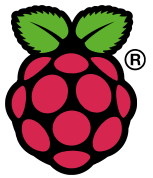
WALES GENE PARK | PARC GENETEG CYMRU

# Beginning UNIX on a Raspberry Pi

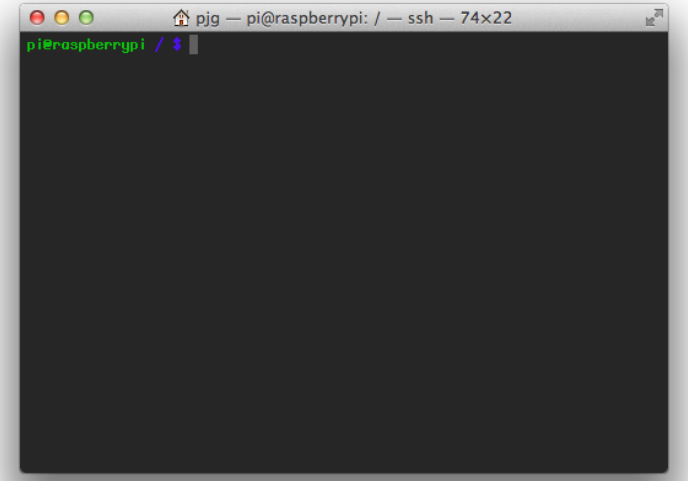


Beta version 1.1 (04 June 2013)

# Contents

# The command line

The command line was the way that any user interacted with their computer until the early 1990's when graphical interfaces like Windows took over as the means by which the typical computer user interacted with their machine. In contrast to the point and click interface of the graphical interface, the command line typically viewed as a flashing cursor waiting to receive your text based instructions.




## Why do people fear the command line?

The main reason is that the command line relies on your memory. In contrast to a graphical interface there is no clues as to what you may want to do next, or any of the settings/parameters that you might want/need to set. As a user you have to have a plan about what you want to do and how about are going to go about achieving it.

Linked to the issues of memory, is the additional challenge of getting help on what you are doing. In a graphical environment, you can have overlaid help pages, pop-up help text or a wizard that steps you through each step. In contrast, a UNIX environment has a manual that is consulted separately in they way you might consult a reference book. Again, we need to know what we are doing in order to get the help we need.

Another issue is one of too many options. If you want to edit a text file on a Windows system you will typically use notepad. On a UNIX machine you get the choice of vi, pico, nano and many others depending on the default configuration. To undertake some text substitution in a workflow we can use sed or tr. With many opinions out there are to the best options, making choices can be overwhelming.

Often a successfully executed command yields absolutely no response, just another command prompt awaiting your input when the task has completed. To the novice used to progress bars, popups and lots of feedback this lack of response can prove unsettling.

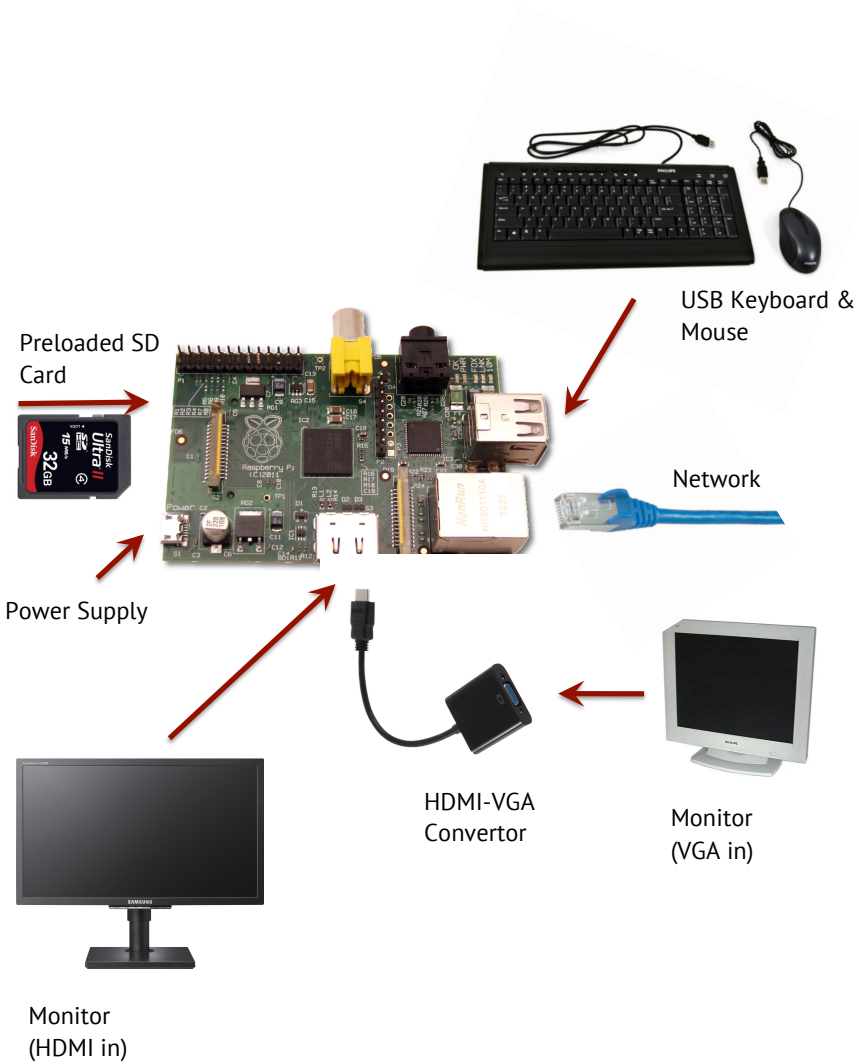
## Why does bioinformatics use the command line?

So why do so many bioinformatics tasks rely on this complicated and unfriendly interface? Bioinformatics often requires the combination of various analysis stages with differing outputs and is often a task of a repetitive nature (e.g. applying a similar analysis with subtle changes to parameters each time).

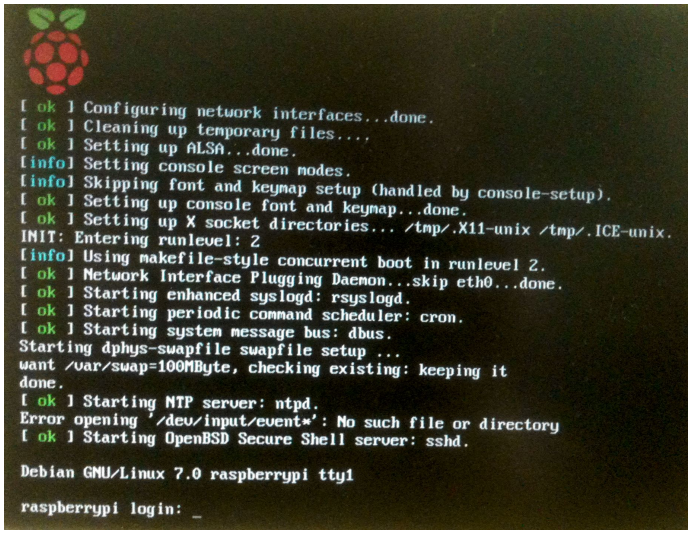
This would be a time consuming and potentially error prone task to undertake in a point and click environment, but programs with a command line interface can overcome these issues and help with the easy automation of your analysis task.

The command line does come with a few discomforts in the early stages, but the benefits once you've overcome this early hump are huge and really do outlay the negatives. After all, not many racing drivers rely on a automatic gearbox!

## Raspberry Pi Setup




## Getting Logged in




Username:

```
pi
```

Password:

```
raspberry
```

*Note your password isn't displayed when typed in.*

Once logged in your command prompt:

```
pi@raspberrypi ~ $
```

You can change your password using the command:

```
passwd
```

## Formatting conventions

In these training materials we will use a few conventions that will hopefully make determining the difference between input, output and options clear.

Commands that you can enter on the command line will we in red and look like this:

```
runme -on thisfile
```

The output returned from a command will be inverted from the way it is show on screen and be black text on a white background:

```
runme... execution successful.
thisfile has been output into thatfile
```

Summary information about a command and any useful flags you may wish to pass to a command are presented as blue text on a grey background:

```
runme          Converts thisfile to thatfile
```

```
-on            Defines the file to convert
```

## UNIX commands

As we've previously pointed out, UNIX relies on your memory to recall which command you need to run to undertake a particular task. However to the novice arrays of commands with names like cd, rm, ls, cp, wc or even cat and man provides absolutely no help to your memory recall.

Just like the evolution of text speak to aid the speed and ease of sending SMS messages, the sending the creators of UNIX were equally creative and lazy and reduced the names of their commands to the fewest numbers of letters, providing ultra-concise names for the most used commands.

In learning these key UNIX commands you may wish to not only learn the two or three character command name, but also the function it is trying to achieve, which will most certainly help in you remembering both the names and functions of these core functions.

For example:

If I want to alter the directory I am looking at,

I need to **C**hange **d**irectory,

Using the **cd** command.

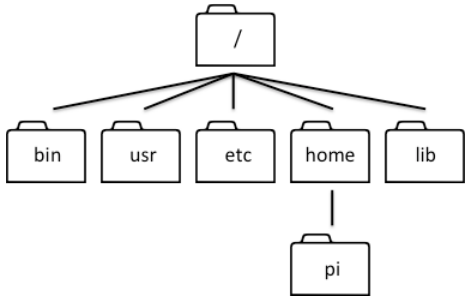
If I want to get a summary of the number of words, lines and characters in a particular file,

I need a **W**ord **C**ount,

Which will use the **wc** command

# Navigating the file system

Just like any other operating system, UNIX operating system keeps things tidy and ordered by storing files in a series of different directories. These directories all have standard uses, but the one we are going to pay attention to is home. Similar to the 'My Documents' directory on a windows system, each user has a home directory where they can store their personal files.




## Home sweet home!

Having logged in using the username 'pi', you will first be placed into your home directory – **'/home/pi'**. We can check this by entering the pwd command (**p**rint **w**orking **d**irectory) that will display our current location.

```
pwd
```

The system will reply with a line indicating your current location in the filesystem.

```
/home/pi
```

We also get a clue about our location by looking at the first bit of the command prompt. The tilde (~) lets us know we are in our home directory.

```
pi@raspberrypi ~ $
```

## Moving around

Changing directory is achieved using the cd (**c**hange **d**irectory) command. So to change into a directory on your SD card called '/home/pi/files' we can just issue the command:

```
cd /home/pi/files
```

We can check this has worked using pwd, or by looking at the command prompt:

```
pwd
```

```
/home/pi/files
pi@raspberrypi ~/files $
```

Alternatively we can use the tilde abbreviation for our home directory.

```
cd ~/files
```

And again we can check our location with pwd.

```
/home/pi/files
```

## Relative and absolute paths

As well as specifying the full path when referring to files or changing directories, we can use relative paths that reduce the length of command we need to execute.  A single dot refers to the current directory, with a double dot referring to the parent directory.

If we first move back to our home directory using one of the following:

```
cd
```

or

```
cd ~
```

or

```
cd /home/pi
```

We can then move into the **'files'** directory using one of the following:

```
cd files
```

or

```
cd ./files
```

or

```
cd ~/files
```

To move back to our home directory we can use

```
cd ..
```

or

```
cd ../
```

Finally, showing off we can nest these commands together making something way more complex than required at the moment, but forming a useful way of thinking about moving around directories.

To move into our files directory we can issue a command which takes us home, then into the files directory referenced from that location.

```
cd ~/./files
```

To move back again we can head up two levels into the home directory, before heading down into the pi directory again:

```
cd ../../pi
```

## Summary of navigation commands

```
pwd             Print the working directory
```

```
cd              Change directory
```

```
.               Current directory
```

```
..              Parent directory (one up)
```

```
~               Home directory
```

## Listing Files

We can view a list of files in a directory outputted in a variety of styles and with different formatting and sorting. To start we will move to the /home/pi/files directory which has a number of example files in it for our explorations.

```
cd ~/files
```

To view the files in a directory, just issues the ls (list) command:

```
ls
```

You will see an alphabetical list of the files in the directory which will wrap across your screen in a tabular format depending on the width and number of results found.

```
blank1  blank2  blank3  blank4  blank5  file1  file2  file3  file4  file5  text1
text2  text3  text4  text5
```

An alphabetical sort is useful, but sometimes we might want to check the most recently created files. We can do this using the –t flag:

```
ls –t
```

With our example data, this will return the files in reverse order, as they were originally created in alphabetical order.

```
text5  text4  text3  text2  text1  file5  file4  file3  file2  file1  blank5
blank4  blank3  blank2  blank1
```

If we want to list all of the files in a particular directory we can append the filename to the ls command and get a list of files elsewhere. E.g. to get a list of files in our home directory we can enter:

```
ls ~
```

Sometimes looking in just a single directory doesn't provide all of the information we want. If we want to find all files and directories that have been created as child directories of our home directory we can include the -R (recursive) flag:

```
ls –R ~
```

The ls command has a few other useful tricks to help in quickly drawing your attention to the information you are after. Using the -F flag we can append a / character to each directory and a * to any executable file (script or program).

```
ls –F ~
```

Another flag which is occasionally useful is the –a flag. The file names of hidden files in UNIX are prefixed with a full stop. The –a (all) flag lists all files including hidden ones. If we have a look at all files in our home directory:

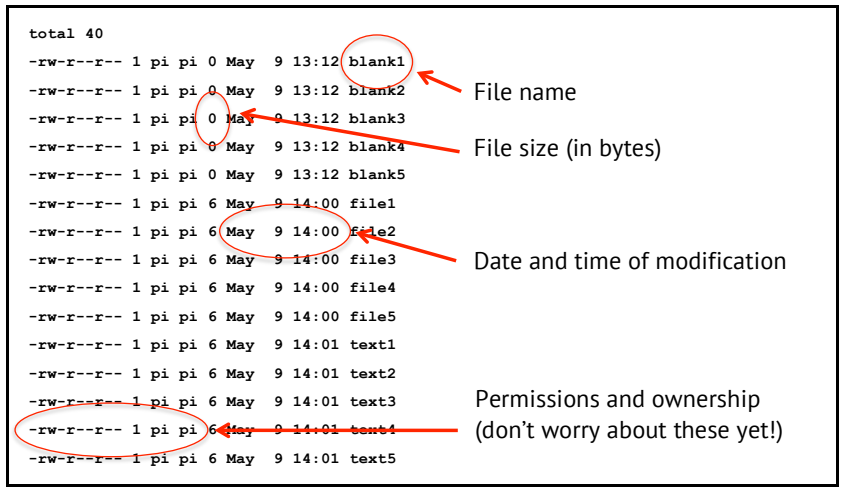
```
ls -a ~
```

You will see a few extra files including:

```
.      ..     .profile
```

The basic ls list is useful for many purposes (e.g. checking a file name, or that it has been created), but sometimes we need a bit more information about a file beyond just its name. The –l (long) flag returns a lot of extra information about a file including the files size, the date and time and modification and information about ownership and permissions to access the file.

```
ls –l
```




At the moment, just ignore the information about ownership and permissions. Whilst they are a fundamental concept in UNIX (and the cause of many headaches to system administrators) they are a bit complex to worry about at this point in your learning.

One final and very useful flag to the ls command is –h. If we combine this flag with the one for a long listing, the file size that is returned is listed in a more human readable format (bytes, kilobytes, megabytes, gigabytes).

```
ls –lh
```

Finally, we should point out that all of the above flags can be comined together in one string to return a listing according to your desires, for example:

```
ls –ltahFR ~
```

## Summary of ls options

| | |
|---|---|
| `-l` | Long listing showing permissions, ownership, file size, modification date and time |
| `-t` | Sort in reverse chronological order |
| `-a` | Show all files (including hidden ones) |
| `-h` | Easier readable file sizes 1K instead of 1024 bytes |
| `-R` | Recursively list all files and directories |
| `-F` | Append to directories and * to executables |

## File information

The file command can make a guess about a file's content and return a description about the identified file type:

```
file blank1
```

```
blank1: empty
```

```
file file1
```

```
file1: ASCII text
```

You can include the names of more than one file in a space separated list files in order to test multiple files in a single command:

```
file blank1 file1 text1
```

```
blank1: empty
file1:  ASCII text
text1:  ASCII text
```

Using * as the filename will return a list for all files in the working directory:

```
file *
```

```
blank1: empty
blank2: empty
blank3: empty
blank4: empty
blank5: empty
file1:  ASCII text
file2:  ASCII text
file3:  ASCII text
file4:  ASCII text
file5:  ASCII text
text1:  ASCII text
text2:  ASCII text
text3:  ASCII text
text4:  ASCII text
text5:  ASCII text
```

## Wildcards

Sometimes we just don't know what we want! Wildcards can be thought of as a way of searching multiple options at the same time and returning all the results together. So if we want to search for all files in our /home/pi/files directory that start with f, we can just type:

```
ls f*
```

```
file1  file2  file3  file4  file5
```

The wildcard * has a specific meaning of nothing or a string of characters of any length. So a search for f* would also return a file called f if it existed in our directory. Likewise a file called floccinaucinihilipilification (the action or habit of estimating something as worthless) would be returned if present.

To search for all files which have a 1 as their final character we can use:

```
ls *1
```

```
blank1  file1  text1
```

Wildcards work with many different programs that can accept multiple files in their input, so for example we can check out the file types of all files ending with a 1 using:

```
file *1
```

```
blank1: empty
file1:  ASCII text
text1:  ASCII text
```

If we want to search for files with the prefix of text, then we can use the ? wildcard which will match any single character.

```
ls text?
```

```
text1  text2  text3  text4  text5
```

To search for files that have 4 characters followed by the number 1 we can use:

```
ls ????1
```

```
file1  text1
```

Finally using brackets we can specify specific matches or ranges. So to find all files with a prefix of file and an ending between 1 and 3:

```
ls file[1-3]
```

```
file1  file2  file3
```

Multiple options can be specified with commas within the brackets (e.g. [1-3,7-9]), include ranges of letters (e.g. [a-z] or [A-Z]), or be combined to specifiy for example just letters and numbers [a-z,A-Z,0-9].

# Creating / moving and copying files

Having spend time treating your Raspberry Pi like a museum and just looking a files as if they are precious exhibits, it now time to start getting creative and looking at how we can create, move, copy and delete files and directories.

## Files

To start we will create a blank file using the touch command (this actually updates the modification time on a file, but if a file doesn't exist creates one first).

```
touch myfile1
```

If we now want to make a copy of this file, we just use the cp (copy) command:

```
cp myfile1 myfile_copy
```

To rename a file, we just use the mv (move) command:

```
mv myfile_copy myfile2
```

And finally to delete a file we just use the rm (remove) command:

```
rm myfile2
```

## Directories

Creating directories is slightly different to files and uses the mkdir (make directory) command:

```
mkdir junkdir
```

If we try removing this directory using the rm (remove) command we get an error:

```
rm: cannot remove `junkdir': Is a directory
```

So need to use the rmdir (remove directory) command:

```
rmdir junkdir
```

## Summary of commands

```
rm              Delete (remove) a file
```

```
mv              Move a file
```

```
cp              Copy a file
```

```
mkdir           Create (make) a directory
```

```
rmdir           Delete (remove) a directory
```

# Making life easier (or being lazy!)

## Tab competition

Tab completion is a function that helps fill in commands, directories and filenames as far as the system is able. By partially typing a command and then pressing tab, the system tries to fill in the rest for you:

```
cd /h<tab>
```

```
cd /home/
```

## History

Sometimes you will want to repeat a command, or reissue a previously executed command with minor modifications. The system keeps a history of your commands that can be viewed using the history command:

```
history
```

```
1     cd /home
2     history
```

This can be useful to create a backup of any work or scripts that you have undertaken. However, when you are actually working pressing the **up** and **down** arrows when on the command line scrolls through this list of your history

# Getting help

## On the command line

There is an almost universal convention on UNIX systems that issuing a command followed by --help will bring up a summary of the key information needed to use a particular command. Sometimes this is just a few lines:

```
mkdir --help
```

And sometimes a few pages:

```
ls --help
```

The core UNIX commands will adhere to this standard, but not all developers of other applications adhere to this standard. If the --help command doesn't bring up any usage information try using the h or -? flags as alternatives. If all else fails, just issuing the command on it's own often brings up some usage information.

## Manuals

Most commands also have a larger body of text that is installed into the operating systems manual database. This can be accessed using the man command. E.g.:

```
man ls
```

You can scroll though this information using the **up** and **down** arrows, use **spacebar** to advance page by page and press **q** to exit.

## File and navigation exercises

1. Create a directory called outputs

2. Create blank files named blank.txt, blank2.txt, blank3.txt in that directory

3. List the files in various formats (normal, detailed, chronological).

4. Rename blank.txt blank1.txt

5. Rename the directory analysis_outputs

6. Delete the contents of the directory

7. Delete the analysis_ouputs directory

8. Display a list of the commands you have just entered.

# Viewing text

UNIX has a number of commands that are useful to look at the contents of text files. Keeping in with the theme of Pis, we are going to use a series of commands used on text files to explore the lyrics of Don McLean's classic American Pie to explore these commands. The file is in your home directory.

To print the contents of a file to screen we can just use the **cat** (catenate) command:

```
cat americanpie.txt
```

For larger files, it can be useful to pause the display after each page of information using the **more** command. Pressing the space bar advances forwards to the next page:

```
more americanpie.txt
```

Pausing after each page can be useful, but sometimes the ability to scroll through a file is what you need to do. In an attempt at being comical, the standard command to achieve this is called less (because less is more!). Just like the man command, you can use the **up** and **down** arrows to scroll through a file and **q** to exit back to the command line.

```
less americanpie.txt
```

Another two very useful commands for looking at text files are **head** and **tail**. As the names may suggest these return a number of lines from either the top or bottom of a file. Typical uses for these commands include looking at the bottom of any log files (either system logs or analysis logs), checking the output from any analysis or command has completed correctly and for checking the file format of a dataset.

By default **head** and **tail** return the first or last 10 lines of a file:

```
head americanpie.txt
```

```
tail americanpie.txt
```

Returning the top 10 lines is a useful default for many purposes, but sometimes you may wish to be more specific in the number of lines you return. This can be achieved in two different ways, a formal way defining a parameter (-n) and then it's value, the second a lazier way just using the number of lines as your flag to the command.

```
head –n 2 americanpie.txt
```

or

```
head –2 americanpie.txt
```

## Summary of text viewing commands

```
cat             Catenates and displays files
```

```
more            Displays a file a page at a time
```

```
less            Scroll forwards and backwards through a file
```

```
head            Display the top line of a file
```

```
tail            Display the bottom lines of a file
```

# Piping and redirection

Piping is a very important concept in UNIX. Put simply it allows you to send the output of a command/program directly into the input for another one. This allows the creation of chains of commands which allow some quite powerful manipulation of data files without the need to resort to a programming based solution.

Linked to piping is the concept of redirection. By default, the output of many commands is printed onto the screen. By using redirection we can send this output to a file, either writing/overwriting a file or appending the output to an existing file.

## Piping

Piping is achieved using the | syntax. As an example we can string together 3 different commands to pull out the 2 classic lines from our American Pie lyrics.

```
cat americanpie.txt | tail -n 4 | head -n 2
```

Working through the logic of what we have done. The command cat displays the file, which is then piped into tail to return the final 4 lines. These lines are then piped into head which takes the top 2 lines (of the final 4 we already selected) and returns these to the display.

## Redirection

If we want to save the results of our previous command to a file, we can just append a > character followed by our desired filename.

```
cat americanpie.txt | tail -n 4 | head -n 2 > junk.txt
```

Occasionally we might wish to redirect our output to file, but also have it displayed on screen as well to check the contents. There are a number of ways to achieve this in a single line command. Using the ; character, we can string together a number of lines of code which are executed sequentially:

```
head americanpie.txt > junk.txt; cat junk.txt
```

Alternatively (and more neatly), we can use the tee command to fork the output stream to both the display and a file:

```
americanpie.txt | tee junk.txt
```

Finally we may wish to append to data to a file. For instance we can create an output log using the date command and the >> append syntax:

```
date > junk.txt
```

```
head americanpie.txt >> junk.txt
```

## Summary of piping and redirection commands

```
|              Pipe the output of one command to another
```

```
>              Redirect the standard output to a file
```

```
>>             Append the standard output to a file
```

```
tee            Fork the output to both screen and a file
```

## Redirection exercises

1. Using head and tail display the first chorus of our lyrics.

2. Output the first chorus to a file called chorus.txt

3. Repeat this command, but also make the output appear on the screen.

4. Repeat the above, but put a title on the chorus “American Pie chorus” and output the file to chorus2.txt

## Searching within files

The grep command is a simple way of looking for data within a file and returning lines that match your search term. This search term can either be an exact match or can include wildcards and other modifiers to the search term to include a variety of matches.

If we want to find out how many times the word Chevy appears in the lyrics of American Pie, we can use the command:

```
grep Chevy americanpie.txt
```

This will return the 7 lines of our file that contain the word Chevy. In its basic form, this is an exact match (case specific) search. So a search for chevy will return no results.

```
grep chevy americanpie.txt
```

We can modify the command to be case insensitive using the –i flag which will again return us our 7 lines of lyric.

```
grep -i chevy americanpie.txt
```

All of this talk of lyrics now may just get you in the mood for some ‘singin’. We can check how much ‘signin’ is occurring using the command:

```
grep -i singin americanpie.txt
```

But this returns all sort of ‘singin’. Lets say we are only interested in the singing about our impending demise we need to modify our search to only include ‘singin’ at the start of a line. This is achieved by use of the circumflex (Chinese hat) character.

```
grep -i ^singin americanpie.txt
```

If we undertake a search for the word ‘down’, we get two results:

```
grep -i down americanpie.txt
```

If we just want to return the line concerning the gaze of the king, we can just append a $ character which indicates the end of a line:

```
grep -i down$ americanpie.txt
```

All these lyrics and ‘singin’ can only lead to one thing, a tapping foot and an sudden interest in dance and dancing. We can undertake a search for lines about dance and/or dancing using the * wildcard character, which you will recall searches for zero or more string characters:

```
grep -i danc* americanpie.txt
```

Finally we might have an interest in lines which are either dry or contain whiskey. To undertake an either or search we need put the grep command into a different mode using the –P flag (which enables Perl based searching), bound our search in apostrophe characters and finally use the | character to separate our search terms.

```
grep -P ‘dry|whiskey' americanpie.txt
```

## Exercises using grep

1. How many times does the jester appear in our lyrics?

Answer:

Command:

2. How many times does Jack appear at the start of a line?

Answer:

Command:

3. What would this command look like if you made it case insensitive?

4. How many times does dance end at the end of a line?

Answer:

Command:

5. Combining two grep searches, how many times were ‘we’ singing ‘bye-bye’ to Miss American Pie?

Answer:

Command:

6. How many times were people saying bye-bye?

Answer:

Command:

## Making your output count

The **wc** (word count) command can be used to sum various bits of information about the searches we were undertaking using grep.

If we pipe the output of grep into **wc**, we are returned a line containing 3 tab separated numbers.

```
grep Chevy americanpie.txt | wc
```

```
      7      77     358
```

The first number is the number of lines, the second the number of words and the third the total number of characters returned. This is a useful summary, but the individual numbers can be returned by adding an additional flag to the **wc** command.

To return the number of lines we can add the **–l** (lines) flag. To return the number of words, use the **–w** (words) flag. And for the number of characters use the **–c** (characters) flag.

```
grep Chevy americanpie.txt | wc –l
```

```
grep Chevy americanpie.txt | wc –w
```

```
grep Chevy americanpie.txt | wc –c
```

## Sort

The sort command will sort the contents of a file (or redirected output from another command) using the entire line as the sort key.

```
head americanpie.txt | sort
```

Addition of the –r flag will reverse the sort order:

```
head americanpie.txt | sort –r
```

By default a the sort is done character by character along the line, which doesn't always make the most sense when sorting numbers:

```
sort numbers.txt
```

```
10
11
9
```

The addition of the –n flag makes the program sort in numerical order:

```
sort –n numbers.txt
```

```
9
10
11
```

## Unique

The uniq command compares adjacent lines in a file and collapses identical lines into a single line. When combined with the sort command it is a way of identifying the unique lines in a file:

```
grep Chevy americanpie.txt | uniq
```

```
Drove my Chevy to the levee, but the levee was dry
```

Applied to our American Pie lyrics, we can see that the uniq command has no effect at all on the output compared to a simple cat display:

```
uniq americanpie.txt
```

But combination with sort, will output the unique lines of our lyrics:

```
sort americanpie.txt | uniq
```

A few extra tricks allow us to return only repeated lines using the –d flag:

```
sort americanpie.txt | uniq -d
```

And by combining the command with the –c (count) flag and an additional numerical sort, we can get a table of the number of times each line of lyrics is repeated:

```
sort americanpie.txt | uniq -c | sort -n
```

## Exercises using wc, sort and uniq

1. How many files are there in our /home/pi/files directory?

Answer:

Command:

2. How many unique lines are in the song American Pie?

Answer:

Command:

3. Can you return a table of repeated lyrics with counts of their frequency, sorted in ascending order of duplication?

## Knowledge check questions

1. How would you change your password?

2. What does the command cd ~ do?

3. How can you change from /home/pi/files to /home/pi/newfiles using a relative path?

4. What command will list all text files (.txt) in a directory showing their file sizes in an easily readable form?

5. What command will show you the first 10 lines of a file?

6. What about the first 25 lines?

7. How about the last 12?

8. How can you find out how to use a new command?

9. What command file will rename file1.txt to file2.txt?

10. How can you move /home/pi/files/file1/txt to the /home/pi directory ?

11. How can you delete all text (.txt) files in a directory?

12. How about deleting all files that have the same format (file1.txt, files2.txt etc)?

13. How can you list all the lines in a file which contain the word raspberry or Raspberry?

14. How can you count the number of lines in a file that contain the word pi?

15. What command will append the output from a program called analyse to a file called output.txt?

16. How can you create a new directory called newdir?

17. How can you count the number of files in your home directory with a .txt suffix?

18. How can you scroll though your output.txt file?

19. How can you find out what all the files in your directory may contain?

20. How can you redirect the output from a program called analysis to both a file called output.txt and also show it on screen?

## Answers to file and navigation exercises

1. Create a directory called outputs

```
mkdir outputs
```

2. Create blank files named blank.txt, blank2.txt, blank3.txt in that directory

```
cd outputs
```

```
touch blank1.txt
```

```
touch blank2.txt
```

```
touch blank3.txt
```

3. List the files in various formats (normal, detailed, chronological).

```
ls
```

```
ls –l
```

```
ls -lt
```

4. Rename blank.txt blank1.txt

```
mv blank.txt blank1.txt
```

5. Rename the directory analysis_outputs

```
cd ..
```

```
mv outputs analysis_outputs
```

6. Delete the contents of the directory

```
rm ./outputs/*
```

7. Delete the analysis_ouputs directory

```
rmdir analysis_outputs
```

8. Display a list of the commands you have just entered.

```
history
```

## Answers to redirection exercises

1. Using head and tail display the first chorus of our lyrics.

```
head -n 21 americanpie.txt | tail -n 5
```

2.. Output the first chorus to a file called chorus.txt

```
head -n 21 americanpie.txt | tail -n 5 > chorus.txt
```

3. Repeat this command, but also make the output appear on the screen.

```
head -n 21 americanpie.txt | tail -n 5 | tee chorus.txt
```

4. Repeat the above, but put a title on the chorus “American Pie chorus” and output the file to chorus2.txt

```
echo “american pie chorus” > chorus2.txt
```

```
head -n 21 americanpie.txt | tail -n 5 >> chorus2.txt
```

## Answers to exercises using grep

1. How many times does the jester appear in our lyrics?

Answer:

```
3
```

Command:

```
grep jester americanpie.txt
```

2. How many times does Jack appear at the start of a line?

Answer:

```
1
```

Command:

```
grep ^Jack americanpie.txt
```

3. What would this command look like you made it case insensitive?

```
grep –i ^Jack americanpie.txt
```

4. How many times does dance end at the end of a line?

Answer:

```
2
```

Command:

```
grep dance$ americanpie.txt
```

5. Combining two grep searches how many times were 'we' singing 'bye-bye' to Miss American Pie?

Answer:

```
2
```

Command:

```
grep bye-bye americanpie.txt | grep ^We
```

6. How many times were people saying bye-bye?

Answer:

```
7
```

Command:

```
grep bye-bye americanpie.txt
```

## Exercises using wc, sort and uniq

1. How many files are there in our /home/pi/files directory?

Answer:

```
15
```

Command:

```
ls ~/files | wc -l
```

2. How many unique lines are in the song American Pie?

Answer:

```
94
```

Command:

```
sort americanpie.txt | uniq | wc
```

3. Can you return a table of repeated lyrics with counts of their frequency, sorted in ascending order of duplication?

```
sort americanpie.txt | uniq -c | sort -n | grep -v 1
```

## Knowledge check answers

1. How would you change your password?

```
passwd
```

2. What does the command cd ~ do?

```
Changes your working directory to your home directory.
```

3. How can you change from /home/pi/files to /home/pi/newfiles using a relative path?

```
cd ../newfiles
```

4. What command will list all text files (.txt) in a directory showing their file sizes in an easily readable form?

```
ls –lh *.txt
```

5. What command will show you the first 10 lines of a file?

```
head file.txt
```

6. What about the first 25 lines?

```
head –n 25 file.txt
```

7. How about the last 12?

```
tail –n 12 file.txt
```

8. How can you find out how to use a new command?

```
command --help
man command
```

9. What command file will rename file1.txt to file2.txt?

```
mv file1.txt file2.txt
```

10. How can you move /home/pi/files/file1/txt to the /home/pi directory ?

```
mv ~/files/file1.txt ~/file1.txt
```

11. How can you delete all text (.txt) files in a directory?

```
rm *.txt
```

12. How about deleting all files that have the same format (file1.txt, files2.txt etc)?

```
rm file*.txt
```

13. How can you list all the lines in a file which contain the word raspberry or Raspberry?

```
grep -i raspberry file.txt
```

14. How can you count the number of lines in a file that contain the word pi?

```
grep pi file.txt | wc -l
```

15. What command will append the output from a program called analyse to a file called output.txt?

```
analyse >> output.txt
```

16. How can you create a new directory called newdir?

```
mkdir newdir
```

17. How can you count up the number of files in your home directory with a .txt suffix?

```
ls *.txt | wc -l
```

18. How can you scroll though your output.txt file?

```
less output.txt
```

19. How can you find out what all the files in your directory may contain?

```
file *
```

20. How can you redirect the output from a program called analysis to both a file called output.txt and also show it on screen?

```
analysis | tee output.txt
```