Raspberry Linux Tutorial

Most people interact with a computer with a graphical OS interface – with a mouse and windows – where you click, drag and sometimes type stuff into boxes. Most computer with Microsoft windows are managed that way although some steps require a command window. Android phones are the same way, point and click although you can get a command line interface.

Linux systems are still managed through the command line. To get to the command line we bring up a shell or terminal window.

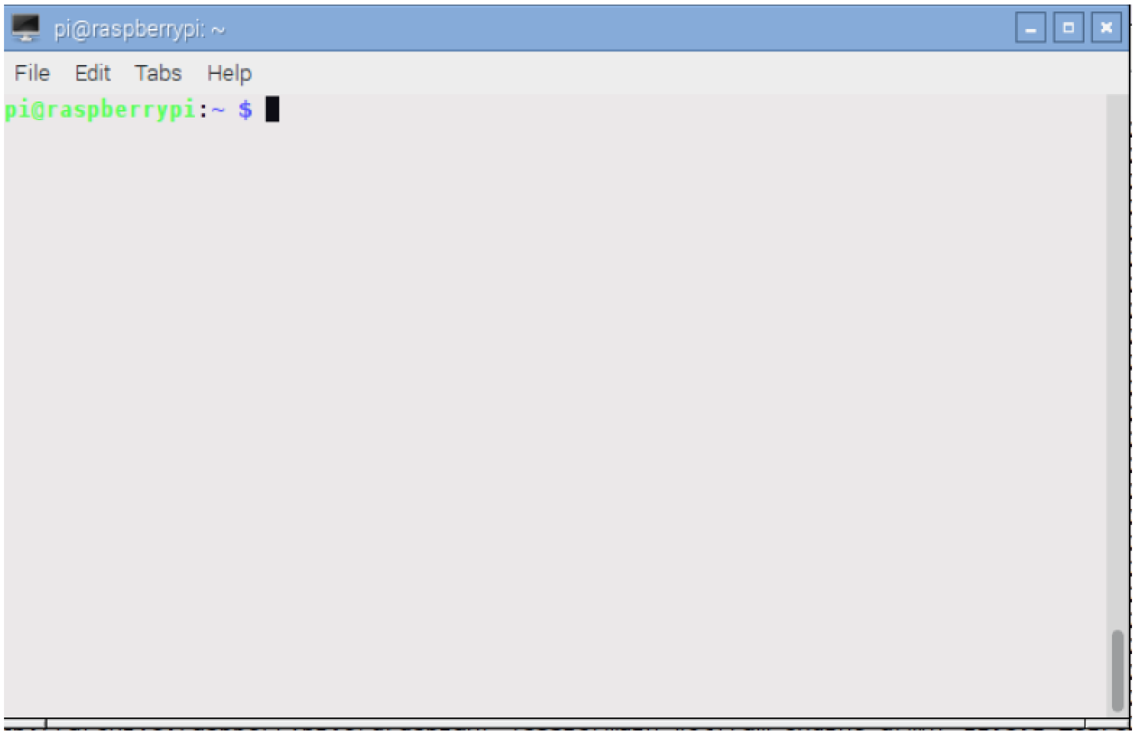
1. The Basics

The terminal window

The terminal window goes back to the 60's and 70's when you connected to a computer system through a teletype terminal – a typewriter if you will. Eventually they moved to simple terminals with a keyboard and monitor but you were limited to 24 lines of 80 characters per line. All linux systems have a terminal window available.

On your raspberry PI it's on the top menu and looks like one of these

This is also found by clicking on menu->accessories and looking for the icon with the word "Terminal" next to it. Click on the icon and this should appear:



```
pi@raspberrypi:~ $
```

This is also called the linux shell. It's called a shell because you, the user, interact with the computer system through a "shell" – the protective outer cover for the operating system. Those green and blue letters are the shell prompt. It tells me that this shell belongs to the user named pi on the computer named raspberrypi. The ":" character is just for separation. The ~ character tells me what directory the shell is in at the moment. We'll explain that one in a bit. The $ character tells me that me, the user, can start typing commands.

You can change colors and text sizes by clicking on edit->preferences and playing with the fonts and text colors.

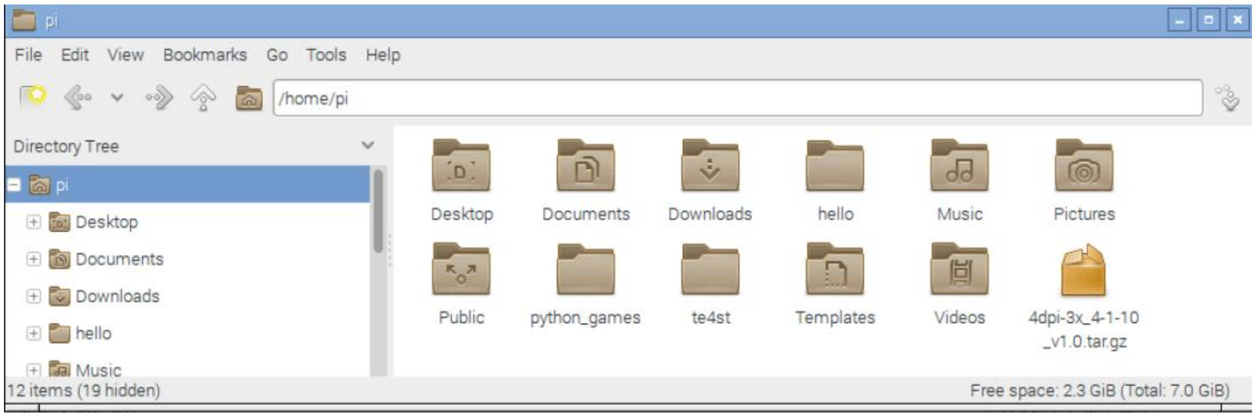
In this tutorial, I'll rarely show you the entire window and instead will just post the commands that you type like this:

```
pi@raspberrypi:~ $ <some commands here>
```

You don't have to type the "pi@raspberrypi: $" stuff – you just type the commands that come after the $ character and hit the enter key. All commands are started by pressing the enter key.

Where am I?

You can click on the file manager icon at the top menu, a window pops up and you would normally see something like this (your files and directories may vary)

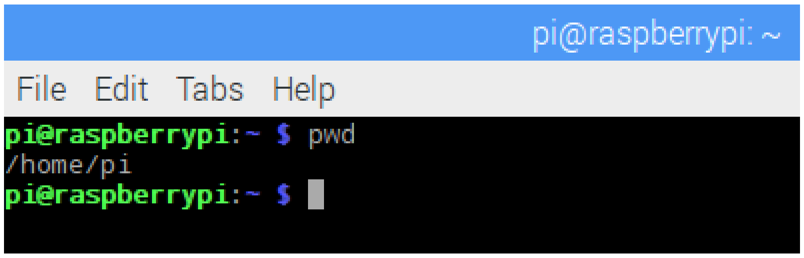


For user "pi", the home directory is /home/pi. You can click on a folder icon to view a subdirectory or click on the up arrow to view a directory above /home/pi. Now let's start using the terminal window to navigate around instead of the file manager.

type:

```
pwd
```

You should see something like this.



```
pi@raspberrypi:~ $ pwd
/home/pi
pi@raspberrypi:~ $ 
```

Make sure to press the enter key after typing the letters "pwd". The pwd command stands for "print working directory". Do you see how it matches the directory name in the file manager?

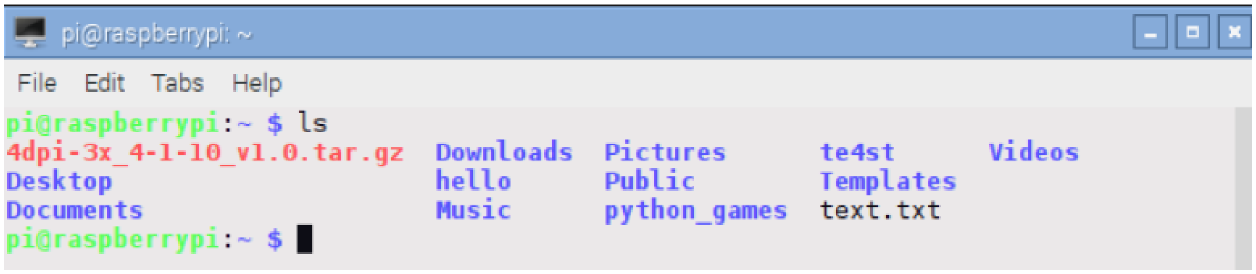
There's a special meaning for the ~ character in the command line prompt. It's a shortcut for "/home/pi". If I was user "steve", the prompt would still have the ~ in it but it would refer to /home/steve. Each user on a linux computer has a home directory. Your raspberry pi comes with just one user to start with, that's user "pi". If you add additional users, each will have their own home directory. All users directories are under the /home directory.

Listing files and directories

Type:

```
ls
```

You should see the same files and directories that you see in the file manager.c. Directories are typically in blue, regular files are regular black, compressed files are red and programs you can run will show up as green. Advanced users can change the colors.



```
pi@raspberrypi:~ $ ls
4dpi-3x_4-1-10_v1.0.tar.gz  Downloads  Pictures      te4st      Videos
Desktop                     hello      Public        Templates
Documents                   Music      python_games  text.txt
pi@raspberrypi:~ $ 
```

"ls" is a command that runs, prints some output to the window and finishes. You know it finished because the prompt appears after it's done. The terminal shell is ready for you to type more commands. You will always know a command is finished when you see the prompt printed out again.

Commands often have options.

type:

```
ls-a
```

Did it say "command not found"? You left out the space between the *s* and the *–a*. Spaces are incredibly important in linux. Let's try that again with a space between the command and the option.

```
ls -a
```

The "-a" option means to show all the files, including any normally hidden ones. Hidden files start with a period. These are usually configuration files and directories. The ones in black text are files and the ones in blue are directories containing more files.

type:

```
ls -la
```

Adding the "-l" option means "long listing". The long listing contains information file permissions, dates, sizes and owners. We'll discuss what the long listing means in a later section. At this point, just be aware you can find interesting information like the size and date created/modified.

permissions	hard links	owner	group	size	mod/create date	file name
-rw-r—r--	1	pi	pi	675	feb 9 2016	.profile

Navigation

If you like, you can think of directories as rooms in a really big house. When you type commands or look at files, you are operating in the working directory. The working directory can be found by typing "pwd" as above and it's also in the prompt. Remember ~ means /home/pi if you are logged in as user pi.

When using the mouse with a file manager you click on icons and arrows to look in other directories. In the shell, we use a command called "cd". It's short for "change directory".

to take a look around the room to find the furniture (files) and the doors (directories) type:

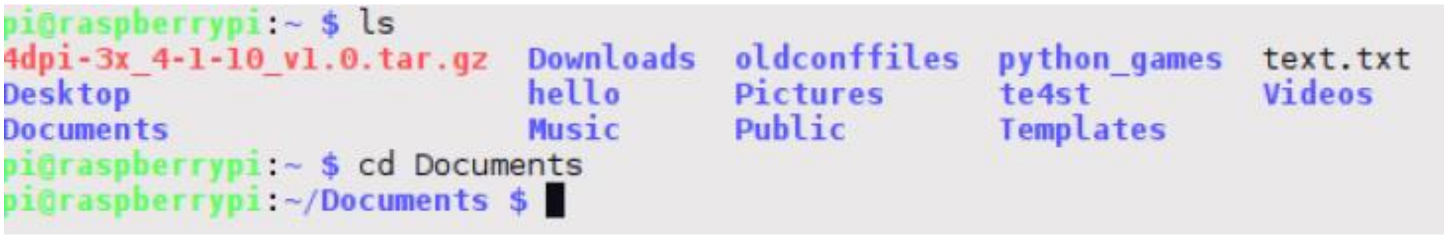
```
ls -l
```

Look for the directories in blue. Find the directory called “Documents”.

type:

```
cd Documents
```

Not much happened but the command was successful. Did you see the prompt change? The front of the prompt didn’t change because you are still logged in as user pi. The directory name part of the prompt changed.



```
pi@raspberrypi:~ $ ls
4dpi-3x_4-1-10_v1.0.tar.gz  Downloads  oldconffiles  python_games  text.txt
Desktop                     hello      Pictures      te4st         Videos
Documents                   Music      Public        Templates
pi@raspberrypi:~ $ cd Documents
pi@raspberrypi:~/Documents $ 
```

The working directory is now “/home/pi/Documents”. Look around in this directory. Type

```
ls
```

The directory may be empty or have files and other directories in it. Get in the habit of typing “ls” after you type “cd”. You are in a different directory and you want to see what’s there.

Repeat the cd command. Type:

```
cd Documents
```

It should say “No such file or directory” and the prompt remained the same. You are already in the Documents directory. There is no other Documents directory to change into. If you feel lost, type “pwd” again to see. Your location is already in the prompt but sometimes it’s nice to see it printed out.

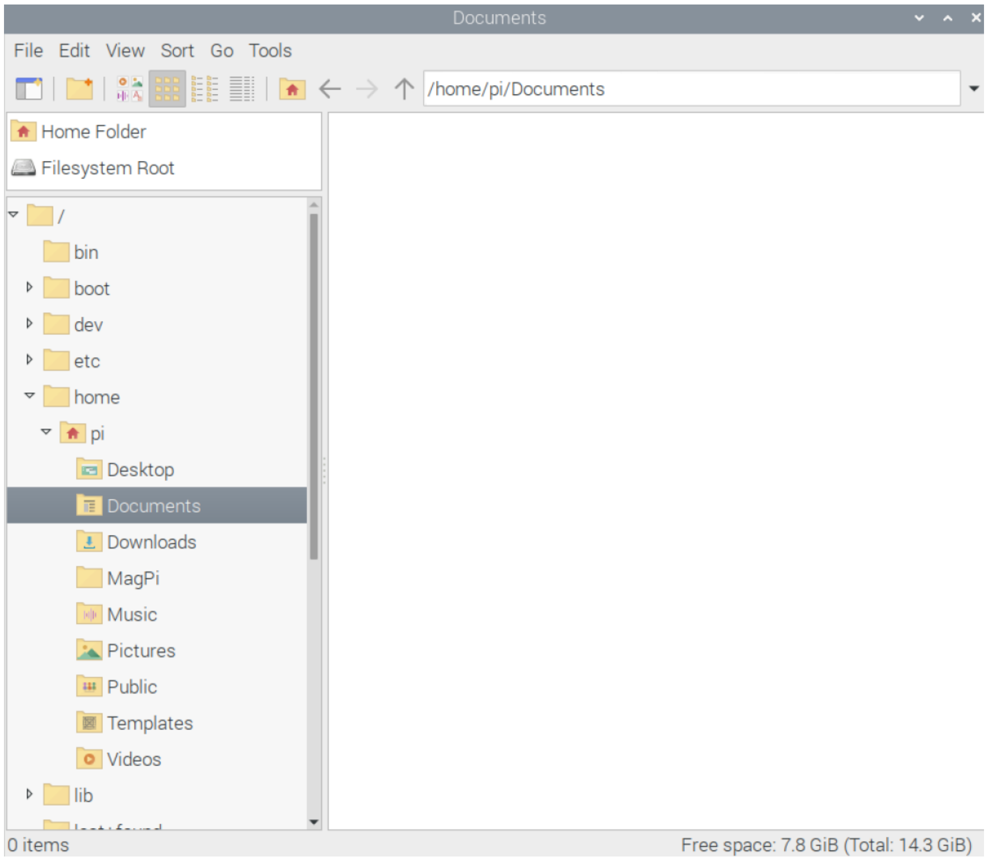
Now we’re going to return to the home directory. Type:

```
cd ..
```

```
pwd
```

You should be back in /home/pi. The **“..”** has a special meaning. It stands for the directory above the one you are in. This is called the “parent” directory. It’s the parent of the directory you are in. It takes you one directory up the hierarchy. Every directory except for the root directory has a parent directory.

Here’s a typical raspberry pi directory structure in tree form. The subdirectories in your directory should be mostly empty. Your version of the raspberry may have a slightly different look. The top level directory name is “/”. No letters or numbers, just “/”. There are several directories below it, including the home directory for user pi and several subdirectories. The parent directory of “Documents” is “pi”. The parent of “pi” is “home” and so on.



Your terminal screen should show:

```
pi@raspberrypi:~ $
```

Let’s work our way up to the top, again, Type:

```
cd ..
```

```
pwd
```

You should now be in /home. Again, Type:

```
cd ..
```

```
pwd
```

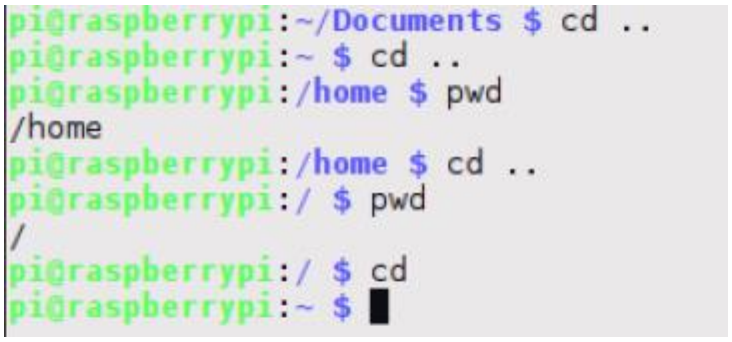
The prompt should just say “pi@raspberrypi:/ $”

You are in the top level directory. This is also called the “root” directory. You can’t go any higher. You can go lower, but not higher. Directory navigation in linux is either going “up” a level or going “down” a level. You go up by typing “cd ..” and down by typing “cd <directory name”.

Let’s get you back to your home directory. Type with no options or other letters:

```
cd
```

The prompt should say ““pi@raspberrypi:~ $”. You are back where you started.



```
pi@raspberrypi:~/Documents $ cd ..
pi@raspberrypi:~ $ cd ..
pi@raspberrypi:/home $ pwd
/home
pi@raspberrypi:/home $ cd ..
pi@raspberrypi:/ $ pwd
/
pi@raspberrypi:/ $ cd
pi@raspberrypi:~ $ █
```

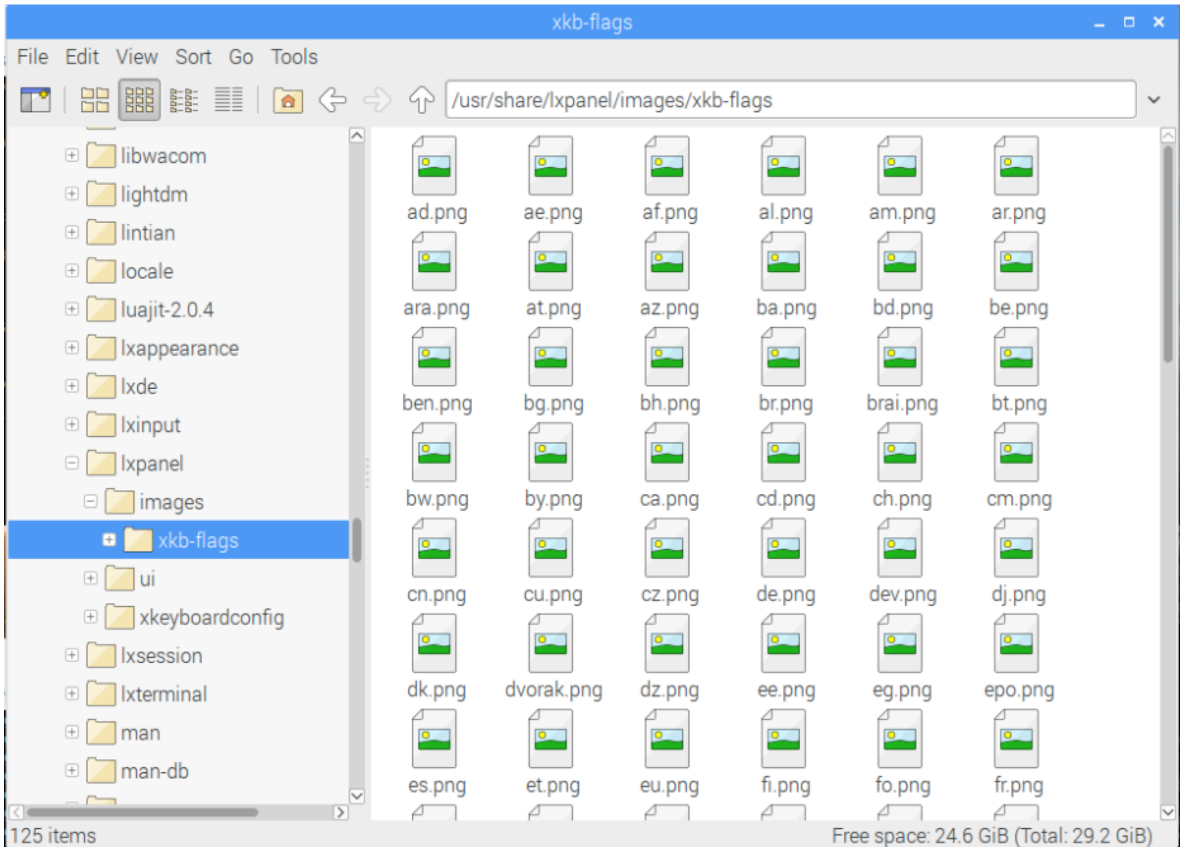
Typing "cd" by itself will always return you to your home directory. If you ever feel lost in the directory tree, type "cd" to get you back home.

Pathnames

Current working directory should be /home/pi/ or ~

NOTE: Some raspbian/noobs releases may not come with the following example directory so to see lots of files also check the "/home/pi/python games" directory if you have that directory. If you don't, any directory with lots of files and subdirectories will work.

Pathnames let you work out where you are relative to the entire file system. So far, we've been navigating up and down, one directory at a time. With a full pathname, you can navigate immediately to a directory or access a file. The full pathname of the home directory is "/home/pi". Let's look at a longer example. On the default raspberry operating system, there's a file called "af.png" in a subdirectory (assuming it hasn't been deleted for some reason):


xkb-flags
File Edit View Sort Go Tools
/usr/share/lxpanel/images/xkb-flags
libwacom
lightdm
lintian
locale
luajit-2.0.4
lxappearance
lxde
lxinput
lxpanel
images
xkb-flags
ui
xkeyboardconfig
lxsession
lxterminal
man
man-db
ad.png ae.png af.png al.png am.png ar.png
ara.png at.png az.png ba.png bd.png be.png
ben.png bg.png bh.png br.png brai.png bt.png
bw.png by.png ca.png cd.png ch.png cm.png
cn.png cu.png cz.png de.png dev.png dj.png
dk.png dvorak.png dz.png ee.png eg.png epo.png
es.png et.png eu.png fi.png fo.png fr.png
125 items
Free space: 24.6 GiB (Total: 29.2 GiB)

The full pathname of that file is **/usr/share/lxpanel/images/xkb-flags**. In the shell, navigate to that directory directly by typing:

```
cd /usr/share/lxpanel/images/xkb-flags
```

If you got the "no such file or directory" message, then try again with the correct spelling and capitalization.

If you don't have this directory, look for another directory filled with lots of files.

To see what files are in the directory, type:

```
ls
```

You should see the files pictured above. If you don't then check to see that you didn't get an error message from that "cd" command above. Check your spelling and spaces and characters.

Let's try some relative path navigation to get back to another directory. Type:

```
cd /var/log/apt
```

```
cd ../../../home/pi/Desktop
```

Of course if you are running under a different user, then substitute that username for pi.

The forward slashes separate the names. If it worked, you moved back up 3 directory levels and then down rhew directory levels in another direction. I could have also done that with several “cd ..” commands entered one after another followed by “cd Desktop” You should be in the Desktop directory that’s underneath your home directory. If you see an error, get back to your home directory and repeat the steps above.

Making Directories

Current working directory should be /home/pi/ or ~

You can create directories with a command called “mkdir”. It stands for “make directory”. Make sure you are in your home directory. Create a directory called unixstuff. Type:

```
cd

mkdir unixstuff

ls
```

You should see a new directory created. If you get a message that it already exists, that’s because someone else used your raspberry before you!!!. More likely, if you get the error, you already did this step. If you get a “permission denied” message, then you are probably in the wrong directory. Double check what the prompt says.

Command	Definition
ls	list files and directories in the current directory
ls –a	list all files including ones hidden with “.” as the first character
ls –l	displays more information about files and directories
ls –la	you can combine options to ls and "ls –al" works the same
cd	change to home directory
cd <directory>	change to named directory
cd ~	change to named directory
cd ..	change to parent directory (or one above)
cd .	change to the current directory or in other words, do nothing since you are already in the working directory
pwd	print the full path of the current directory
mkdir	make a new directory

Key Concepts

Every directory and file has a pathname

You can navigate the file system with the cd command by:

Typing in the full pathname

Going up and down one directory at a time

Stringing together cd commands on one line

Exercises – Using what you learned about ls and cd attempt the following

1. How many directories can you see in your home directory? How many, if any, files are there? Remember a directory is not a file.

2. Using cd and cd .., navigate the directories in your home directory, find any file and use ls -l to find its size. Do you think that's in bytes or kilobytes?

3. Navigate back to your home directory

2. File Editing and Creation

Current working directory should be /home/pi/ or ~

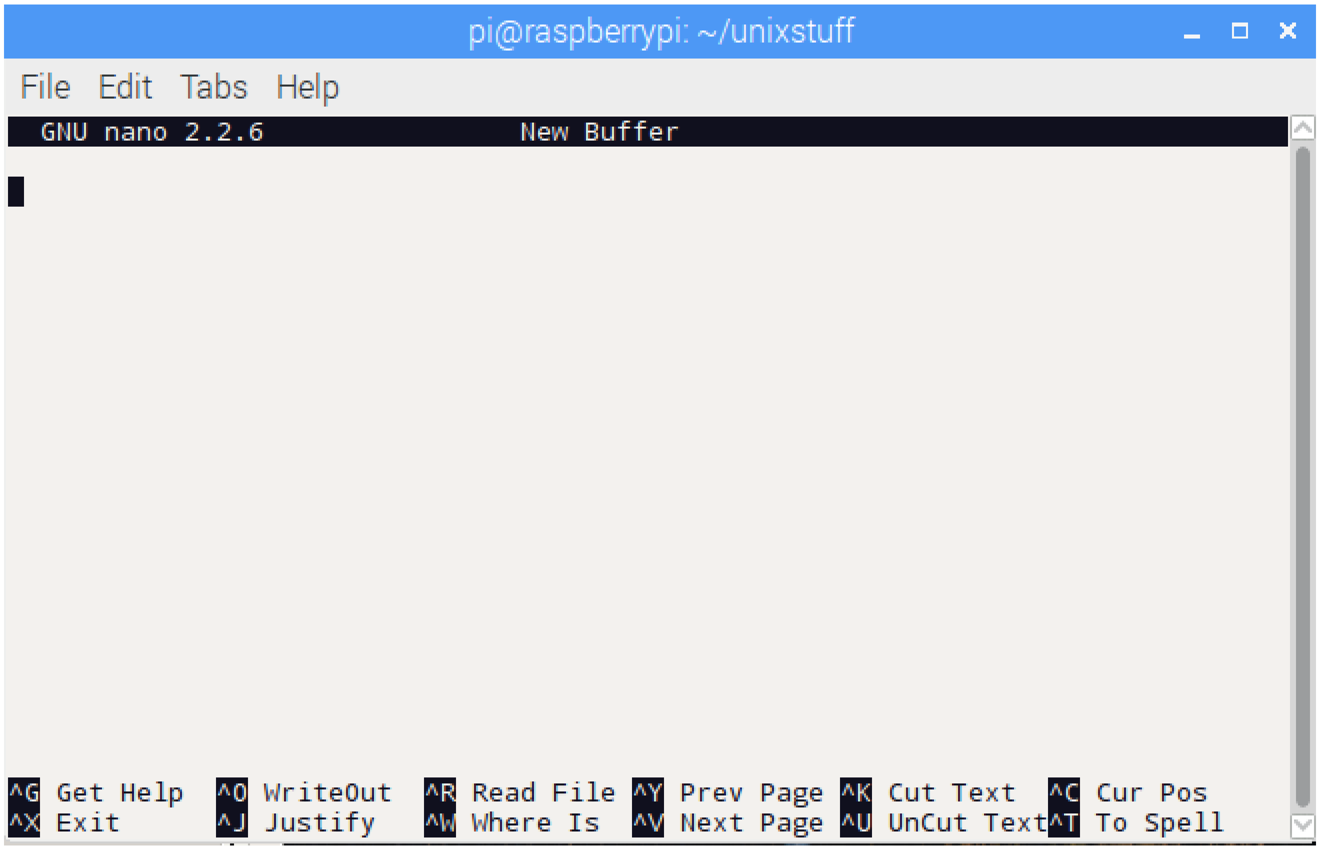
Let's get you into the unixstuff directory so we can edit and create files without making a mess in your home directory. Type:

```
cd unixstuff
```

Linux systems have many ways to edit text files. Some work entirely in the terminal window like vi, others like vim and emacs pop up their own windows. The editor "Nano" is an easy to use text editor. Let's run nano. At the prompt, type:

```
nano
```

You should see this window



```
  GNU nano 2.2.6                New Buffer

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

For the most part, you'll just use the CTRL-X and CTRL-O commands at the bottom. You do this by pressing the CTRL key down in the corner at the same type you press the letter key.

Go ahead and type whatever you want, at least several lines. When done press

```
CTRL-X
```

It will ask you to save the modified buffer, Press Y for yes and it will ask you for a file name. **Give it the name "eet178.txt"**

You should be back to the command prompt after typing the file name in. Now type:

```
nano eet178.txt
```

The file you just created should be on the screen.

Nano isn't quite like the Microsoft windows based editors. Copy/Paste/Cut don't work the same. You can't just use the mouse to grab text, press CTRL-C, and then CTRL-V to paste. You have to use the other CTRL keys and copy entire lines . Here's a link to an internet tutorial for more help - https://www.howtogeek.com/howto/42980/the-beginners-guide-to-nano-the-linux-command-line-text-editor/

You can make changes and exit to get back to the command prompt.

```
CTRL-X
```

Viewing Files

Current working directory should be /home/pi/unixstuff or ~/unixstuff

Frequently you need to view the contents of a file. You don't need to edit it – just look at it. Linux and its applications utilize so many text files for configuration and logging. You'll frequently need to look at them. The simplest linux program to view a file is called "cat". It means "concatenate the contents of the file to the screen"

Type

```
cat eet178.txt
```

The entire file, no matter how big it is, gets dumped to the screen. Let's try a much longer file. Type

```
sudo cat /var/log/boot.log
```

When the raspberry boots up, there's lots going on behind the scenes. This file contains a log of all those activities. It's more than will fit onto one screen. You also used a full pathname reference there. That boot.log file isn't in your working directory. It's somewhere else on your disk and you specified it with its full pathname.

This is a system protected file so you must use "sudo" to look at it. It's permissions are set to only allow

NOTE:->->-> If you get a "no such file or directory" message, check the spelling carefully and try again. If it still doesn't work your raspberry may not have boot logging turned on. The boot.log file is a reasonably sized file that has searchable words – so it makes a good example file. If your boot.log file doesn't exist, try using the syslog file or the messages file instead and remember to substitute those filenames for boot.log in subsequent steps.

Let's use a different program to view the file. Type

```
sudo more /var/log/boot.log
```

With this program, you have to press the spacebar for "more". Keep pressing spacebar until the entire file is scrolled onto the screen and you can see the prompt again. The percent number shows you how much of the file has been displayed.

Getting Help!

Linux has some built-in help resources when working with commands and the shell. Most commands have documentation called the "man page". That's short for "user manual". The man page is accessed by typing "man <command>". Try:

```
man ls
```

Keep pressing the space bar to see all the documentation. It's not the most user friendly way of displaying help but the style is consistent across all unix commands in all distributions and it's been this way for a few decades.

Most commands also have a "—help" (two dashes) or "-help" option that displays a subset of the information in the man page.

Copying, Moving, Renaming and removing Files

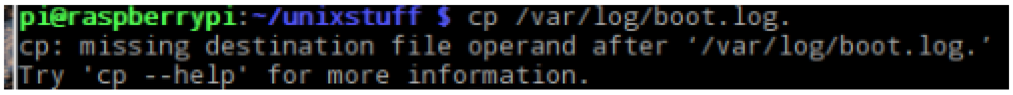
Current working directory should be /home/pi/unixstuff and at a minimum you should see these files and directories with an 'ls': eet178.txt

Let's see how files are copied. Type

```
ls
```

You should just see the one file – eet178.txt (unless you are redoing this part and you may see other files)

```
sudo cp /var/log/boot.log .
```

You should have seen this message: 

```
pi@raspberrypi:~/unixstuff $ cp /var/log/boot.log.
cp: missing destination file operand after '/var/log/boot.log.'
Try 'cp --help' for more information.
```

That's just a reminder of how important spaces are in shell commands. Let's try that again with the correct space:

```
sudo cp /var/log/boot.log .
```

```
sudo chmod 666 boot.log
```

```
ls
```

You should see a new file appear in the current directory. You told the computer to copy the file called "boot.log" that is in the directory "/var/log" to the current directory. The single period "." tells the computer that the destination is the current directory. The syntax of the cp command is "cp <source> <destination>"

Let's create a local copy of that boot.log file. Type:

```
cp boot.log boot.bak
```

```
ls
```

When you don't supply a path like /var/log, the computer just looks in the current directory. There's no path for the destination so the computer interprets it as a new file name. You kept the first part of the name but changed the extension. The ".bak" extension is popular when creating file backups.

The next command is to rename and/or move a file. Type:

```
mv boot.bak boot.backup
```

```
ls
```

This time you just renamed the file. It also shows that the file extension can be any length. Linux, unlike windows, doesn't have the three letter restriction for file extensions. Now type:

```
mkdir backups
```

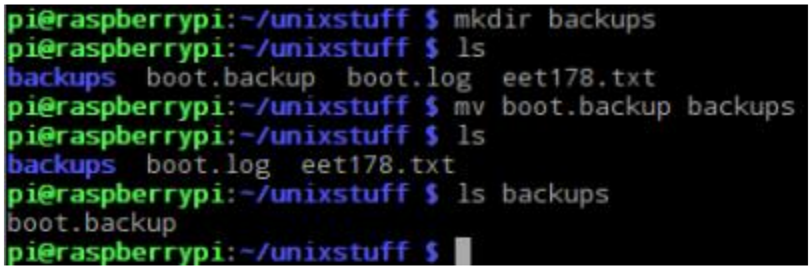
```
ls
```

```
mv boot.backup backups
```

```
ls
```

```
ls backups
```

You created a new subdirectory and moved the boot.backup file to that new directory. With the "mv" command, if the destination is a directory , not a name, it moves the file to that directory instead of renaming it. You also gave a destination directory to the "ls" command. When you do that, instead of showing what's in the current directory, it shows the contents of the destination directory. So commands can have multiple arguments and those arguments are interpreted differently depending on whether they are files or directories. After that sequence of commands, this is what it should have looked like:



```
pi@raspberrypi:~/unixstuff $ mkdir backups
pi@raspberrypi:~/unixstuff $ ls
backups  boot.backup  boot.log  eet178.txt
pi@raspberrypi:~/unixstuff $ mv boot.backup backups
pi@raspberrypi:~/unixstuff $ ls
backups  boot.log  eet178.txt
pi@raspberrypi:~/unixstuff $ ls backups
boot.backup
pi@raspberrypi:~/unixstuff $ 
```

The last command in this section is the remove or delete command. Let's create a file and then delete it. Type:

```
cp boot.log tempfile
```

```
ls
```

```
rm tempfile
```

```
ls
```

The tempfile should have appeared in the directory and then you removed it using the "rm" command. Let's try that with a directory. Type:

```
mkdir tempdir
```

```
ls
```

```
cp boot.log tempdir
```

```
rmdir tempdir
```

Got an error, didn't you? You can't remove directories with files in them. Well you can but that requires additional options. For the moment let's do it the longer way. Type:

```
rm tempdir/boot.log
```

```
rmdir tempdir
```

```
ls
```

You should be back to one directory and two files, unless of course you are repeating this tutorial and have created files in the meantime.

The two commands, "cp" and "mv" work well with files. You've learned to copy files and move them (or rename them). Let's see how they work with directories. Type:

```
mkdir tempdir
```

```
mv tempdir new_tempdir
```

```
ls
```

That renamed the directory. It works with empty directories or directories with files. Now try creating a copy. Type:

```
cp new_tempdir tempdir_copy
```

That didn't work well. Copy only works with files, unless you supply an option. That is the "-r" option. The "r" stands for recursion or repeating. In linux, it means "go into the directory and repeatedly copy everything there to the target". In less technical terms, it's the option that lets the "cp" command copy directories. Type:

```
cp -r new_tempdir tempdir_copy
```

```
ls
```

The "-r" option works with the "rm" command as well though it's actually kind of dangerous to use since if you type "rm –r" in the wrong directory, you'll lose all your files. The "rm –r" command deletes every file and every directory it can, even directories underneath directories. Let's clean up the new directories you just made. Type:

```
rm -r new_tempdir tempdir_copy
```

```
ls
```

The command will work even if the directory has files in it, because the "-r" option tells it to go into the directory first and delete all the files before trying to delete the directory itself.

Command	Definition
cp <source> <destination>	copy the file named <source> to the name <destination> and <source> and <destination> can include full path names
cp –r <source_directory> <target_directory>	copies entire directories and the files in them

mv <source> <destination>	rename <source> to <destination>. If <destination> is a directory, the keep the name and move it to the directory specified in the destination
rm <file>	deletes the file
rm –r <directory>	deletes all files in a directory and then the directory itself
rmdir <directory>	deletes the directory

Other ways to display text in files using

Current working directory should be /home/pi/unixstuff and at a minimum you should see these files and directories with an 'ls': eet178.txt, boot.log, backups

Many linux applications create several large text based log files as they run. A typical job of a software/hardware technician is to monitor and troubleshoot application issues by examining log files. You aren't troubleshooting the source code but usually troubleshooting configuration or run time issues. These files can run to several thousand lines and you can't realistically read the entire file. So, linux has some commands to help with looking at parts of files.

You've seen the cat and more commands which display the entire file. There are two lesser known but often useful commands called "head" and "tail". The "head" command displays the head or beginning lines of a file. The "tail" command displays the ending lines of a file.

Try this example:

```
head -3 boot.log
```

The first option specifies the number of lines to display to the screen and the 2nd argument is just the file name.

The tail command works the same way and is useful for log files. Often you are looking for recent events in the log file and jumping right to the end of the file saves time.

```
tail -3 boot.log
```

When you have to find specific information in a large file, you usually start with a keyword. Linux has a command to search files for keywords. That command is called "grep". It's been in linux systems since the 70's and sort of comes from "Globally search a REgular expression and Print" – G RE P. The keyword is in the form of a regular expression.

Let's look at the boot.log file for some information. As the raspberry boots up, the operating system is starting a lot of services and doing a lot of initialization. Linux programmers like to be verbose with their software so each little thing gets announced in the log file. For the most part, users don't look at the boot file unless something goes wrong. Now one of the important steps in an operating system is bringing up the network. The raspberry needs to be on the network in order to talk to the network. The raspberry has two ways to reach the internet – through the Ethernet jack and through the wifi. These have names in the operating system. The wifi system on the raspberry is called "wlan0". It stands for "wireless local area network" and it's the first and only one so it has the number 0.

So we're having a problem with the wifi and we want to check to see if the operating system started the program that runs the wifi. You don't want to read the entire file – only look at the lines that have the word "wlan0" in them. Type:

```
grep -n LSB boot.log
```

The command name is “grep”. The “-n” option will display line numbers, the “wlan0” is the word you are searching for, and “boot.log” is the file to search. Assuming you have no issues on your raspberry, you should see two matching lines. The first line is the operating system saying it’s starting the wifi program. The word “ifup” stands for “interface up” and we already talked about wlan0. The second line tells us that the startup went OK and the wifi on the raspberry is working.

If you want to search for a sentence, type:

```
grep -n “Starting LSB” boot.log
```

(it’s possible that the items written to the boot.log file have changed over time and if nothing is returned, pick another sentence from the boot.log to search for)

The grep command has a lot of options, type:

```
grep --help
```

All commands have the “—help” option so you can see all the options!

You can combine options. For example, let’s find how many lines don’t have the word “LSB” in them. Type:

```
grep -ivc LSB boot.log
```

The “-i” option tells the program to ignore case – LSB and lsb land Lsb are all the same. The “-v” option tells the program to invert the search – look for lines that don’t have the keyword. The “-c” option tells the program to just count the lines and don’t display them.

One last command that’s sometimes useful is “wc” – word count. Type:

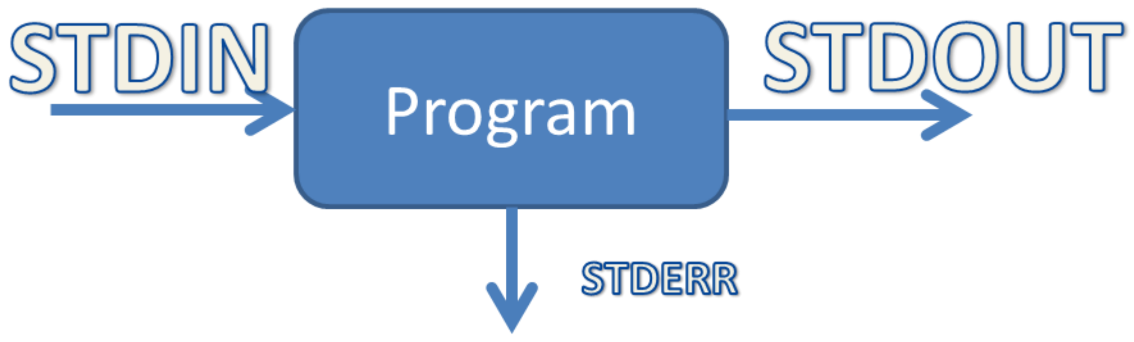
```
wc -wl boot.log
```

This tells us the number of lines and words are in the file. On my system it displays 72 and 499.

Redirecting and Piping the input and output

Current working directory should be /home/pi/unixstuff and at a minimum you should see these files and directories with an 'ls': eet178.txt, boot.log, backups

These commands you’ve been running are really just computer programs. Computer programs in linux have instructions that write text out for the user to see in the terminal window (shell). This is called STDOUT (standard output). There’s also STDIN (standard input) which is just whatever you type on the keyboard. The keyboard is the default standard input device. Finally, there is a third type of output called STDERR (standard error). A program can designate some output to be error messages. Normally these just go to the terminal window but you can make them go to a file of your choice. These are also called input and output ***Streams.***



With redirection, we can redirect the input and output of commands to files. Let's look at how the "cat" command handles standard input and output. Remember that the "cat" command is the simplest one to use that displays file contents. Let's try using it in some different ways. Type, with no other options:

```
cat eet178.txt > temp_file
```

It looks like nothing happened! Before when you tried the "cat" command, it printed out the contents of the file to the string. This time, by using the ">" symbol, it created a new file and sent all that text into the new file. Sometimes in linux the lack of an error message means the command was successful. Type:

```
ls
```

```
more temp_file
```

Now try the "cat" command in another way: Type, by itself with nothing else:

```
cat
```

Notice that you have no prompt now. The computer is now waiting for you to type characters on the keyboard. Type a few words and press the enter key. The "cat" command is accepting what you typed on standard input and sending it out on standard output.

To exit the command – press CTRL-D or ^D (some instructions will refer to the action of pressing the ctrl key and a letter as ^D). You can also press CTRL-C but these have two different meanings in linux. CTRL-C means to interrupt or break what is running and CTRL-D means "end input" or "input is finished".

Let's play with redirecting the output. Type:

```
cat > fruit_list
```

Now type in the names of some fruits followed by CTRL-D:

```
apple
```

```
orange
```

```
pineapple
grape
banana
CTRL-D
```

The “cat” program reads the standard input and then echoes it out to standard output – but you’ve told it to redirect standard output to a file instead of the screen with the “>” symbol. Type:

```
more fruit_list
```

Here’s a new command “sort”. It alphabetically sorts each line. Type:

```
sort
ford
chevy
nissan
toyota
CTRL-D
```

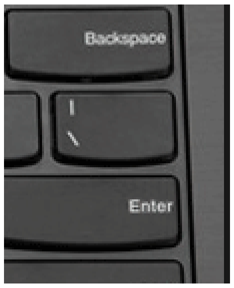
The command rearranged what you typed and then wrote it out to standard output. Type:

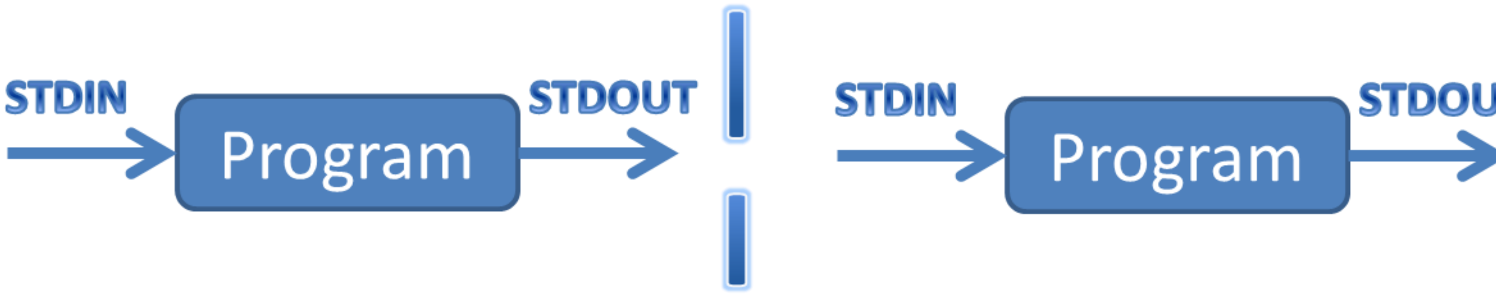
```
sort < fruit_list
```

Instead of getting input from the keyboard, the “<” told the command to get its input from the file called fruit_list and send it out to the standard output (the terminal window). You can direct input into a command and direct output from a command on the same line. Type:

```
sort < fruit_list > sorted_list
more sorted_list
```

There's one last type of direction and that's called piping. With the < and > redirection, you have to create intermediate files. Piping means to take the output of one program and send it directly to the input of another program. The symbol for pipe is the vertical bar. It's usually combined with the backslash character and near the enter key.





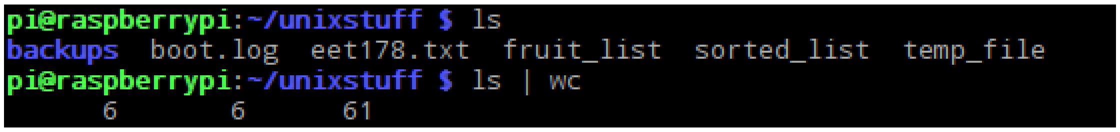
Type:

`ls`

How many "words" or files and directories do you count? If you have started this tutorial on a clean machine, you should see about 6 entries or words separated by spaces. Now type, using the "pipe" key and watching for spaces on either side of the pipe symbol:

`ls | wc`

You should have seen (assuming you have the same files):



```
pi@raspberrypi:~/unixstuff $ ls
backups  boot.log  eet178.txt  fruit_list  sorted_list  temp_file
pi@raspberrypi:~/unixstuff $ ls | wc
      6       6      61
```

It's telling you there are 6 "newlines", 6 words, and that there were 61 characters.

Command	Definition
head –<number> <file>	displays the first lines of the file. The number of lines displayed is given by the -<number> parameter
tail –<number> <file>	displays the last lines of the file. The number of lines displayed is given by the -<number> parameter
grep <keyword> <file>	search a file for any lines that contain the <keyword>. There are many options to grep. If the <keyword> is

	actually a phrase with spaces, then it must be surrounded by quote marks.
wc <file>	counts the number of lines, words, and total bytes of a file or whatever is sent to STDIN
|	connects the output of one command to the input of another command
>	takes the output of a command and sends it to a file
<	The input for a command comes from a file instead typing it in
CTRL-D	Signals the end of entering text to a command

Exercises – Using what you learned about redirection and pipes attempt the following

1. How many files and directories, including hidden, are in your home directory?

2. Create a single line command using grep, then a pipe symbol and then the wc commands to count how many times does the word "Starting" appears in boot.log

    a) Use grep to search for "Starting" in boot.log

    b) Redirect the output from that grep using the pipe symbol into the input of the wc command.

3. Research the “cp” command by typing "man cp" or consulting your linux guide. Find the "-r" option. It copies both directories and files recursively. Create a new directory in your home directory and copy all the directories and files in the /etc directory to the new directory you created. You may get several errors about permission denied illustrating the fact that you don’t have default access to even view a lot of directories or files – but if you check into this new directory you created, you’ll see quite a few files and directories.

4. Delete the new directory and all of its contents using a single command. The rm command also has a "-r" option. Use it wisely.

3. Groups and File permissions

Current working directory should be /home/pi/unixstuff

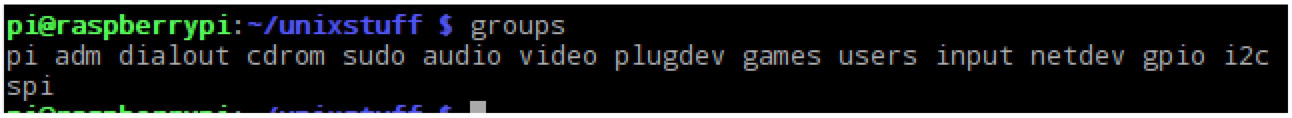
Linux was originally developed from an operating system referred to as UNIX. UNIX was designed for multi-user computers. Many people could be logged into a single system at the same time. Even when people weren't logged in, it was possible to view their directories and files. It was risky to let any user do anything they want or have access to any file on the system. That is part of the reason groups were invented in UNIX (and subsequently Linux). Groups could be defined for various purposes. For example, a group was created to have access to the printer. Only people added to the printer group could print files. In an academic setting there would be a group for faculty and a separate group for students. Faculty could share files among themselves but students would not be able to view the files because they were not in the faculty group.

Though many linux systems these days are only used by one person, the raspberry pi for example, the concept of groups has become an important part of linux security. If a user or application needs to access or do something with security implications, they need to be able to have access to the correct group.

Let's look at the raspberry PI groups. Type:

```
groups
```

You should see something like this:



```
pi@raspberrypi:~/unixstuff $ groups
pi adm dialout cdrom sudo audio video plugdev games users input netdev gpio i2c
spi
```

There are several groups, most of which you'll likely never need to worry about. These are the groups that user pi belongs to. If user pi wants to access the GPIO pins, they are in the correct group to do that. A raspberry PI has about 63 different groups, mostly for various operating system functions. If a user account is setup correctly, there's usually no need to change or add users to groups. However, knowing about groups helps with the next topic of file permissions. To get back to your home directory and view a long listing, type:

```
cd
```

```
ls -la
```

Some of the files and directories you might see are listed here:

permissions	hard links	owner	group	size	mod/create date	file name
-rw-r-r---	1	pi	pi	675	feb 9 2016	.profile
drwxr-xr-x	2	pi	pi	4096	Mar 3 08:24	Desktop
drwx------	12	pi	pi	4096	Mar 15 16:57	.config

Let's look at each of these fields. The first is the permissions.

The first character for file type is easy. A 'd' means it's a directory. a '-' means it just a regular file.

Permissions									
file type	owner of the file/directory			members of the group			all other users		
	r	w	x	r	w	x	r	w	x
d or -	read	write	execute	read	write	execute	read	write	execute

Using the file examples above, the way to interpret the permissions is like this:

permissions	file name	explanation
-rw-r-r---	.profile	the file owner can read, delete and modify

		the file
drwxr-xr-x	Desktop	the owner can delete the directory and delete or add files to the directory. Group members and other can only change into the directory and view files there
drwx------	.config	Only the owner can delete the directory and add or delete files in the directory. Others can't cd into the director and can't view files in the directory

There are three sets of permissions for three types of users:

The file owner or person who initially created the file or directory

The file or directory belongs to a listed group

Everyone else with an account on the system

There are three types of permissions

read – can view the contents of the file or copy it

write – can change the contents of the file or create files in the directory

execute – can run the file as an executable program or change into the directory.

All can be set independently. The letter r, w, or x means the permission is granted. The "-" or dash means the permission is not granted.

Look at the list of files above, here's how to interpret the permissions for each

.profile – the file owner pi can read and write the file but not run it as a program. Anyone in the group pi can only read the file and any other user on the system can read the file.

Desktop – it's a directory. The user pi can see files in the directory, can create and delete the directory and "cd" into the directory. Other users and users in the group pi can see files in the directory but not create new ones or delete anything.

.config – it's a directory. The user pi can see files in the directory and create delete as well. All other users on the system can see the directory name but can't cd into it, view any files, or create files in it.

The other fields are pretty simple. The owner of all these files is user pi. All of these files belong to the group pi. The size of directories is usually 4096 bytes. The mod/create date is the last time and day that the file was changed. Let's ignore the hard links field.

You can change the permissions of files and directories with the "chmod" command. The word "chmod" comes from "change the mode bits" of a file. The "chmod" command identifies the three sets of users as:

u – user or owner

g – group

o – others

a – all (user, group and others)

The command uses + for adding a permission and – for removing a permission to a user set. The letters r, w and x are used to identify which type of permission. For example, "g+w" means to give the group permission to write or modify the file. Another example, is "a+rw" which means to give every user the ability to read and write the file. Now try this yourself. Type:

```
cd ~/unixstuff
```

```
chmod a-rw eet178.txt
```

```
ls -l
```

```
more eet178.txt
```

If everything worked properly you should have gotten a "permission denied" message and you are now unable to see or edit your file. The permissions on the file listing for that file should be all dashes. Don't worry, you will set it back by typing:

```
chmod u+rw eet178.txt
```

```
ls -l
```

You've given yourself back permission to read and write the file. You can restore permissions to the group and others if needed by calling the command again with different parameters.

There's another way to set permissions using the file mode bits as binary numbers. Here's a table that decodes the rwx settings into numbers:

for user, group OR all		
permissions	binary	decimal value
---	000	0
--x	001	1
-w-	010	2
-wx	011	3
r--	100	4
r-x	101	5
rw-	110	6
rwx	111	7

The decimal value can replace the rwx or binary permissions. For example, here are some typical codes:

777 read, write and execute for user, group and all (rwxrwxrwx)

644 read and write for user, just read only for group and all (rw-r—r—)

755 The owner can modify (rwx) and all others can just view or cd (rwxr-xr-x)

If you wanted to change the permissions for the file eet178.txt and give yourself read/write permissions and everyone else just read permissions, you could type "chmod u=rw,g=w,o=w eet178.txt. That's a lot to type. There's a shorter way, type:

```
chmod 644 eet178.txt
```

```
ls -l
```

The '6' is read/write for user

The '4' is read for group

The '4' is read for everyone else.

You changed the eet178.txt file to have read/write permission for yourself and read access for the group and for everyone else. The 6 corresponded to your permissions and the 44 corresponded to permissions for group and others. Let's try one more example. Type:

```
chmod 077 backups
```

```
ls
```

```
cd backups
```

You got the permission denied message again although anyone else with an account on your raspberry would be able to access the backups directory. To change it back, type:

```
chmod 755 backups
```

```
ls
```

Before leaving the topic of permissions, let's look at something more complicated. Type

```
ls -la /var/log
```

What's interesting about these files in directories are that most of them are owned by user "root" and that some of them belong to other groups such as "adm". Files in this directory are log files produced by various bits and pieces of the operating system.

The "root" user is a special user with full permissions to do anything, mostly administrative operating system things. The user "root" is also known as the superuser. Most of the operating system files and directories on the raspberry are owned by root. You can't login as root but you can assume the administrative powers of root.

This is done with the "sudo" command. It means "execute the next command as if I was the superuser". Anytime you need to modify a system file, you precede the editor command with the term "sudo" – for example "sudo nano /etc/network/interfaces".

Command	Definition
groups	tells you what system groups you, the current user, belong to
chmod <parameters> <file or directory>	changes the file and directory access permissions for three sets of users – the file owner, the group and everyone else.

Key Concepts

Linux controls access to files and directories using groups and permissions

Permissions can be changed with the chmod command

You can restrict access to your files and directories

Some files and directories controlled by the operating system can't be changed unless you become a superuser

Exercises – Using what you learned about redirection attempt the following

1. When you create a new file using a text editor, what are the standard permissions of that file?

2. Create a new directory in your home directory and try to move that directory to the same level as your home directory (alongside the "pi" directory in /home. For example for user pi, create directory eet123 under /home/pi and try to move it to /home. Did it work? Why not?

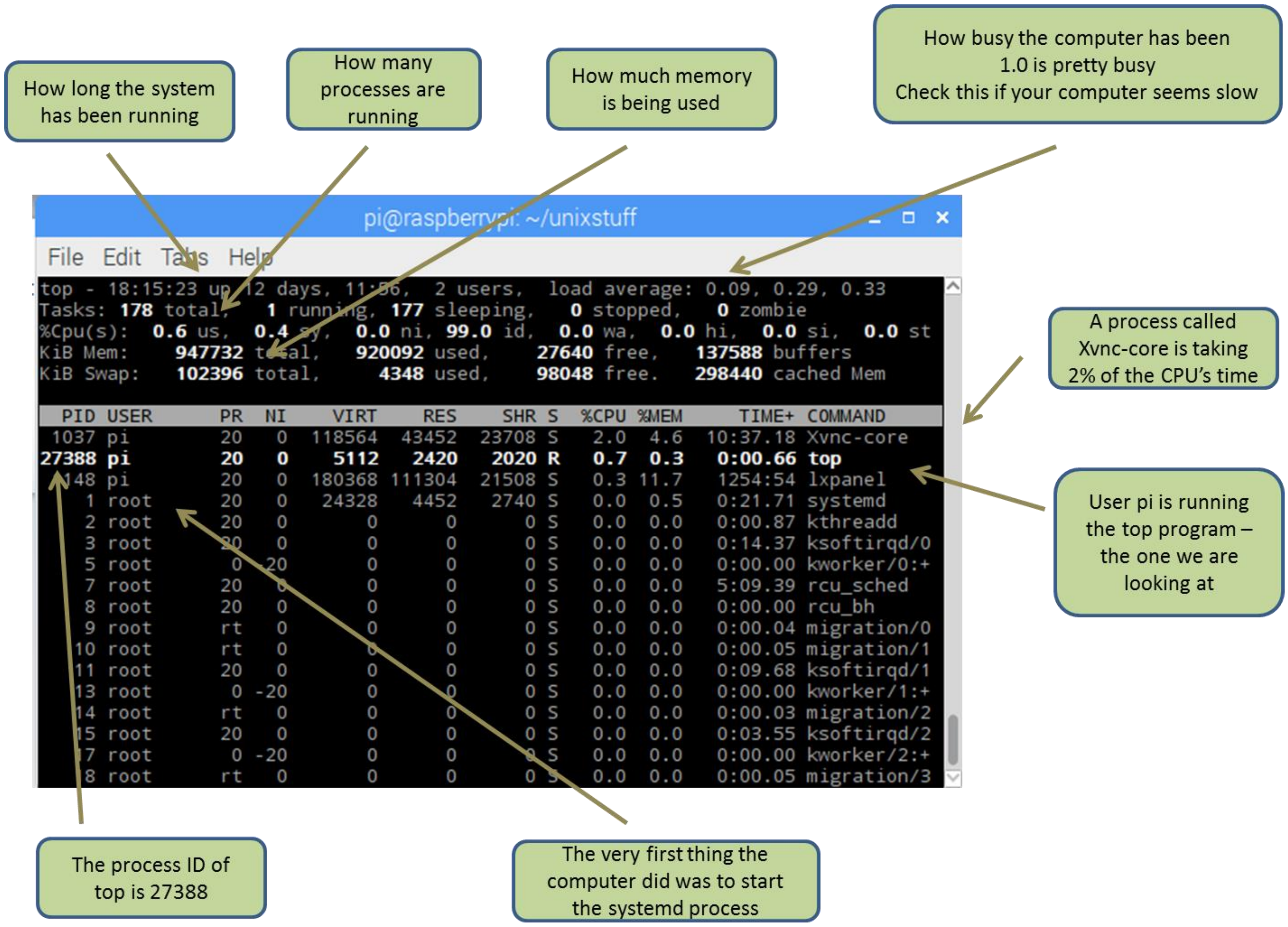
Controlling the execution of processes

Current working directory should be /home/pi/unixstuff

Linux is a multi-tasking operating system. That means that the processor or CPU inside the computer is running multiple programs at the same time. We call these processes. The computer runs many processes by taking turns running different processes. For a fraction of a second, the process controlling the display is run and then the computer switches to the process handling the network, then it switches to the process behind the internet browser. This goes on until the computer gets back to the process controlling the display and it starts all over again. These are tiny fractions of a second so you never notice. Let's take a look at these processes. In the terminal window, type:

```
top
```

It's a very busy window with a lot of information. Press CTRL-C to exit. Here's a typical screenshot:


How long the system has been running
How many processes are running
How much memory is being used
How busy the computer has been 1.0 is pretty busy Check this if your computer seems slow
A process called Xvnc-core is taking 2% of the CPU's time
User pi is running the top program – the one we are looking at
The process ID of top is 27388
The very first thing the computer did was to start the systemd process
pi@raspberrypi: ~/unixstuff
File Edit Tabs Help

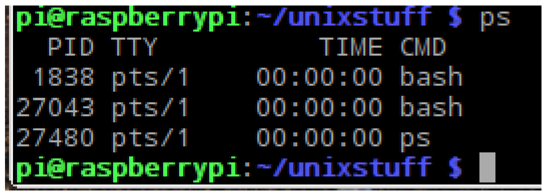
```
top - 18:15:23 up 12 days, 11:56,  2 users,  load average: 0.09, 0.29, 0.33
Tasks: 178 total,   1 running, 177 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.6 us,  0.4 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:    947732 total,   920092 used,    27640 free,   137588 buffers
KiB Swap:   102396 total,     4348 used,    98048 free.   298440 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1037 pi        20   0  118564  43452  23708 S   2.0  4.6  10:37.18 Xvnc-core
27388 pi        20   0    5112   2420   2020 R   0.7  0.3   0:00.66 top
 1148 pi        20   0  180368 111304  21508 S   0.3 11.7   1254:54 lxpanel
    1 root      20   0   24328   4452   2740 S   0.0  0.5   0:21.71 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.87 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:14.37 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:+
    7 root      20   0       0      0      0 S   0.0  0.0   5:09.39 rcu_sched
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S   0.0  0.0   0:00.04 migration/0
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.05 migration/1
   11 root      20   0       0      0      0 S   0.0  0.0   0:09.68 ksoftirqd/1
   13 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:+
   14 root      rt   0       0      0      0 S   0.0  0.0   0:00.03 migration/2
   15 root      20   0       0      0      0 S   0.0  0.0   0:03.55 ksoftirqd/2
   17 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/2:+
   18 root      rt   0       0      0      0 S   0.0  0.0   0:00.05 migration/3
```

Make sure to press CTRL-C to exit back to the shell prompt.

Linux itself is pretty stable but sometimes you need to stop and start processes or you are running applications that require stopping and starting. There's another command that is more often used to see what processes are running, especially in the specific terminal window. Type:

```
ps
```

You should see something like this (but your PIDs will be different):



```
pi@raspberrypi:~/unixstuff $ ps
  PID TTY          TIME CMD
 1838 pts/1    00:00:00 bash
27043 pts/1    00:00:00 bash
27480 pts/1    00:00:00 ps
pi@raspberrypi:~/unixstuff $ 
```

The two entries for "bash" refer to the shell itself and it's running on TTY pts/1. TTY is an old word that used to mean "teletypewriter" but now it usually identifies which window. If you open additional terminal shells, the next one will probably be "pts/2". The terminal shell has two process IDs (PIDS) – 1838 and 27043. Sometimes you may just see a single PID for bash. "bash" stands for "bourne again shell" and "bourne" is the name of a computer scientist from the United Kingdom who developed the original version of the shell (or command terminal interface) we use today.

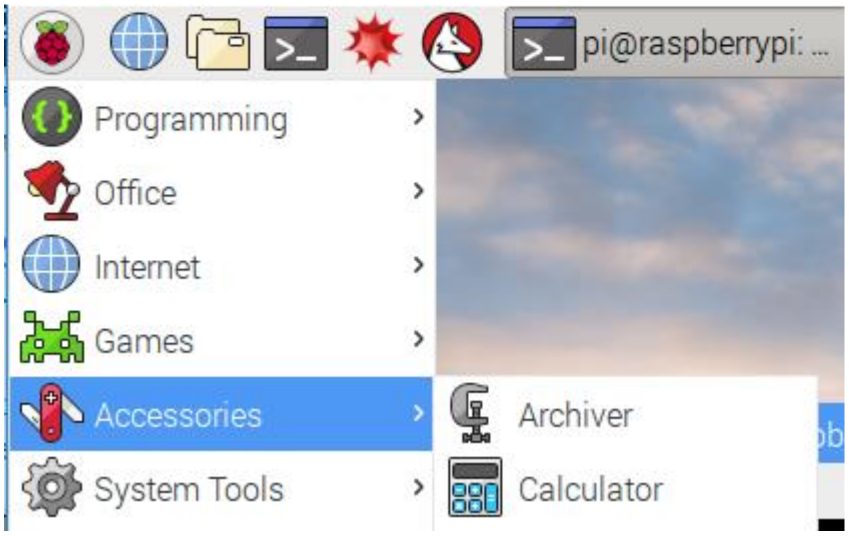
The other entry is for the "ps" program you just ran. The computer started the ps command and gave it the process ID of 27480. If you run "ps" again, it will have a new number because the "ps" command ran, looked at what was running, and then stopped.

"ps" with no other parameters or arguments just displays the processes associated with the shell where it was executed. Type:

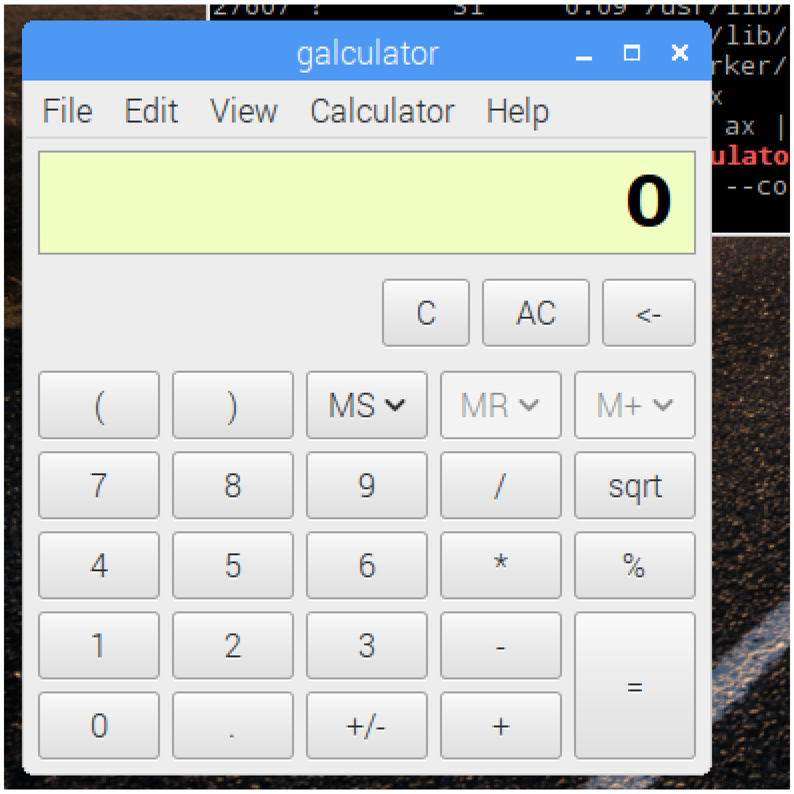
```
ps aux | more
```

Remember to keep pressing the space bar until you get back to the prompt. This shows you all of the couple hundred processes running on a raspberry. Usually we're just interested in the name, the process owner and the PID.

Let's do something with this information and start controlling some processes. On your raspberry, click on the raspberry menu in the upper left corner and open up accessories. Click on the Calculator icon.



You should see a simple calculator window pop up:



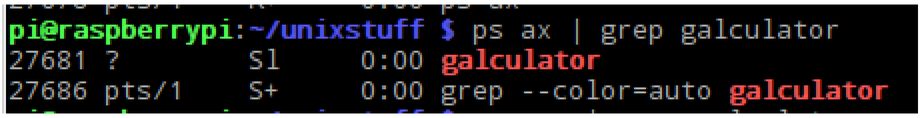
Go back to the shell window. In the shell type:

Type:

```
ps ax | grep galculator
```

This is a shorter version of the ps command. The grep searches that long list of processes and only gives us the lines that have the word "galculator" in them. We also used the pipe symbol again!

Look for the line that has galculator on it and find the PID number at the very left. On my raspberry it looks like this:



```
pi@raspberrypi:~/unixstuff $ ps ax | grep galculator
27681 ?        Sl     0:00 galculator
27686 pts/1    S+     0:00 grep --color=auto galculator
```

On my raspberry the galculator program as the PID of 27681. On you raspberry it will be a different number. Let's say the galculator program is stuck or not responding. You want to kill it but clicking on the kill 'x' isn't working. Type:

```
kill 27681
```

The galculator window went away! If you type the ps command again, you'll see that the only entry appearing is the one from the grep command we used to search for it.

The galculator app was a user app and is easy to "kill". If you are doing system administration tasks and need to kill a system process, you may need to supply an option to kill – the "-9" option. For example, if you have a terminal window shell that seems stuck, find the PID and type "kill <PID>". If that doesn't work, you would type "kill -9 <PID>". The "-9" sends a different signal to the process manager.

Foreground and Background Jobs

Current working directory should be /home/pi/unixstuff

Nearly all the processes you see using the ps and top commands are running in the background. That command you ran a few steps ago – top, was running in the foreground. Running in the foreground means the shell can't accept any further commands because it's busy. You can run processes and commands in the background and free up the shell to run other commands.

We are going to use two commands called "sleep" and "jobs" to control background and foreground jobs. Type the following command and count 5 seconds:

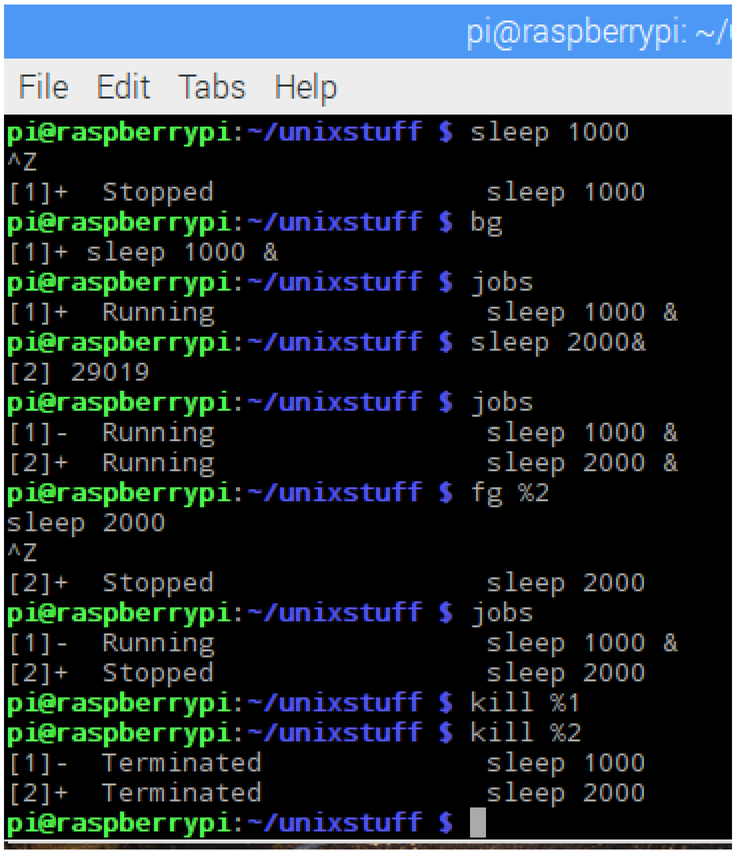
```
sleep 5
```

Sleep is a command that does nothing for the number of seconds given in the argument. It pauses the shell for that amount of time. There are other uses for the sleep command but we're using it in a simple way. The shell prompt came back in 5 seconds. Now type:

```
sleep 10&
```

The "&" at the end of any command tells the computer to run this command in the background. The PID is displayed. While this is running in the background for 10 seconds, you can run other commands at the same time because the shell prompt came back right away.

You can move processes between the foreground and background and stop them when needed. For example, look at the following sequence:



```
pi@raspberrypi:~/unixstuff $ sleep 1000
^Z
[1]+  Stopped                 sleep 1000
pi@raspberrypi:~/unixstuff $ bg
[1]+ sleep 1000 &
pi@raspberrypi:~/unixstuff $ jobs
[1]+  Running                 sleep 1000 &
pi@raspberrypi:~/unixstuff $ sleep 2000&
[2] 29019
pi@raspberrypi:~/unixstuff $ jobs
[1]-  Running                 sleep 1000 &
[2]+  Running                 sleep 2000 &
pi@raspberrypi:~/unixstuff $ fg %2
sleep 2000
^Z
[2]+  Stopped                 sleep 2000
pi@raspberrypi:~/unixstuff $ jobs
[1]-  Running                 sleep 1000 &
[2]+  Stopped                 sleep 2000
pi@raspberrypi:~/unixstuff $ kill %1
pi@raspberrypi:~/unixstuff $ kill %2
[1]-  Terminated              sleep 1000
[2]+  Terminated              sleep 2000
pi@raspberrypi:~/unixstuff $ 
```

Start the sleep command in the foreground
press CTRL-Z to suspend the command
"bg" restarts the command in the background
"jobs" tells us what is running
the job number is 1
start another sleep command in the background

see that both commands are running

bring the 2nd sleep command back into the foreground and then suspend it with the CTRL-Z

list the running commands again

kill both commands using their job numbersb
both commands have been terminated

Command	Definition
top	gives you information about the system and what processes (tasks) are running
ps	lists the running processes and tells you what their PIDs are in the current shell
ps aux	lists all running processes in the computer – a simpler version of top without the system information
kill <PID>	The command to terminate a process given the PID
kill %<job number>	The command to terminate a process in a shell given it's job number (which is not the same as the PID)
sleep <time>	Pauses the shell (but not the entire computer) for the time in seconds
CTRL-C	interrupts and stops a running process or command
CTRL-Z	suspends but doesn't kill a running process or command
jobs	lists what processes are running in the current shell (it won't show you any processes running from the desktop or other shells)
bg	restarts a suspended process and runs it in the background
fg	moves a background or suspended process into the foreground. The shell can't accept any commands while it's running

Key Concepts

> **Linux is made up of many running processes**
>
> **Commands that you run are processes run from a shell or terminal window**
>
> **A foreground process means the shell runs that process until it is complete and you can't start any new ones**
>
> **A background process means you can start other commands in the shell while the background process is running**
>
> **You can terminate processes with the kill command**
>
> **You can suspend running processes and move them back and forth between the foreground and background**

Exercises – Using what you learned about redirection attempt the following

1. Start a different app from the raspberry menu, start it and then stop it from a terminal window shell

2. What do you think "kill -9 1" will do?

3. Open up a new terminal window, in addition to the one you've been working in – so you have two terminal windows visible.

4. Run the "top" command in one terminal window shell and in another terminal window shell, run the command "find /." What happens to the load average?

5. How long has your system been running?

Wildcards

Current working directory should be /home/pi/unixstuff and you should see these three files from a previous step – eet178.txt, fruit_list and sorted_list

Sadly this is not a section on poker. A wildcard in Linux is actually a set of symbols that can match one or more characters. The symbol used most is "*". For some commands, that's not a multiply sign, it represents "match any character". The "*" is used with any command that deals with files such as "ls", "more", "wc" and "grep". Type:

```
ls -l fruit_list
```

```
ls -l *fruit_list
```

`ls -l *list`

`ls -l *.txt`

`ls -l *txt`

`ls -l *t`

`cp fruit_list fruit_list2`

`ls fruit*`

The "ls" command by itself lists all files in a directory. The "ls" command with all or part of a file name lists only those files that match that name or part of the name. The "*" character can match zero characters, one character or any characters. The symbol "*" can go anywhere in the name and you can have multiple wildcards in the name.

Useful Commands

Current working directory should be /home/pi/unixstuff and you should have at least these files – boot.log, and fruit_list and fruit_list2

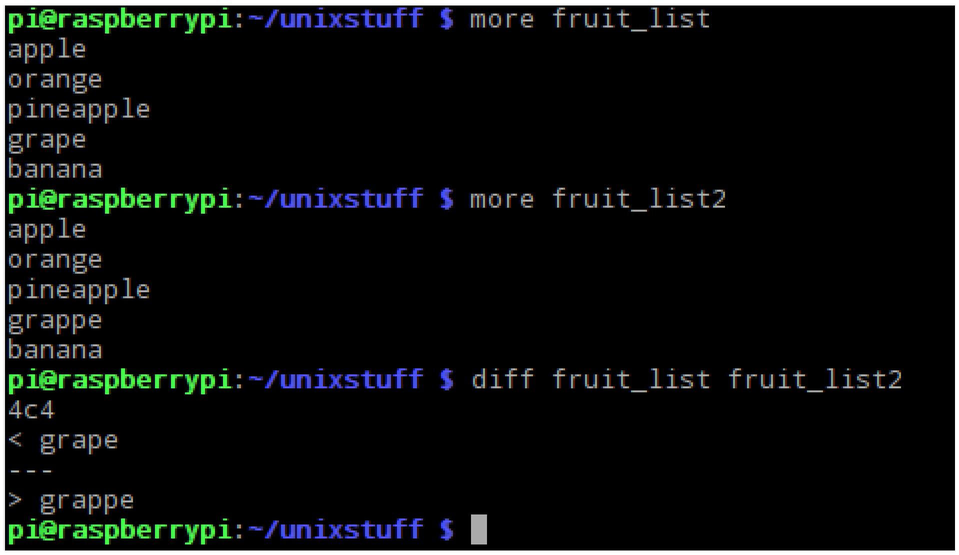
Finding differences between files

Linux has a command that finds differences between text files. It's called "diff".

1. Use nano to open up fruit_list2 and change the spelling of one of the fruits on one line

2. Run this command `diff fruit_list fruit_list2`

Here's how that worked on my system:



```
pi@raspberrypi:~/unixstuff $ more fruit_list
apple
orange
pineapple
grape
banana
pi@raspberrypi:~/unixstuff $ more fruit_list2
apple
orange
pineapple
grappe
banana
pi@raspberrypi:~/unixstuff $ diff fruit_list fruit_list2
4c4
< grape
---
> grappe
pi@raspberrypi:~/unixstuff $
```

The "4c4" means that line 4 in the first file is changed from line 4 in the second file. The "grape" is line 4 from the file fruit_list and the "grappe" is from the file fruit_list2

Compressing and uncompressing files

Windows users may be used to the "zip" function. When downloading files from the internet you may have to "unzip" files although the operating system makes that easy. In Linux the utility "gzip" is used to compress files for space savings. The "g" in "gzip" comes from the word GNU which typically means that it's free and open source. Type:

```
ls -l boot.*
```

```
gzip boot.log
```

```
ls -l boot.*
```

```
more boot.log.gz
```

```
gunzip boot.log.gz
```

Several things happened here:

1. The boot.log file went away and was replaced by boot.log.gz – a smaller file

2. The compressed file is unreadable

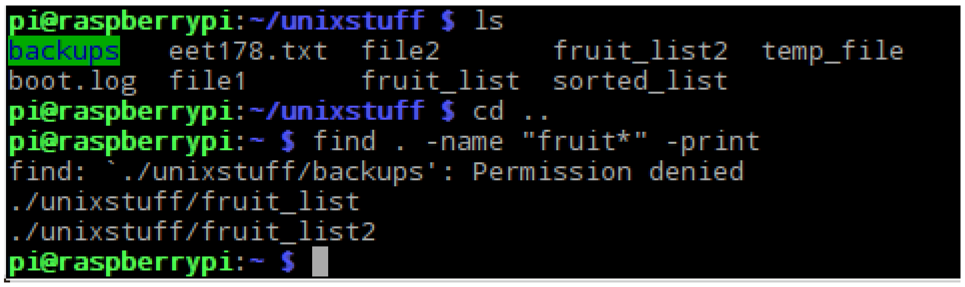
3. File names can have multiple "." symbols in them

4. Files are uncompressed by using the "gunzip"

Finding specific files in a subdirectory

Current working directory should be /home/pi/ (~)

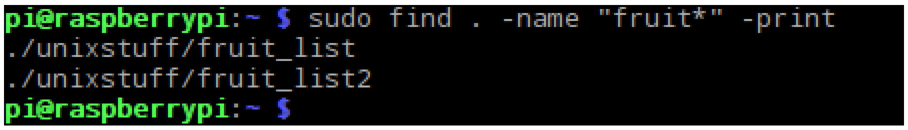
Let's assume you have a file somewhere under your home directory, probably deep in some subdirectory. It will take a long time to search each directory individually. There is a utility called "find" that will help.



```
pi@raspberrypi:~/unixstuff $ ls
backups   eet178.txt  file2       fruit_list2  temp_file
boot.log  file1       fruit_list  sorted_list
pi@raspberrypi:~/unixstuff $ cd ..
pi@raspberrypi:~ $ find . -name "fruit*" -print
find: `./unixstuff/backups': Permission denied
./unixstuff/fruit_list
./unixstuff/fruit_list2
pi@raspberrypi:~ $
```

I changed to the parent directory which was ~ or /home/pi
I told the computer to find
start in the current directory – "."
look for a file with the name "fruit*"
meaning any file that starts with
fruit

Notice I received a "permission denied" message about the backups directory. At a previous step, I had restricted the access permissions and made it unreadable for myself (your directory may be setup differently). This is no problem however and we can make use of the "sudo" superuser. I'll rerun the command with elevated superuser permissions that let me look at any file or directory and the "permission denied" message goes away.



```
pi@raspberrypi:~ $ sudo find . -name "fruit*" -print
./unixstuff/fruit_list
./unixstuff/fruit_list2
pi@raspberrypi:~ $
```

History of commands

The shell keeps track of every command that you've entered. Each command is given a number in order. Unless you change it, the last 1000 commands are recorded. Type:

```
history
```

If you need to retype something, Linux makes it easy. Find any recent command in your list and note its number. Now type, replacing <number> by the number you noted:

```
!<number>
```

You should have typed something like "!185" or "!90". The command should have repeated itself as if you had retyped it yourself.

There's another way to access and run previous commands without retyping. At the prompt, press the up-arrow key. Press the down-arrow key to go back down the list of recent commands.

You can also use the left arrow and right arrow keys to edit a command.

Autocomplete

Current working directory should be /home/pi/ (~)

It's not strictly a command but autocomplete saves a lot of typing. When entering the filename or directory name in commands, you only need to give the first few letters and the shell will try to autocomplete the rest of the name. Try the following:

1. Without pressing enter, type `cd un`

2. Press the tab key and watch as the shell fills in the rest

3. Press enter

There was only one directory that started with the letters "un" so that was easy. Let's find out what happens when there are multiple choices for the autocomplete. Try the following from the unixstuff directory:

1. Without pressing enter, type `more fruit`

2. Press the tab key three times and you should see a list of the choices

3. Complete by pressing enter or typing additional characters to complete the line

Command	Definition

diff <file1> <file2>	looks for any differences between the two files and displays the changed lines.
gzip	compresses a file to save on disk space or time to download
gunzip	uncompresses a file
find <file>	searches through directories for the named file
history	displays the last 1000 commands run
!<command number>	runs a recent command
up-arrow, down-arrow, left-arrow, right-arrow	scrolls up and down through recent commands and allows editing of command lines

Key Concepts

Linux has many utilities for handling and locating files

You can view the history of recent commands

You can easily re-run recent commands and edit command lines

Exercises – Using what you learned about redirection attempt the following

1. Look in the directory /opt/Minecraft-pi/data/images/gui. Using ls with a * wildcard, how many files in the directory start with the letter g and end with the .png extension.

2. Without retyping the command you created above and using history and the arrow keys, edit the command to find files starting with the letter i and ending with the letter g.

3. Copy a large file from the /var/log directory to your local home directory. Compress it with gzip and calculate the compression percentage (how much was the file compressed?

4. Create a small file that contains one word. Find the size, compress the file and then calculate the compression percentage. Do you have any theories about the percentage?

Linux Variables

When you log into a windows computer or your smartphone, something remembers what apps you last ran, what the background image on your desktop is and various bits of information important to the program. Linux does the same thing. It remembers what font size you set the terminal window to, how many recent commands to keep track of and several other settings. These settings are part of the Linux environment and are stored in environment variables. An environment variable is a storage element that keeps track of a value. A value can be a number or a word. When a command or program runs it looks in the environment for associated variables. If it finds them, it uses the values stored in those variables. Some of these values are set by the system, set when the shell starts up, by you or by any other program that runs.

There are two broad categories of variables: Environment variables are set at login and remain valid as long as the computer is running and a user is logged in. Environment variables can affect any running program and be read or written by any program. Most variables are meant to be written by the system and read by programs and are not very informative to the typical user. Shell variables are set when a terminal window is opened (the shell is started by that action) and remain valid only as long as the terminal window is open and the shell is running. Variables set in a shell do not affect any other shells and go away when the shell (terminal window) is closed.

Shell variables are case sensitive. By convention (though programmers break convention often), environment variables are upper case and variables while the shell is running usually are lower case.

Here are some examples of variables

variable	Typical value	purpose
USER	pi	Lets programs know who is logged in
HOSTNAME	raspberrypi	the default name of the system – though you can change it
SHELL	/bin/bash	the active shell (it's not common by there are other types of shells you can run)

To see what variables are defined, type this in the terminal window:

```
printenv
```

```
printenv | grep SHELL
```

The first "printenv" gave you a lot of information and that's a lot to filter through. You used a pipe and the "grep" command the second time and just displayed the variable you are interested in.

You can access a variable directly by using the "echo" command. Type:

```
echo $OSTYPE
```

The "echo" command is very simple program, mostly for displaying the value of single variables. To use it, you have to put a "$" in front of the variable. To set or create a variable you don't need the "$" and you have to use a new command called "export". That command exports the variable to the shell. Try the following:

```
export temp_var = 5
```

```
export temp_var=5
```

```
echo $temp_var
```

Which one worked? You can't have any spaces before the "=" when setting Linux variables.

Command line compiling and running

Current working directory should be /home/pi/unixstuff (~/unixstuff)

Last week, you used the IDE to edit and run a simple program using the IDE. The IDE is just a front end for the editing and compiling process. That process consists of the following steps:

1. Editing the file

2. Compiling the file (turning the human readable program into an intermediate object code)

3. Linking the file (combining the object code with existing library code together and generating machine instructions or assembly code)

4. Running the program like any other linux command you have learned about

Exercises – Compiling and running a program

1. Using nano, create a file called "guess.c" in your unixstuff directory

```
#include <stdio.h>
int main()
{
  printf("Enter your favorite number:\n");
  int num = 0;
  scanf("%d", &num);
  printf("Your favorite number is %d \n", num);
  return 0;
}
```

2. Compile and link the program using this command `gcc guess.c -oguess`

3. Run the program by typing `./guess`

Paths

When you type a linux command into the terminal window, the computer has to find that command somewhere. Linux commands are executable programs that are in a directory on the system. The computer needs to know which directories to search in to find the executable program. Linux maintains a list of directories and that list is called the "PATH".

Type:

```
ls
```

```
guess
```

It didn't work, did it? But you say the file is right there in the directory. Above, when you typed "./guess" you were telling the computer exactly where to find the file to run. You told the computer to 1) look in the current directory (the shortcut is ".", 2) use the separator "/" and 3) run the file there called "guess"

There is a special environment variable called "PATH". Type:

```
echo $PATH
```

You should see a list like this: "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games"

That is the list of directories that are searched. When you type a command, the computer first looks in /usr/local/sbin, then /usr/local/bin, and so on. The ":" is a separator. You can modify that search list. Type the following to add the "unixstuff" directory to the search path:

```
export PATH=$PATH:~/unixstuff
```

```
echo $PATH
```

```
guess
```

That command now should have worked without the need for the "./" . Change to a different directory and try typing `guess`.

The computer searches the new path list, searching your unixstuff directory last, for the guess program and runs it.

Command	Definition
printenv	displays all variables defined in the shell
export <variable>=<value>	creates or modifies a variable and makes it available in the shell (but not other terminal window shells)
echo $<variable>	prints out the value of the environment variable

Key Concepts

Programs communicate with other programs by setting and reading shell variables

Shell variables are used a lot to configure how the system and programs operate

You can set shell variables from the command line

Exercises – Compiling and running a program

1. Assuming you set the path variable correctly, run your guess program from any directory.