Linux Basics

Like **Windows** and **MacOS**, Linux is an operating system. An *operating system* connects the software to the hardware on a computing system of any kind - laptop, desktop, mobile device, virtual machine, etc.

Linux is open-source, so it can be run for any purpose, studied to learn how it works, modified and distributed. Therefore, many official **distributions** of Linux have been developed over the years, such as

- Alpine
- Debian
- CentOS
- Fedora
- Red Hat
- Ubuntu
- and many more (over 100!)

Some distributions are optimized for desktops; others, for servers. Luckily, many commands are the same or similar across the various distributions.

In this lesson, you will learn or review commands which you may find helpful while working with **Linux** in a Docker container or on a virtual machine. These are especially useful for debugging when you are not seeing the behaviors you expect. Debugging using the command line is an invaluable skill that you've already started to learn, and will continue to grow with practice.

Where is Linux found?

First, **Linux** can be the primary operating system (OS) for a computer. Most of the hosted and cloud-based services run some distribution of Linux. Some developers run Linux on their personal laptop or desktop computers. You are probably running *Windows* or *MacOS* as the operating system on your computer, like most home and business users.

Windows 10 comes with a tool called **WSL**, or Windows Subsystem for Linux, which allows users to run Linux commands from a terminal window. Modern versions of the *MacOS* are built upon a Unix framework. **Unix** is also the framework that each distribution of Linux has

been built upon.

Like its name suggests, a **virtual machine**, or **VM**, is a way to run an operating system within another operating system on the same physical hardware. Each VM runs separately and securely. VMs can run any distribution of Linux, as well as other operating systems like *Windows* or *MacOS*.

A **Docker Container** is a form of Virtual Machine which also allows for easy bundling of software with an operating system. The **Docker Engine** running on the underlying OS allows containers to share the system resources for higher efficiency and reduced cost. The standards developed by Docker allows developers to distribute software to other developers, servers and even into the cloud with reliability and ease.

You will learn more about **Docker Containers** and **Virtual Machines** in future lessons.

Emergency escape route

Before you start playing with commands, it's good to know how to get yourself out of trouble. In Linux (as well as nearly every OS on the planet), holding `control` (a.k.a. ctrl) and pressing the `c` key will stop any command that's running whether it is actually finished or not. It is the command-line form of "force stop" (e.g. Android) or "force quit" (e.g. Mac) which you may be familiar with from your mobile device or computer (on Windows it's Ctrl+Alt+Delete).

Don't be afraid to press `ctrl+c` multiple times. For example, if a script is running then there might be multiple commands you need to stop.

Shells

While this could be a whole topic on its own, it is sufficient for now for you to know that there are several shells or interfaces in the terminal which you are likely to see in Linux. The three most common are `bash`, `zsh` and `sh`. If you pay attention as you work in VS Code, Docker and VMs, you'll see one (or several) of these pop up from time to time.

Now it's time to get into the commands. While you many have seen some of these before, it is worth taking a moment to make them fresh in your mind. You may discover a new twist or tip in the process!

`ls`

To *list* the files in a folder, use the `ls` command. If no path is given, then the contents of the current path are displayed.

Additionally, you can reference an absolute path (`/usr/local`), a relative path (`somefolder`), the parent folder (`..`), or the logged in user's home directory (`~`).

The most common flags to use with ls are `-l` for producing list with many columns of useful information about the files / folders found and `-a` to output all files / folders including the current folder (`.`) and the parent folder `..`.

If you'd like just 1 command to remember, then you may choose `ls -la` or `ls -al` (as they do the same thing).

cd

To *change directory*, use the `cd` command. Give it the path where you want to go. As before, the path can be absolute (starting with a `/`), relative (NOT starting with a `/`), parent (`..`), or home (`~`). If no path is provided, then you will end up in the home directory.

Some distributions of Linux allow you to use the TAB key to auto-complete folder names in a path.

pwd

To *print* the *working directory*, use the `pwd` command. This can help if you are unsure which folder you are currently in.

echo

To see the value of a variable that is set in the OS environment, *echo* is the quickest option. For example, `echo $PATH` will show you the list of folders that the shell will use to look for commands you enter.

ps

It can be very useful to know what processes are running on a machine or in a Docker container. This can be achieved by looking at the *process status* using `ps`. The `-a` flag will show processes run by all users. The `-x` flag tells `ps` to include processes that are running

without a terminal, such as daemons running in the background or started when the system launched. The `-u` flag provides additional detail about each process.

Different distributions of Linux support different flags. The three most common use cases are

- `ps -aux`
- `ps -ax`
- `ps -a`

It will not harm either Docker or a virtual machine to try the longest variation and shorten as needed to match what is supported.

`less` and `more`

Two variations of displaying long outputs in the console are `less` and `more`.

While there are a number of subtle differences in implementation, the most notable is that ON SOME FLAVORS OF LINUX `more` can only move down through the content while `less` can move both up and down.

The up and down arrow keys as well as page up and page down may be used for the movement. Again, with subtle differences depending on the flavor of Linux.

To view a file you may use a command like `more README.md` or `less npm-debug.log`. Like other commands, absolute and relative paths may also be used when specifying a filename.

Piping

In the Linux command line, there is a special character used to chain commands together where the output from one is sent to the next. This character is the pipe; that is `|`. The `less` or `more` command will add paging to the output that is piped to them.

For example, `ls -la /usr/local/bin | less`.

`cat`

Yet another way to view a file - this time without paging - is `cat`. For example, `cat /etc/hosts`.

The name of this command comes from its functionality - con*cat*ination. This means you can include multiple files in order to output them back to back. A useful case for this is concatenating several files into a single output file.

```
cat file1.txt file2.txt > combined.txt
```

The > operator creates or replaces the contents of the output file. Using the >> operator will append to the output file. So putting a third file on the end of *combined.txt* would look like this:

```
cat file3.txt >> combined.txt
```

`tail`

If you like to monitor a file for new writes to the end, then `tail` is the command you seek. Specifically with the `-f` flag meaning *follow*. By default, `tail` outputs the last portion of the file. You may control how many lines using – followed by a number; for example, `tail -10 README.md` would output the last 10 lines of the README to the console.

Log files are particularly useful to follow with zero lines of what's already in the file so you only see what happens AFTER you start tailing.

```
tail -0 -f my-app.log
```

`vi` or `vim`

The classic editor in Linux is `vi`; it's newer (relatively speaking) cousin is `vim`. Both are keyboard controlled only, not visual as other tools you know (like **VS Code**). That means you need to learn additional commands to use them. IT would take multiple articles to cover all possibilities, so here are just a few necessities if you find yourself using one of these.

Open a file with `vim file.txt` or `vi file.txt`.

Searching by type slash (`/`) followed by the text you'd like to find. It is case-sensitive. Hit `ENTER` to move the cursor to the found text. You may type `/` followed by `ENTER` to repeat the same search.

Type `i` to enter *insert mode*. Or `o` to create a new line and enter *insert mode*. After typing what you need to add, then use `ESC` to leave *insert mode*.

Type `x` to delete one character. Type `dd` to delete a whole line. While in *insert mode*, `vim` allows you to use the `delete` key to remove characters (standard behavior).

Type `G` to jump to the end of a file.

Type `:` to enter commands. For example, `:w` saves the file (that is, *write*) and keep `vim` or `vi` running. The `:q` command quits. Worth memorizing are `:wq` which writes and closes the file, and `:q!` which closes the file without saving.

One of the reasons developers choose `vim` over `vi` when av available, is because `vi` may input special characters for arrow keys when in edit mode which can be frustrating to delete. However, `vim` will allow you to move up and down lines without leaving insert mode.

> Tip: When merging in `git`, it is likely that `vi` or `vim` will be the editor that opens automatically for you to modify or confirm the comment.

nano

Another editor is `nano`. It also uses the keyboard exclusively; however, `nano` relies on *control* plus the appropriate key for saving, searching, closing the file, and other commands. A list of commands will show at the bottom of the terminal when `nano` is launched. Because of this, `nano` is in edit mode all the time. Just type whatever you want to change.

To edit a file in nano, simply give it the filename, e.g. `nano README.md`. In most versions, `control+x` will close the editor. If you've made changes, you will be promoted to save (or not).

grep

When you want to find where a certain piece of text shows up, use `grep`. This will show you every line where your search text appears in a file. You may include the `-R` flag to recursively search a directory.

For example, to find all occurrences of "academy" in the readme, use `grep academy README.md`. And if you'd like to search for a longer phrase, then enclose it in quotes; for example `grep "npm start" package.json`.

You may also recursively search a directory using `grep`. For example, to see all occurrences of "localhost" in the files under */etc*.

```
grep -R localhost /etc
```

Remember you can use the pipe to send the output to one of the paging commands.

```
grep -R localhost /etc | less
```

find

The `find` command is extremely powerful and allows for many complex operations. However, you are likely to need it most often to find a file by name. That means using the `--name` flag. For example, to find *package.json* somewhere under the current folder, run the following.

```
find . -name "package.json"
```

You can also use wildcards (e.g. all *json* files).

```
find . -name "*.json"
```

Or make the search case-insensitive (e.g. all files with 'readme' of "README" anywhere in the name).

```
find . -iname "*readme*"
```

You can use any absolute or relative path as the starting point for the search (in place of `.` which means the current folder).

chown

Every file and folder is owned by a user and a group. You can view the current ownership using `ls -l` (or `ls -la`). The third column is the user; the fourth, the group. Sometimes you'll need to change the ownership of a file or folder in order for your program to access it. This is done with the `chown` command.

For example, `chown crb:staff file.txt` will set the owning user to "crb" and the group to "staff" for *file.txt*. Using the `-R` flag will recursively change the ownership of all files and subfolders of a folder. For example, `chown -R ubuntu /opt` will set both the user ownership to "ubuntu" for a folder called */opt* and all the subfolders and files therein.

chmod

Other way to control access to files is with the read, write and execute permissions. Again you can view the current settings using `ls -l` (or `ls -la`). The first column is a collection of permissions indicated by 10 characters.

First character is `d` when the item is a folder (a.k.a. directory). The next three characters indicate the read (`r`), write (`w`) and execute (`x`) permissions of the owner (user). The next three are read, write and execute by the group. The final three characters are the same for all users on the system.

A directory must be executable in order to `cd` into it so you will notice most folders show `drwxr-xr-x`. That means it is a directory that can be read, written and entered by the owner, but not written (modified) by users in the group or other users on the system. Most files will show `-rw-r--r--` which means they can be read and written by the owner, but only read by other users.

If you have a shell script, then you may need to make it executable. The command for this would look something like `chmod +x your-script.sh`. Knowing how to make a shell script executable will be useful when working with Docker in a future lesson.

Some files - like private keys - need to be hidden by users who are not the owner so that their permission needs to look like `-rw-------`. The easiest command for this is `chmod 600 private.pem`.

As you can see, `chmod` is another Linux command with many options. Too many to discuss in this short article. The good news is that you can easily find what you need with a little online research when the need arises.

man and -h

To open the manual for a command, you can use `man`. For example, `man chmod` will let you study the full usage and options for setting file permissions.

Some commands include the flag `-h` or `--help` which will show you the basics of how to use the command and what flags it supports. For example, `vim -h` or `bash --help`.

You should not be afraid to try flags on a command. The worst thing that happens is you see a warning message and list of supported flags. For example, `chown -h` may show something like this:

```
usage: chown [-fhnv] [-R [-H | -L | -P]] owner[:group] file ...
       chown [-fhnv] [-R [-H | -L | -P]] :group file ...
```

When you see this, you can then use `man chown` to view the full explanation.

Package Managers

The final topic worth mentioning is the management of what is installed on a Linux system. This is controlled through a *package manager*. Each flavor of Linux has a particular package manager that you will need to use. Some examples include:

- `apt` - the Advance Package Tool is used by Debian and Ubuntu
- `yum` - the Yellowdog Updater Modified is used on Red Hat
- `apk` - the Alpine Package Keeper used by Alpine Linux

You will learn more about package managers as you go more in depth on a particular version of Linux. For now, it is sufficient to be aware that you will need to pay attention to the version of Linux in your Internet searches.

What you've learned

Now you have a number of tools to help you work proficiently in Linux.

- Recognize common shells: `bash`, `zsh`, and `sh`
- Stop any command using `ctrl+c`
- Navigate with `ls`, `cd`, and `pwd`
- Check environment variables using `echo`
- Verify processes are running with `ps`
- View files using `less`, `more`, `cat`, and `tail`

- Make minor edits using `vi`/`vim` or `nano`
- Run basic searches with `grep` and `find`
- Set access permissions using `chown` and `chmod`
- View built-in documentation using `man` or `-h`
- Recognize common package managers: `apt`/`apt-get`, `yum`, and `apk`

See also...

If you'd like to read more about Linux in general, the "What is Linux (https://www.linux.com/what-is-linux/)" article on linux.com is a great resource.

If you are interested in learning a little more about shells, here's a short article on the differences between bash and zsh (https://dev.to/jasmin/a-brief-difference-between-zsh-and-bash-5ebp).

You can also learn more keyboard commands for navigating with less (https://www.linode.com/docs/quick-answers/linux/how-to-use-less/).

Or perhaps you'd like to read more about cat (https://linuxize.com/post/linux-cat-command/), view a reference sheet for vim (http://tnerual.eriogerg.free.fr/vimqrc.html), or dig into more technical details on ps (http://www.linfo.org/ps.html).