run the following commands, and type "y" (for yes) to indicate you truly intend to delete the files.

```
(base) ubuntu@student-1:~/learning_unix$ cd temp

(base) ubuntu@student-1:~/learning_unix/temp$ ls

earth.txt  heaven.txt  rags  temp2

(base) ubuntu@student-1:~/learning_unix/temp$ rm -i earth.txt heaven.txt rags

rm: remove regular empty file 'earth.txt'? y

rm: remove regular empty file 'heaven.txt'? y

rm: remove regular empty file 'rags'? y
```

We could have simplified this step by using a wild-card (e.g. rm -i *.txt) or we could have made things more complex by removing each file with a separate rm command. Let's finish cleaning up:

(base) ubuntu@student-1:~/learning_unix/temp$ rmdir temp2/temp3/

(base) ubuntu@student-1:~/learning_unix/temp$ rmdir temp2/

(base) ubuntu@student-1:~/learning_unix/temp$ cd ..

(base) ubuntu@student-1:~/learning_unix$ rmdir temp/

Copying files

Copying files with the cp (copy) command is very similar to moving them. Remember to always specify a source and a target location. Let's create a new file and make a copy of it:

(base) ubuntu@student-1:~/learning_unix$ touch file1

(base) ubuntu@student-1:~/learning_unix$ cp file1 file2

(base) ubuntu@student-1:~/learning_unix$ ls

file1 file2

What if we wanted to copy files from a different directory to our current directory? Let's put a file in our home directory (specified by ~ remember) and copy it to the current directory (learning_unix):

(base) ubuntu@student-1:~/learning_unix$ touch ~/file3

(base) ubuntu@student-1:~/learning_unix$ ls ~

anaconda2 data Desktop file3 galaxy-app galaxy-indices galaxy-tools learning_unix ncbi scripts

(base) ubuntu@student-1:~/learning_unix$ cp ~/file3 .

(base) ubuntu@student-1:~/learning_unix$ ls

```
file1  file2  file3
```

This last step introduces another new concept. In Unix, the current directory can be represented by a . (dot) character. You will mostly use this only for copying files to the current directory that you are in. Compare the following:

```
ls
```

```
ls .
```

```
ls ./
```

In this case, using the dot is somewhat pointless because ls will already list the contents of the current directory by default. Also note how the trailing slash is optional. You can use rm to remove the temporary files.

Finally, let's clean up this directory. Note the use of the * wildcard, which allows us to delete all three files at once.

```
(base) ubuntu@student-1:~/learning_unix$ rm file*
```

The cp command also allows us (with the use of a command-line option) to copy entire directories. Use man cp to see how the -R or -r options let you copy a directory recursively.

Looking at files

So far we have covered listing the contents of directories and moving/copying/deleting either files and/or directories. Now we will quickly cover how you can look at files. The less command lets you view (but not edit) text files. We will use the echo command to put some text in a file and then view it:

```
(base) ubuntu@student-1:~/learning_unix$ echo "Call me Ishmael."
```

Call me Ishmael.

(base) ubuntu@student-1:~/learning_unix$ echo "Call me Ishmael." > opening_lines.txt

(base) ubuntu@student-1:~/learning_unix$ ls

opening_lines.txt

(base) ubuntu@student-1:~/learning_unix$ less opening_lines.txt

On its own, echo isn't a very exciting Unix command. It just echoes text back to the screen. But we can redirect that text into an output file by using the > symbol. This allows for something called file redirection.

Careful when using file redirection (>), it will overwrite any existing file of the same name

When you are using less, you can bring up a page of help commands by pressing h, scroll forward a page by pressing space, or go forward or backwards one line at a time by pressing j or k. To exit less, press q (for quit). The less program also does many other useful things (including text searching).

Let's add another line to the file:

(base) ubuntu@student-1:~/learning_unix$ echo "The primroses were over." >> opening_lines.txt

(base) ubuntu@student-1:~/learning_unix$ cat opening_lines.txt

Call me Ishmael.

The primroses were over.

Notice that we use >> and not just >. This operator will append to a file. If we only used >, we would end up overwriting the file. The cat command displays the contents of the file (or files) and then returns you to the command line. Unlike less you have no control on how you view that text (or what you do with it). It is a very simple, but

sometimes useful, command. You can use cat to quickly combine multiple files or, if you wanted to, make a copy of an existing file:

(base) ubuntu@student-1:~/learning_unix$ cat opening_lines.txt > file_copy.txt

(base) ubuntu@student-1:~/learning_unix$ less file_copy.txt

And again, let's clean up the redundant file:

(base) ubuntu@student-1:~/learning_unix$ rm file_copy.txt

Now let's learn how to count the characters, words, and lines in a file.

(base) ubuntu@student-1:~/learning_unix$ ls

opening_lines.txt

(base) ubuntu@student-1:~/learning_unix$ ls -l

total 4

-rw-rw-r-- 1 ubuntu ubuntu 42 Mar 20 08:52 opening_lines.txt

(base) ubuntu@student-1:~/learning_unix$ wc opening_lines.txt

 2 7 42 opening_lines.txt

(base) ubuntu@student-1:~/learning_unix$ wc -l opening_lines.txt

2 opening_lines.txt

The ls -l option shows us a long listing, which includes the size of the file in bytes (in this case '42'). Another way of finding this out is by using Unix's wc command (word count). By default this tells you many lines, words, and characters are in a specified file (or files), but you can use command-line options to give you just one of those statistics (in this case we count lines with wc -l).

What options would you use to count just the words in the file?

Answer

-w counts the words.

Editing files

nano is a lightweight editor installed on most Unix systems. There are many more powerful editors (such as 'emacs' and 'vi'), but these have steep learning curves. nano is very simple. You can edit (or create) files by typing:

(base) ubuntu@student-1:~/learning_unix$ nano opening_lines.txt

You simply type the text you want to include in the file.

The bottom of the nano window shows you a list of simple commands which are all accessible by typing 'Control' plus a letter. E.g. Control + X exits the program, Control + O will 'write out' or 'save' the file.

Use nano to add the following lines to opening_lines.txt (you can copy and paste them into the window using Ctrl+C and right-click), making sure to use the 'write out' command to save (follow the options at the bottom of the screen):

Now is the winter of our discontent.

All children, except one, grow up.

The Galactic Empire was dying.

In a hole in the ground there lived a hobbit.

It was a pleasure to burn.

It was a bright, cold day in April, and the clocks were striking thirteen.

It was love at first sight.

I am an invisible man.

It was the day my grandmother exploded.

When he was nearly thirteen, my brother Jem got his arm badly broken at the elbow.

Marley was dead, to begin with.

Searching files

You will often want to search files to find lines that match a certain pattern. The Unix command grep does this (and much more). The following examples show how you can use grep's command-line options to:

- show lines that match a specified pattern
- ignore case when matching (-i)
- only match whole words (-w)
- show lines that don't match a pattern (-v)
- Use wildcard characters and other patterns to allow for alternatives (*, ., and [])

(base) ubuntu@student-1:~/learning_unix$ grep was opening_lines.txt

The Galactic Empire was dying.

It was a pleasure to burn.

It was a bright, cold day in April, and the clocks were striking thirteen.

It was love at first sight.

It was the day my grandmother exploded.

When he was nearly thirteen, my brother Jem got his arm badly broken at the elbow.

Marley was dead, to begin with.

(base) ubuntu@student-1:~/learning_unix$ grep -v was opening_lines.txt

Call me Ishmael.

The primroses were over.

Now is the winter of our discontent.

All children, except one, grow up.

In a hole in the ground there lived a hobbit.

I am an invisible man.

(base) ubuntu@student-1:~/learning_unix$ grep all opening_lines.txt

Call me Ishmael.

(base) ubuntu@student-1:~/learning_unix$ grep -i all opening_lines.txt

Call me Ishmael.

All children, except one, grow up.

(Note how searching for ‘all’ without any options matches the word ‘call’, but not ‘All’ due to the capital letter.

(base) ubuntu@student-1:~/learning_unix$ grep in opening_lines.txt

Now is the winter of our discontent.

The Galactic Empire was dying.

In a hole in the ground there lived a hobbit.

It was a bright, cold day in April, and the clocks were striking thirteen.

I am an invisible man.

Marley was dead, to begin with.

learner@:learning_unix$ grep -w in opening_lines.txt

In a hole in the ground there lived a hobbit.

It was a bright, cold day in April, and the clocks were striking thirteen.

Here are some more advanced examples to look at if you have time:

Show advanced examples

(base) ubuntu@student-1:~/learning_unix$ grep -w o.. opening_lines.txt

Now is the winter of our discontent.

All children, except one, grow up.

(base) ubuntu@student-1:~/learning_unix$ grep [aeiou]t opening_lines.txt

In a hole in the ground there lived a hobbit.

It was love at first sight.

It was the day my grandmother exploded.

When he was nearly thirteen, my brother Jem got his arm badly broken at the elbow.

Marley was dead, to begin with.

(base) ubuntu@student-1:~/learning_unix$ grep -w -i [aeiou]t opening_lines.txt

It was a pleasure to burn.

It was a bright, cold day in April, and the clocks were striking thirteen.

It was love at first sight.

It was the day my grandmother exploded.

When he was nearly thirteen, my brother Jem got his arm badly broken at the elbow.

Working with columns

Along your “command line adventures”, you will encounter many files that are divided in columns, such as “csv” or “tsv” files.

Fortunately, unix has many tools to handle and manipulate this type of files.

First let’s download ourselves a test file and look at it’s contents:

```
cd

curl https://raw.githubusercontent.com/Blahah/command_line_bootcamp/master/testfiles/grades.txt > grades.txt

less grades.txt
```

curl will download the contents of any URL you provide it and print it to STDOUT. Since we want our test file on the filesystem, we redirect the output of curl to the file “grades.txt” using the > operator you saw previously.

As you can see, this file containing hypothetical grades for hypothetical characters. First of all, one character stands out - “Spock”, as he aces every class. Let’s extract his information:

```
cut -f 5 grades.txt
```

This command provides us with all the rows for column “5” (-f 5), which contains the grades for “Spock” and prints it to STDOUT.

What else stands out here? “Luke” has a value of 150 where the maximum is 100. He’s probably “forcing” that grade, and that’s cheating. Speaking of cheaters, Malcom is a known cheater, and his scores of 50 on everything raise suspicions. Let’s remove both these students from our file.

```
cut -f -2,4-7,9 grades.txt > grades_no_cheaters.txt
```

Ok, there is a lot to take in here. First, the syntax of what the -f argument takes: the - means “everything up to” when used as the first character, but also means “everything between” when used between two other values (it also can mean “everything after” if used as the last character). So this means “keep all the columns up to 2, then the ones between 4 and 7, and then also column 9”.

Note that we are separating values with “,”. The > grades_no_cheaters.txt will redirect the output into a new file.

Ok, so now let’s add back the cheaters as the last columns of our grades file.

```
cut -f 3,8 grades.txt | paste grades_no_cheaters.txt - > sorted_grades.txt
```

We just cut out the columns with the names of the cheaters, and then “piped” them to paste which placed the columns in the end of the grades file. The “-” here means “read from STDIN”, and we could use another file instead, to merge the contents of both files.

There you have it. Now all you have to do is read sorted_grades.txt and figure out what to do with the cheating students.

Combining commands with pipes

One of the most powerful features of Unix is that you can send the output from one command or program to any other command (as long as the second command accepts input of some sort). We do this by using what is known as a pipe. This is

implemented using the '|' character (which is a character which always seems to be on different keys depending on the keyboard that you are using). Think of the pipe as simply connecting two Unix programs. Here's an example using two commands you saw earlier:

```
(base) ubuntu@student-1:~/learning_unix$ grep was opening_lines.txt | wc -c

316
```

The first use of grep searches the specified file for lines matching 'was', then it sends the lines that match through a pipe to the wc program. We use the -c option to just count characters in the matching lines (316).

Here's an example which introduces some new Unix commands:

```
(base) ubuntu@student-1:~/learning_unix$ grep was opening_lines.txt | sort | head -n 3 | wc -c

130
```

This example first sends the output of grep to the Unix sort command. This sorts a file alphanumerically by default. The sorted output is sent to the head command which by default shows the first 10 lines of a file. We use the -n option of this command to only show 3 lines. These 3 lines are then sent to the wc command as before.

Whenever making a long pipe, test each step as you build it!

Some more useful Unix commands

The following examples introduce some other Unix commands. Remember, you can always learn more about these Unix commands from their respective man pages with the man command. Try them out:

View the penultimate 10 lines of a file (using head and tail commands):

tail -n 20 opening_lines.txt | head

Show lines of a file that begin with a capital I (the ^ matches patterns at the start of a line):

grep "^I" opening_lines.txt

Cut out the 3rd column of a tab-delimited text file and sort it to only show unique lines (i.e. remove duplicates):

cut -f 7 grades.txt | sort -u

Note how it also re-orders the lines in a file into alphanumerical order.

Count how many lines in a file contain the words ‘it’ or ‘at’ (-c option of grep counts lines):

grep -c '[ia]t' opening_lines.txt

Try adding the -i option (case-insensitive) to see how this changes the count.

Turn lower-case text into upper-case (using tr command to ‘transliterate’):

tr 'a-z' 'A-Z' < opening_lines.txt

This uses the > operator which is the reverse of the > operator - instead of sending the command output to a file, it reads the file and sends that to the command.

Environment variables

One other use of the echo command is for displaying the contents of something known as environment variables. These contain user-specific or system-wide values that either reflect simple pieces of information (your username), or lists of useful locations on the file system. Some examples:

(base) ubuntu@student-1:~/learning_unix$ echo $USER

ubuntu

(base) ubuntu@student-1:~/learning_unix$ echo $HOME

/home/ubuntu

(base) ubuntu@student-1:~/learning_unix$ echo $PATH

/home/ubuntu/bin:/home/ubuntu/.local/bin:/home/ubuntu/anaconda2/bin:/home/ubuntu/anaconda2/condabin:/home/linuxbrew/.linuxbrew/bin:/home/linuxbrew/.linuxbrew/sbin:/mnt/galaxy/tools/bin:/usr/lib/postgresql/9.5/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin

The last one shows the content of the $PATH environment variable, which displays a colon-separated list of directories that are expected to contain programs that you can run. This includes all of the Unix commands that you have seen so far. These are files that live in directories which are run like programs (e.g. ls is just a special type of file in the /bin directory).

Knowing how to change your $PATH to include custom directories can be necessary sometimes (e.g. if you install some new software in a non-standard location).