Linux Terminal Instructions (Ubuntu-flavoured)

The aim of this sheet is to get you happy with Linux command line and to make your web server run on your Linux box (and see it running). If you are already familiar with Linux, go ahead and install the things that you need (git, node…at least), grab your website (from the last lab) from GitHub and serve it using node. If you want to be really clever, put all the right instructions into a .sh file and run them all with one command. **history** is one of the most useful commands.

Many servers that you find online run on Linux, and you may only be able to access them using ssh. It is important that you can do things on Linux computer via a terminal. This lab sheet will walk you through some basics. Run them on your terminal to see what they do. **Develop a frequent habit of typing a few letters and then hitting the "tab" key to see your options. This goes mainly for paths in the system.**
"Directory" is the proper word for "Folder".
ctrl+c will usually get your cursor back.

This lab sheet is mostly a cheat sheet. Instructions to help you learn are in red.

pwd

Stands for Print Working Directory. Tells you exactly where you are on the system.

ls

Stands for List. Lists the contents of the specified directory. More useful is **ls -al** which lists (in a list) all the contents in a directory, including the hidden ones, and gives their permissions.

whoami

Tells you your own username. Useful when you're trying to work out exactly where this particular cursor is.

man

Stands for **man**ual. You can run it with any command and it will tell you what that command does. Used like **man echo** to get the manual for the echo command. If you get stuck inside a man page, scrolling and scrolling, **q** will get you out.

When you first log in to your linux box, you will be in your home directory (use pwd to find out where that is), and it will look empty. See what files you can find in your home directory, and check your username.

cd

Stands for Change Directory. **cd ..** means go up one directory. **cd ~** will always take you home. You can put any path after it, hitting the tab key will tell you which tabs are available. You will not be able to go to directories that you don't have access to (unless you use **sudo**)

Have a look around your Linux computer using cd. Go up the directories as far as you can go (to "root" or /) and then return to your home directory. Look at the names of the directories and what goes in them. What is in /var? What is in /etc? What is in /users?

echo

Just a print statement.

>

STDOUT redirection. Means "pipe output into and overwrite". >> means "pipe output into and append".

cat

Stands for con**cat**enate. But really it just reads the contents of a file and puts it to STDOUT (i.e. displays it in your terminal in a big dump of all the text). Useful for viewing the contents of short files without changing them.

Using echo and >, create a file called hello.txt whose contents are "Hello Wally". Check you've managed this with ls to list all your available files and cat to view the contents of the actual file.

mkdir

Means Make Directory. Makes a directory at the specified path (provided you have permissions to do so).

cp

Means copy. Use it like cp file1.txt file2.txt to copy the contents of file1.txt to file2.txt

mv

Means move, but it's really a copy followed by a delete. Use it like mv.

rm

Stands for remove. Can be used to delete files (and directories using **rm -Rf**) but use it with care because you're never getting those things back.

Use the tab key to help with paths. Use mkdir to make a new directory called temp. Use cp to copy hello.txt and call the new file hello2.txt. Move hello2.txt into temp. Remove the temp directory.

history

Shows all the commands that you've ran recently. Useful for remembering complicated instructions. Use it along with **grep** when your history gets too long (like **history | grep echo** to look for the last *echo* command in your history – replace *echo* with anything you like to look for). Pipe it into a file using **>** (like **history > history.txt**) for safe keeping and editing later if you want to repeat what you've done.

Save your history to a file called “myFirstHistory.txt”. Look in your history for some of the commands that you wrote.

|

This is a pipeline command. It means pipe the output from the first command into the second command. Can be used for counting words or lines in a file. Often used with **grep** as the second command.

grep

Means find. Like ctrl+f in a document editor. Used at the end of a pipe. Can work miracles if you can use regex statements, but also good at just matching a single string or using with a **-v** option to match everything *except* that string.

Find Wally. You put him in a file earlier. You should now be able to find him in at least 3 different places (increasing as you look for him).

Installing Stuff

You want to set up your new server machine to serve your web page. You will need to install some things.

which

Used with a command as the parameter (like man). Tells you where the actual executable lives. Returns nothing if the command is not installed. Use it like **which git**.

Check to see if git and node are already installed.

sudo

Means Super User Do! Users in Linux have permissions. Normally the root user has all the permissions all the time. This is dangerous. Ubuntu uses sudo instead of having a root user. Whenever you want to do something that requires elevated permissions (like installing stuff), you prepend the instruction with sudo.

apt

Means Advanced Package Tool. Used for installing things (like brew on a mac). To install things you will need to know the current name of the tool you want to install. And you will probably need sudo. See the instruction below for an example.

Install git using the following commands
sudo apt-get update
sudo apt-get install git-all
Check that it has installed using which, and then **git version** to see if it works.
Install node and npm similarly (the packages are called **nodejs** and **npm**). Check the installation.

Which version of node, exactly?

The version of node is important. Make sure you install the latest version of node. Find out what the current node number is and make sure that's the version you've installed. If you do:

sudo apt install nodejs
node --version

Then you will see that version is quite old. To install a newer version (v21), use:

sudo apt install curl
curl -fsSL https://deb.nodesource.com/setup_21.x | sudo -E bash
sudo apt-get install -y nodejs

Grabbing and running your web server

You should have a GitHub from last week (or you can use the solution for last lab). You should:

1. Make a directory for your website
2. Use git clone to pull it on to the linux box
3. Change to the appropriate directory (where package.json and ReadMe.md is)
4. **npm install** all the node modules
5. Start up the server with node (same as last week). **Note that port 8080 is a port that *will* work (others might not).**
6. Check that you can see it running on the server (browser and localhost or 127.0.0.1)
7. See if you can browse to it from another machine on the network (i.e. the one in front of you). **Note: for this you might need to poke open a port in the firewall (see below).**
8. Fix any errors along the way. You should be looking at both front- and back- end code. If you're feeling adventurous, and you don't have any bugs to fix, add some console.log statements to any JavaScript (in the html file or in the server file) and see where you can see the output (on the Linux terminal or in the developer tools on your browser?)
9. Commit any fixes back to GitHub.
10. If it all works, use your history command to create a script that will do what you just did on a new machine. This will be useful.
11. (optional) Check your script doesn't contain anything sensitive.
12. (optional) Commit your script to GitHub (seriously, this could be a passing part of your coursework!).

I want my prompt back!

When you run a node server, the command prompt disappears. Most of you will just use ctrl+c to quit the server or spin up another terminal. But if you're going to be starting several processes from the same script, you might want to know a few more commands.

&

This is the background option. If you add **&** to the end of any command, the command will execute in a subprocess in the background. The real trick is stopping the process again.

ps

Means processes. Lists all the processes running on the machine. Can also be used with arguments (e.g. **ps -a**). You can pipe the output into grep if it all feels too much.

top

A nicer version of ps, maybe. Use **q** to get out of it.

pkill

A way to kill all the processes of one type. For example, **pkill python** would kill all the python scripts running on a server (that the user had permission for).

kill

A way to kill a process by process number. You need to find the number from **ps** first.

Spin up your server again but put it into a sub-process using **&** at the end of the line. Check that it's running using a browser somewhere. Now locate and kill the process to stop it running. Check that it has stopped. If you are working on a .sh script, you might consider having the & inside the script and outside of the script – does it make any difference right now?

There is a better way to do this using daemons (and, actually, Nginx, because it's running a full production server rather than a node development server), but it involves more installs. See:
https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-20-04

A Firewall on the Computer?!

Yes, there is a firewall. You can see your website ok from localhost, but when you browse to it from another machine on the same network, it won't load (and you *know* your IP address is ok because it sits there for ages and doesn't return immediately with a host unknown error). You can poke a hole in the firewall using (for example, for port 8080):
sudo ufw allow 8080/tcp
Be careful because it's sudo and it's adding a rule to the firewall. My port was 8080 but you could replace it with your own choice in port. In a real world setting, you would use a port specific to a protocol, or set up a proxy in your server software.