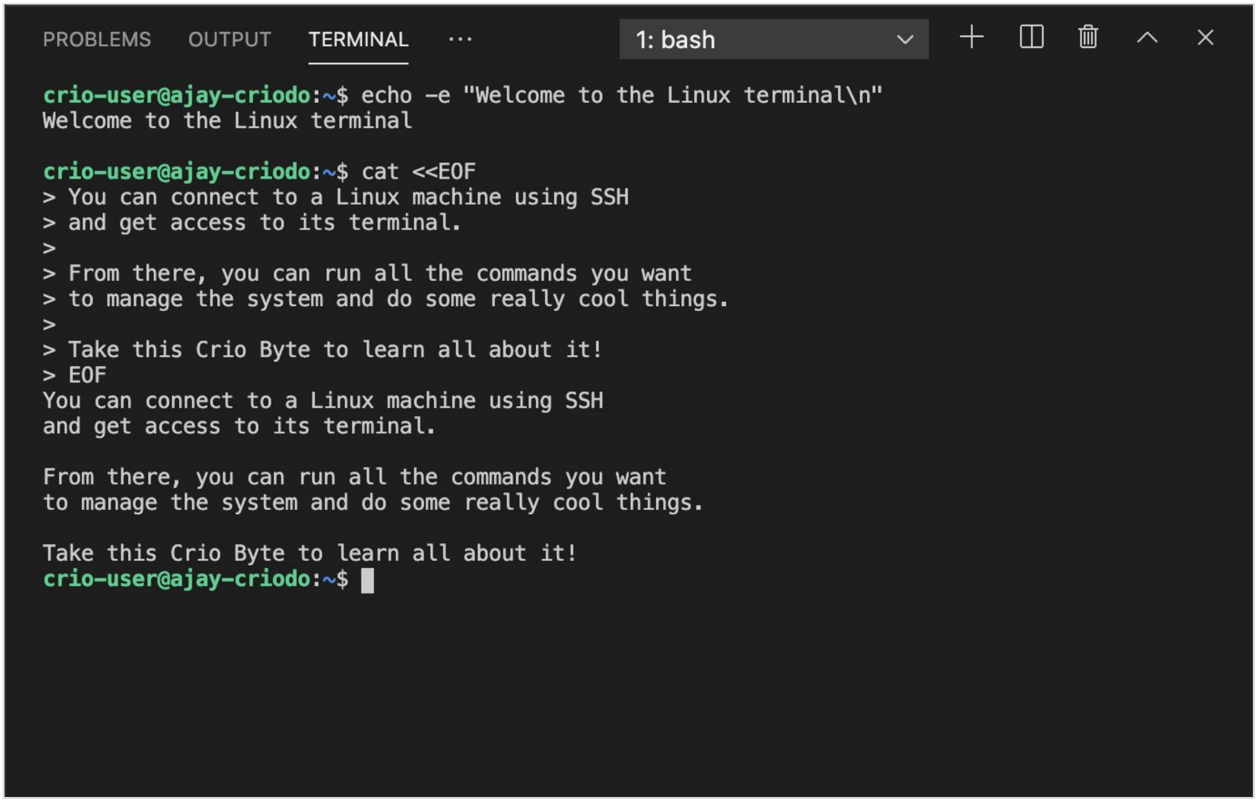
Objective

Get started with the Linux terminal & begin using terminal commands

Background

90% of public cloud computing services run Linux. A competent software developer must know how to use a Linux system comfortably.

When you create your own Linux virtual machine (VM) from services like GCloud, AWS or Microsoft Azure, you don't usually get access to a Graphical User Interface (GUI). You have to use the Linux terminal to operate and manage your VM.



```
crio-user@ajay-criodo:~$ echo -e "Welcome to the Linux terminal\n"
Welcome to the Linux terminal

crio-user@ajay-criodo:~$ cat <<EOF
> You can connect to a Linux machine using SSH
> and get access to its terminal.
>
> From there, you can run all the commands you want
> to manage the system and do some really cool things.
>
> Take this Crio Byte to learn all about it!
> EOF
You can connect to a Linux machine using SSH
and get access to its terminal.

From there, you can run all the commands you want
to manage the system and do some really cool things.

Take this Crio Byte to learn all about it!
crio-user@ajay-criodo:~$
```

The Linux terminal

Some of the advantages of the Linux terminal include:

1. The ability to create and run scripts in several languages (Bash, Python, Perl, and more) right from the command line.
2. Easily automate several workflows that are much harder to do in a GUI.
3. The network bandwidth required to access a Linux system via terminal is far lesser than that required by a GUI.

In this Crio Byte, you will familiarize yourself with the Linux terminal starting with some simple commands and proceed to learn commonly used developer workflows.

Primary goals

Here are a list of things you will learn in this Crio Byte:

1. Get familiar with the Linux terminal.
2. Learn the commands necessary to navigate directories and files.
3. Perform file management operations - create, read, update and delete.
4. Search directories for files and search files for interesting patterns.
5. Learn output redirection where you send the output of a command to a file (or as the input to a different command)
6. Performing **Data Analysis of Hadoop log files** right from the terminal and realize the power of Linux.

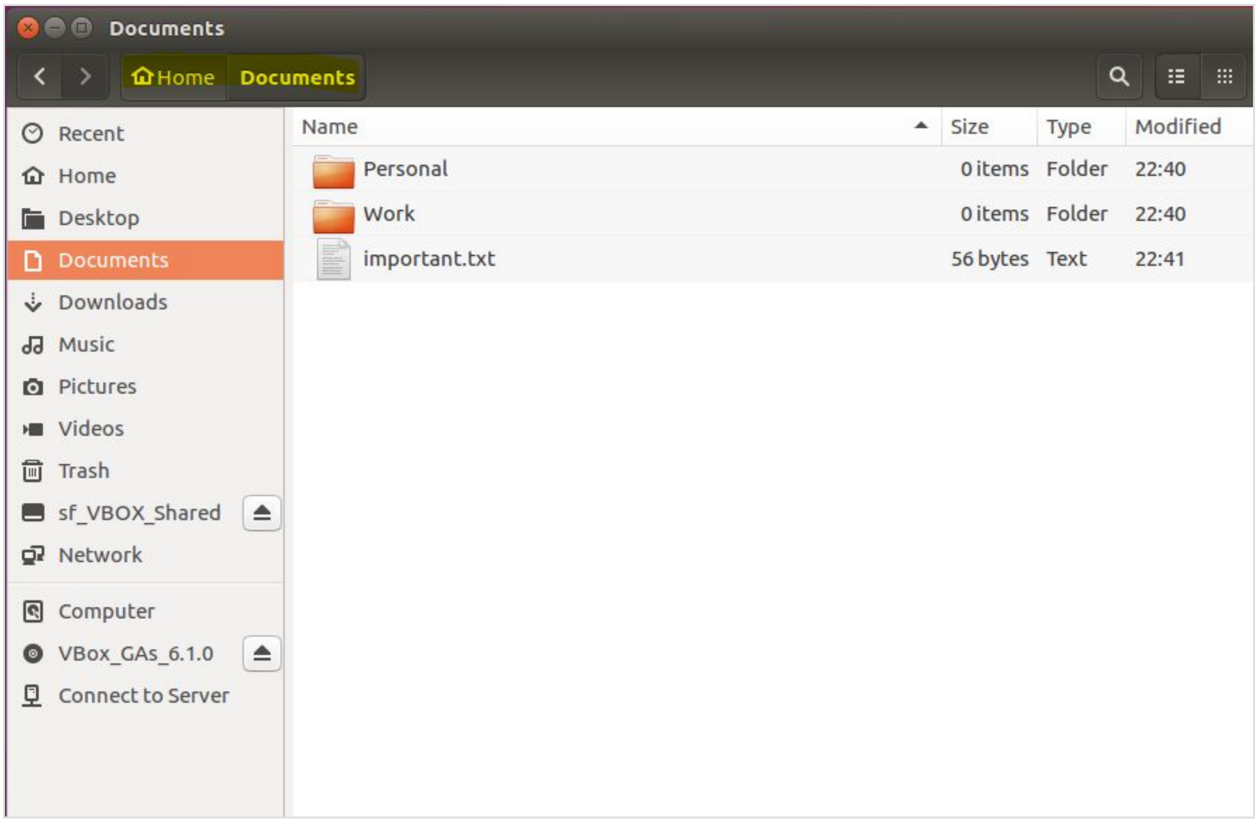
References

1. Why learning Linux command line is important?
2. 6 interesting funny commands in Linux
3. Why you need Terminal?
4. Living entirely in a terminal for 30 days

MILESTONE #1

What info does the GUI give us?

Let us see how the functionality provided by the GUI can be performed using the terminal.



Name	Size	Type	Modified
Personal	0 items	Folder	22:40
Work	0 items	Folder	22:40
important.txt	56 bytes	Text	22:41

The Linux file explorer as seen in the GUI

Can you examine the screenshot of a file explorer window above and answer the following questions?

1. What is the name of the current working directory (a.k.a. present working directory)?
2. Which is the immediate parent directory?
3. Can you list the names of files and sub-directories in the present working directory?
4. Which file has the largest size?
5. Which file was modified most recently?

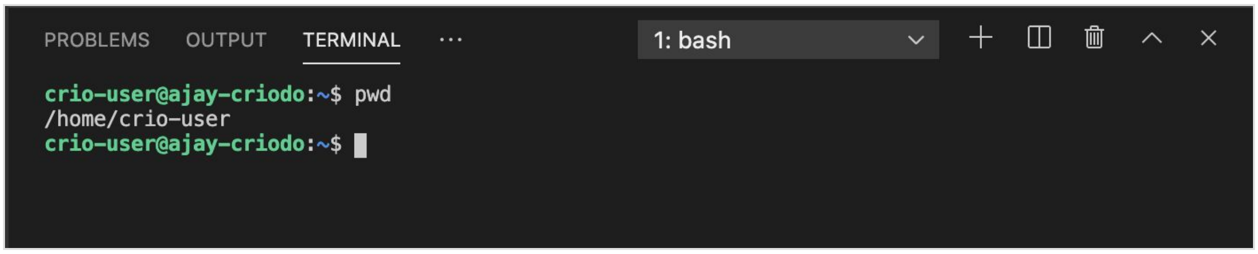
Pretty simple, right? Here are the answers for your reference:

1. The present working directory is `Documents`.
2. The immediate parent directory is the users `Home` directory (Note: `Home` is a special directory. We will learn more about this later).
3. There is one file: `important.txt` and two sub-directories: `Personal` and `Work`.
4. `important.txt` has the largest size (the others have size = 0 bytes since they are directories).
5. `Personal` and `Work` have the same last modified time, however, `important.txt` appears to have been modified most recently.

How do you get the above info using the Linux terminal? Below are a few commands to get you started.

1. pwd

You can use the `pwd` command to **p**rint the **w**orking **d**irectory..



```
crio-user@ajay-criodo:~$ pwd
/home/crio-user
crio-user@ajay-criodo:~$ 
```

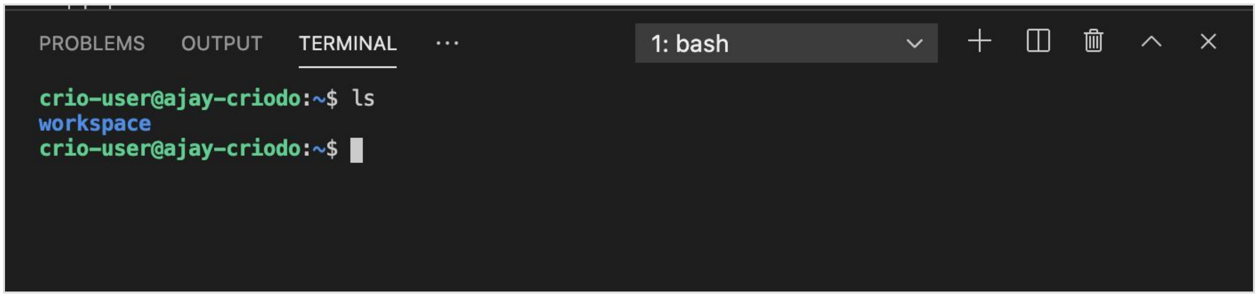
The present working directory is `/home/crio-user`. Each linux user gets their own home directory. All user home directories are in `/home`. The username here is `crio-user` and hence the "**Home**" directory of `crio-user` is `/home/crio-user`.

Note

In Linux, directories are separated by the forward slash `/` (see the `?` key on your keyboard). This is different from Windows where the backslash `\` is used to separate directories.

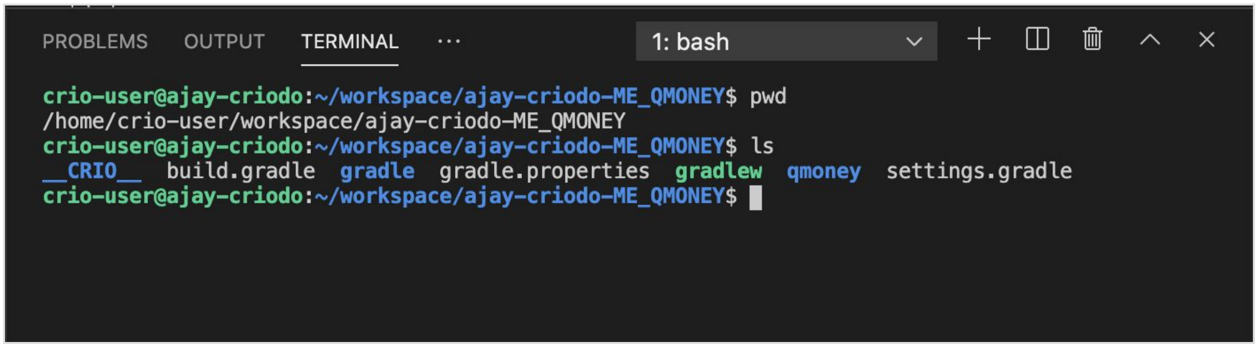
2. ls

To list the contents of a directory, you use the `ls` command (short for **list**). When you run the `ls` command without any arguments, it lists the contents of the present working directory by default.



```
crio-user@ajay-criodo:~$ ls
workspace
crio-user@ajay-criodo:~$ 
```

The output of the `ls` command is formatted differently based on the file type.



```
crio-user@ajay-criodo:~/workspace/ajay-criodo-ME_QMONEY$ pwd
/home/crio-user/workspace/ajay-criodo-ME_QMONEY
crio-user@ajay-criodo:~/workspace/ajay-criodo-ME_QMONEY$ ls
__CRIO__  build.gradle  gradle  gradle.properties  gradlew  qmoney  settings.gradle
crio-user@ajay-criodo:~/workspace/ajay-criodo-ME_QMONEY$ 
```

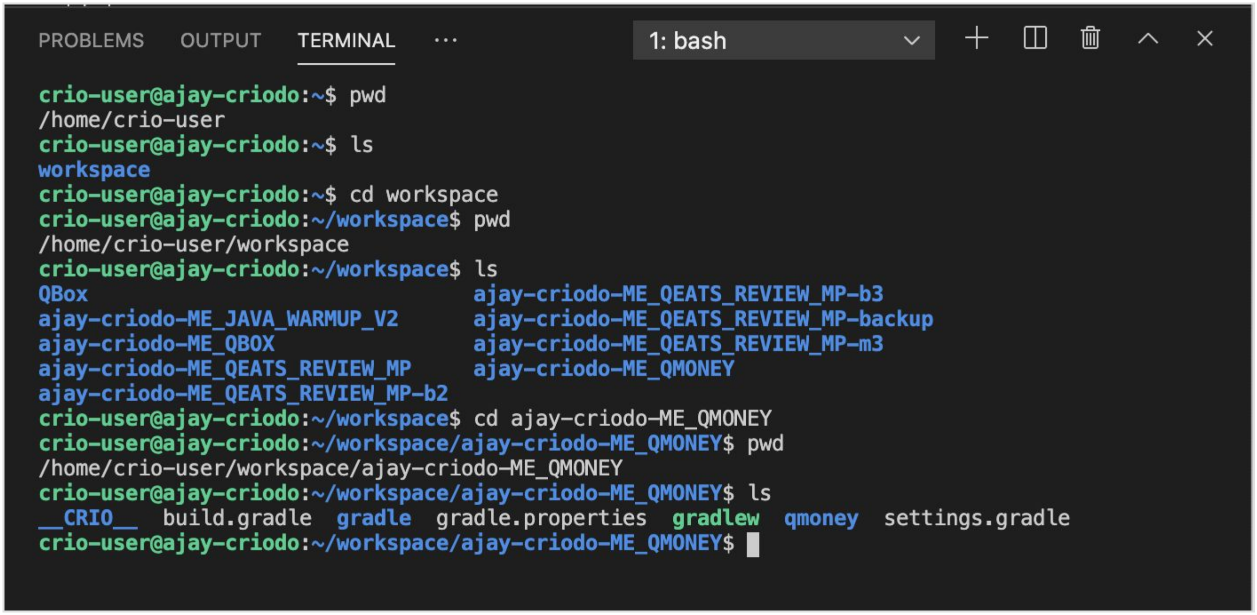
In the above example,

1. Directories in blue
2. Files are in white
3. Executables are in green (we will learn more about executable files later)

Did you notice in the previous screenshot that the present working directory is different from the earlier example? How did we go from `/home/crio-user` to `/home/crio-user/workspace/ajay-criodo-ME_QMONEY`?

3. cd

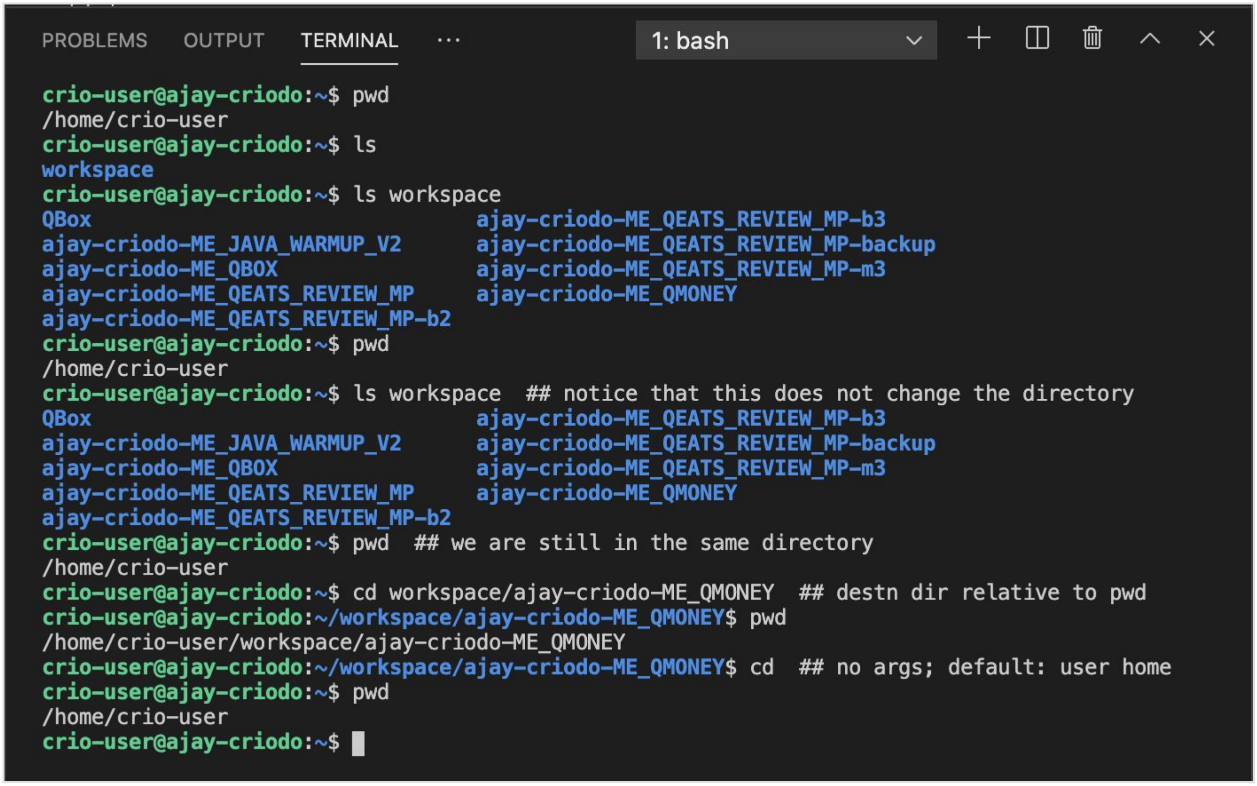
You can change to a different directory using the `cd` command (short for **c**hange **d**irectory).



```
crio-user@ajay-criodo:~$ pwd
/home/crio-user
crio-user@ajay-criodo:~$ ls
workspace
crio-user@ajay-criodo:~$ cd workspace
crio-user@ajay-criodo:~/workspace$ pwd
/home/crio-user/workspace
crio-user@ajay-criodo:~/workspace$ ls
QBox                             ajay-criodo-ME_QEATS_REVIEW_MP-b3
ajay-criodo-ME_JAVA_WARMUP_V2    ajay-criodo-ME_QEATS_REVIEW_MP-backup
ajay-criodo-ME_QBOX              ajay-criodo-ME_QEATS_REVIEW_MP-m3
ajay-criodo-ME_QEATS_REVIEW_MP   ajay-criodo-ME_QMONEY
ajay-criodo-ME_QEATS_REVIEW_MP-b2
crio-user@ajay-criodo:~/workspace$ cd ajay-criodo-ME_QMONEY
crio-user@ajay-criodo:~/workspace/ajay-criodo-ME_QMONEY$ pwd
/home/crio-user/workspace/ajay-criodo-ME_QMONEY
crio-user@ajay-criodo:~/workspace/ajay-criodo-ME_QMONEY$ ls
__CRIO__  build.gradle  gradle  gradle.properties  gradlew  qmoney  settings.gradle
crio-user@ajay-criodo:~/workspace/ajay-criodo-ME_QMONEY$ 
```

Did you notice the difference between the `cd` command and the `pwd` and `ls` commands you used earlier?
The examples shown in the screenshot pass an argument to the `cd` command. In this case, the destination directory was passed as the argument.

The destination directory can include a full path or a relative path (see the screenshot below for an illustration). This is true of the `ls` command as well.



```
crio-user@ajay-criodo:~$ pwd
/home/crio-user
crio-user@ajay-criodo:~$ ls
workspace
crio-user@ajay-criodo:~$ ls workspace
QBox                                ajay-criodo-ME_QEATS_REVIEW_MP-b3
ajay-criodo-ME_JAVA_WARMUP_V2       ajay-criodo-ME_QEATS_REVIEW_MP-backup
ajay-criodo-ME_QBOX                 ajay-criodo-ME_QEATS_REVIEW_MP-m3
ajay-criodo-ME_QEATS_REVIEW_MP      ajay-criodo-ME_QMONEY
ajay-criodo-ME_QEATS_REVIEW_MP-b2
crio-user@ajay-criodo:~$ pwd
/home/crio-user
crio-user@ajay-criodo:~$ ls workspace  ## notice that this does not change the directory
QBox                                ajay-criodo-ME_QEATS_REVIEW_MP-b3
ajay-criodo-ME_JAVA_WARMUP_V2       ajay-criodo-ME_QEATS_REVIEW_MP-backup
ajay-criodo-ME_QBOX                 ajay-criodo-ME_QEATS_REVIEW_MP-m3
ajay-criodo-ME_QEATS_REVIEW_MP      ajay-criodo-ME_QMONEY
ajay-criodo-ME_QEATS_REVIEW_MP-b2
crio-user@ajay-criodo:~$ pwd  ## we are still in the same directory
/home/crio-user
crio-user@ajay-criodo:~$ cd workspace/ajay-criodo-ME_QMONEY  ## destn dir relative to pwd
crio-user@ajay-criodo:~/workspace/ajay-criodo-ME_QMONEY$ pwd
/home/crio-user/workspace/ajay-criodo-ME_QMONEY
crio-user@ajay-criodo:~/workspace/ajay-criodo-ME_QMONEY$ cd  ## no args; default: user home
crio-user@ajay-criodo:~$ pwd
/home/crio-user
crio-user@ajay-criodo:~$ 
```

Note

The pound sign `#` is used to write comments. Everything after the first `#` is ignored. Did you notice the comments in the screenshot above?

Default arguments

1. What if you execute the `cd` command without any destination? Can you try it and see where it takes you? (Hint: Use `pwd` to check where you were taken to)
2. Can you try the `ls` command with a directory name?
3. Experiment with the `cd` and `ls` commands with different arguments.

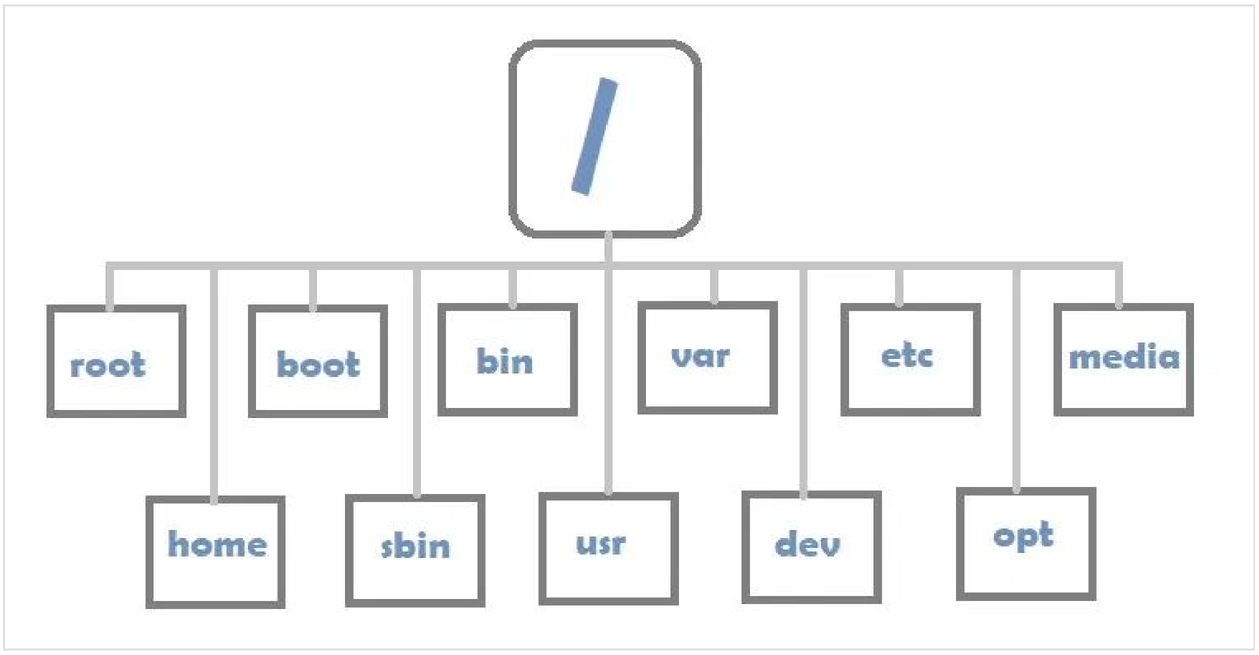
Note

1. If you don't provide any argument to the `cd` command, it defaults to the user's home directory. The `ls` command defaults to the present working directory.

2. There is a symbol to denote a user’s home directory. The tilde (`~`) sign.
3. `cd ~` will take you to your home directory.
4. `ls ~` will list the contents of your home directory.

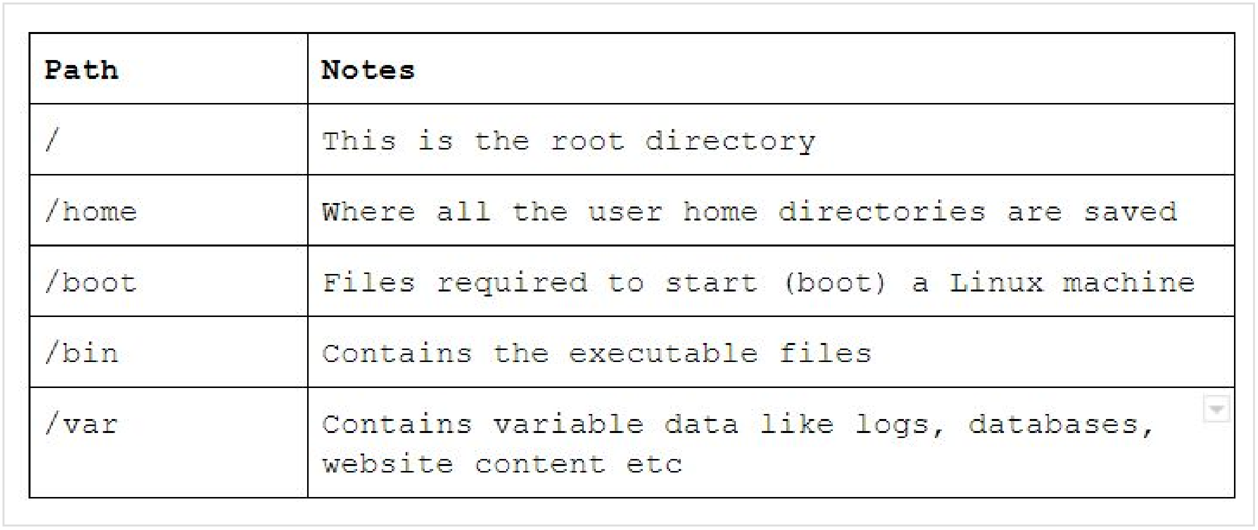
MILESTONE #2

The Linux Directory Structure



Source: https://iotbyhvm.ooo/linux-basic-commands/

Let us take a moment to understand the naming convention above.



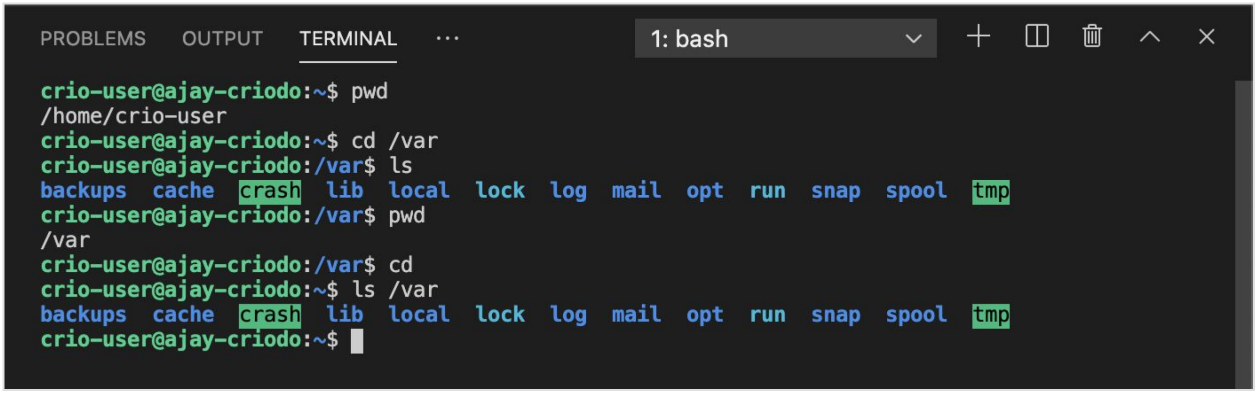
Path	Notes
/	This is the root directory
/home	Where all the user home directories are saved
/boot	Files required to start (boot) a Linux machine
/bin	Contains the executable files
/var	Contains variable data like logs, databases, website content etc

TODO - List the contents of the `/var` directory using the `ls` command learned in the previous milestone

Absolute path

In the Linux terminal, there is always more than one way to do the same thing.

Below are some options to display the contents of `/var`.



```
crio-user@ajay-criodo:~$ pwd
/home/crio-user
crio-user@ajay-criodo:~$ cd /var
crio-user@ajay-criodo:/var$ ls
backups  cache  crash  lib  local  lock  log  mail  opt  run  snap  spool  tmp
crio-user@ajay-criodo:/var$ pwd
/var
crio-user@ajay-criodo:/var$ cd
crio-user@ajay-criodo:~$ ls /var
backups  cache  crash  lib  local  lock  log  mail  opt  run  snap  spool  tmp
crio-user@ajay-criodo:~$ 
```

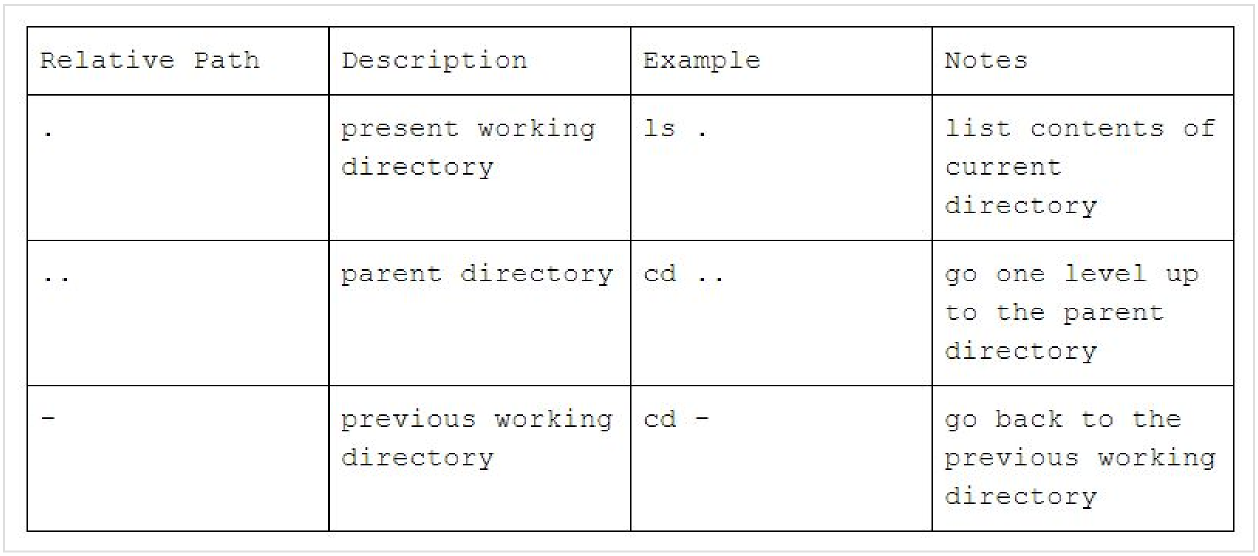
Using the absolute path to access directories

Irrespective of the present working directory, you can run `cd /var` to take you to `/var`. Similarly, `ls /var` can be executed from any working directory and it will always display the contents of `/var`.

This is called using the *absolute path* where we provided the full path to a directory right from the root of the filesystem: `/`.

Relative Path

Relative paths are *relative* to the present working directory. A list of special relative paths are listed in the table below and additional examples are in the code block that follow.



Relative Path	Description	Example	Notes
.	present working directory	ls .	list contents of current directory
..	parent directory	cd ..	go one level up to the parent directory
-	previous working directory	cd -	go back to the previous working directory

Additional examples

1. Using relative path to go back to the parent directory

```
crio-user@ajay-criodo:~$ pwd

/home/crio-user

crio-user@ajay-criodo:~$ ls .     ## list contents of the present working
dir

workspace

crio-user@ajay-criodo:~$ cd ./workspace  ## path is relative to the
present working dir

crio-user@ajay-criodo:~/workspace$ pwd

/home/crio-user/workspace

crio-user@ajay-criodo:~/workspace$ cd ..  ## one level up to the parent
dir

crio-user@ajay-criodo:~$ pwd

/home/crio-user
```

2. Easily go back to the previous working directory

```
crio-user@ajay-criodo:~$ cd workspace/ajay-criodo-ME_QMONEY/

crio-user@ajay-criodo:~/workspace/ajay-criodo-ME_QMONEY$ cd ../..  ## two
levels up

crio-user@ajay-criodo:~$ pwd

/home/crio-user

crio-user@ajay-criodo:~$ cd -  ## go back to the previous dir you came
from

/home/crio-user/workspace/ajay-criodo-ME_QMONEY

crio-user@ajay-criodo:~/workspace/ajay-criodo-ME_QMONEY$ cd -  ## and back
again

/home/crio-user
```

3. Using path relative to the home directory

```
crio-user@ajay-criodo:~$ cd ~/workspace  ## path relative to the home dir

crio-user@ajay-criodo:~/workspace$ pwd

/home/crio-user/workspace

crio-user@ajay-criodo:~/workspace$ cd
../workspace/../workspace/../workspace   ## fun :)

crio-user@ajay-criodo:~/workspace$
```

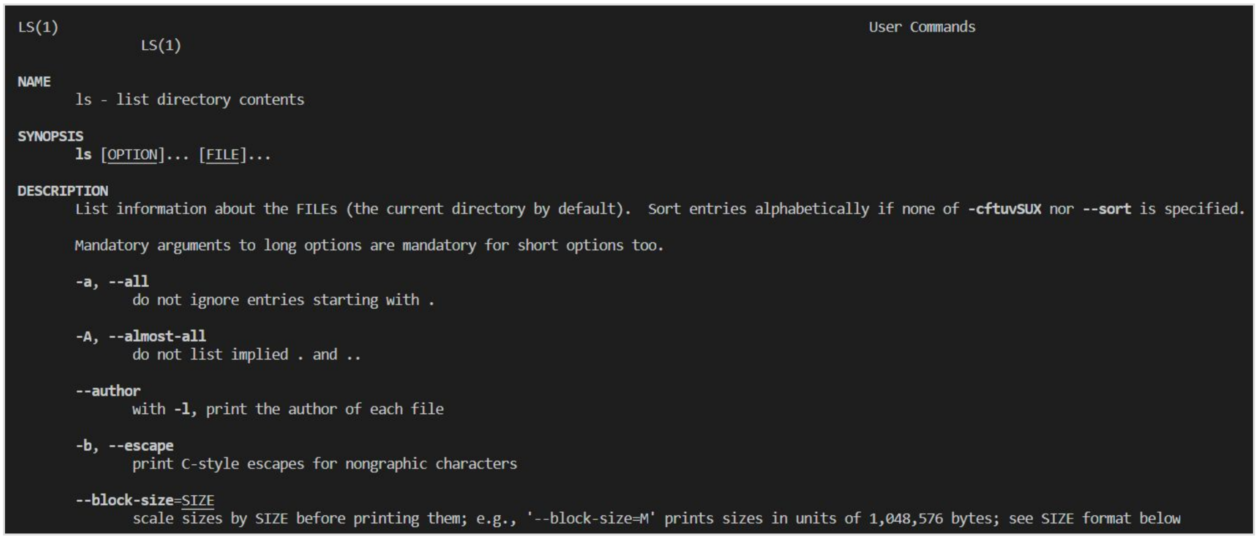
Try `ls ~/workspace` in the Linux terminal

Now, file explorer was giving us more info like the file size & modification time etc. How do we do that with `ls`?

Linux commands can be tuned to our requirements by providing flags along with the command when calling them. These are usually a hyphen (-) followed by an alphabet eg: **-a**, **-B** etc or double-hyphen (--) followed by text eg: --all, --color

But, how will we find a flag for our purpose?

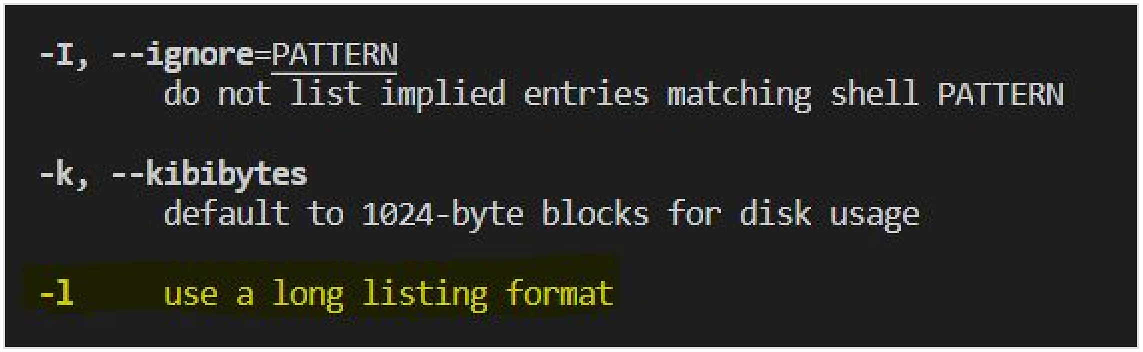
Commands come with a "Manual" as well. We can access it using the `man` command followed by the name of the command we need to see the manual of. For `ls`, we do `man ls`.



```
LS(1)                                                                     User Commands
          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory by default).  Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print C-style escapes for nongraphic characters

       --block-size=SIZE
              scale sizes by SIZE before printing them; e.g., '--block-size=M' prints sizes in units of 1,048,576 bytes; see SIZE format below
```

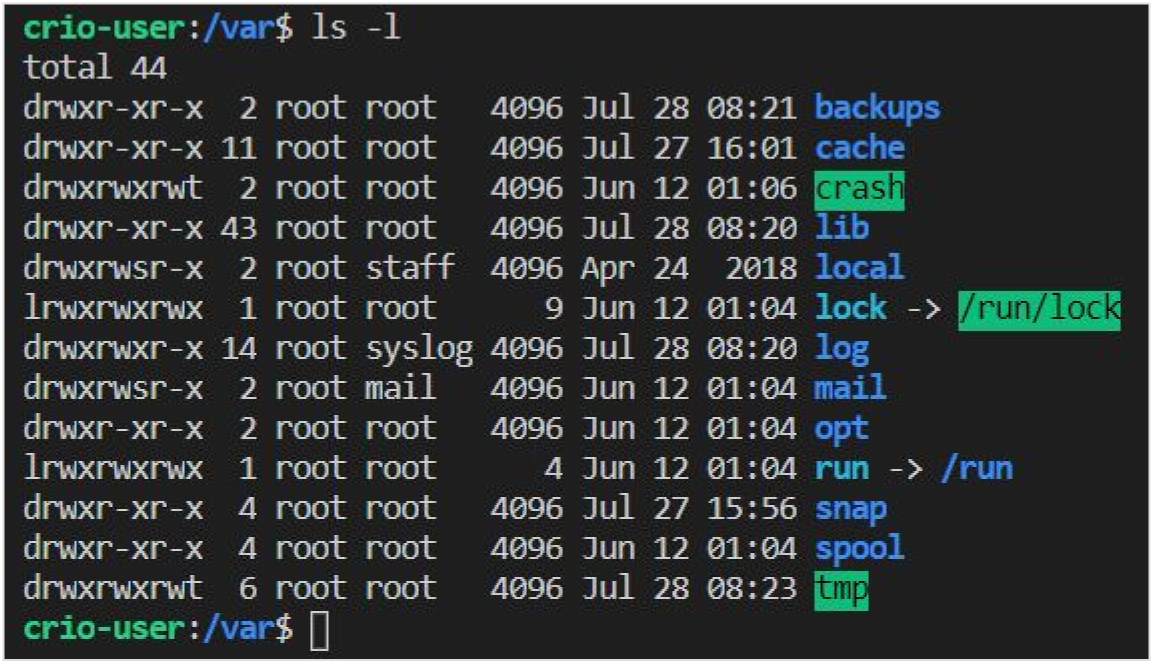
- **NAME** - name of the command & short description of what it does
- **SYNPOSIS** - how the command is used
- **DESCRIPTION** - detailed info on the usage of the command

We can see flags supported by the `ls` command listed out. You'll be able to scroll through the manual. See if you can find the **-l** flag.



```
-I, --ignore=PATTERN
       do not list implied entries matching shell PATTERN

-k, --kibibytes
       default to 1024-byte blocks for disk usage

-l     use a long listing format
```

Ok, seems like something related to our need for finding more info about the directory content. Let's try it. (Stuck inside the manual page? - Hit **q** to exit)


```
crio-user:/var$ ls -l
total 44
drwxr-xr-x  2 root root   4096 Jul 28 08:21 backups
drwxr-xr-x 11 root root   4096 Jul 27 16:01 cache
drwxrwxrwt  2 root root   4096 Jun 12 01:06 crash
drwxr-xr-x 43 root root   4096 Jul 28 08:20 lib
drwxrwsr-x  2 root staff  4096 Apr 24  2018 local
lrwxrwxrwx  1 root root      9 Jun 12 01:04 lock -> /run/lock
drwxrwxr-x 14 root syslog 4096 Jul 28 08:20 log
drwxrwsr-x  2 root mail   4096 Jun 12 01:04 mail
drwxr-xr-x  2 root root   4096 Jun 12 01:04 opt
lrwxrwxrwx  1 root root      4 Jun 12 01:04 run -> /run
drwxr-xr-x  4 root root   4096 Jul 27 15:56 snap
drwxr-xr-x  4 root root   4096 Jun 12 01:04 spool
drwxrwxrwt  6 root root   4096 Jul 28 08:23 tmp
crio-user:/var$ 
```

Gotcha!

The size of the file/directory in bytes is in the 5th column & 6th column is the last modified date & time. There’s a file **lock -> /run/lock**. Weird name, right? It’s a symbolic link i.e, **/var/lock** is actually pointing to the file **/run/lock**

Manipulating file permissions

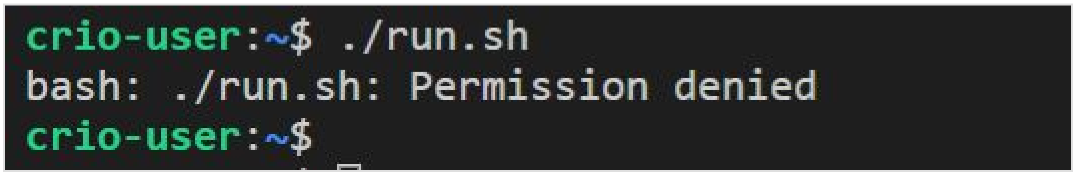
As we learnt earlier, **/home** is where user files are stored. Each user has a directory inside it. My username is **crio-user** and thus the directory is **/home/crio-user**. Find yours & `cd` into it.

```
cd /home/crio-user
```

Executables are programs that can be run to perform some instructions. Let's create one & try to run it. Execute below command to create a file called **run.sh** with some content. Don't get intimidated by the command, you'll understand what this is doing by the end of this Byte :)

```
echo "echo 'Congratulations on running a bash script'" > run.sh
```

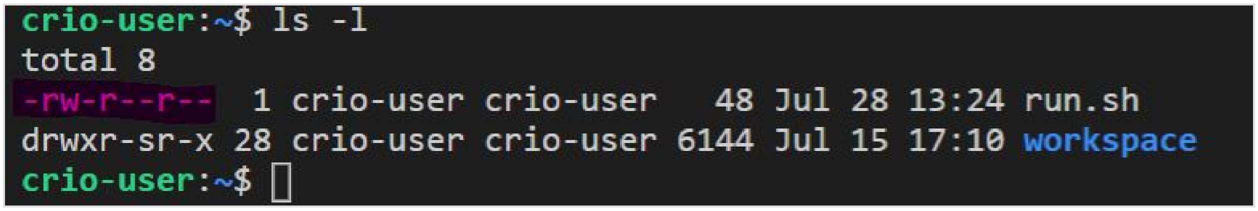
Now, how do we run an executable? We do `./<name-of-executable>`. Ok, that'd be `./run.sh` for us.



```
crio-user:~$ ./run.sh
bash: ./run.sh: Permission denied
crio-user:~$
```

Hmm, that's bad :(

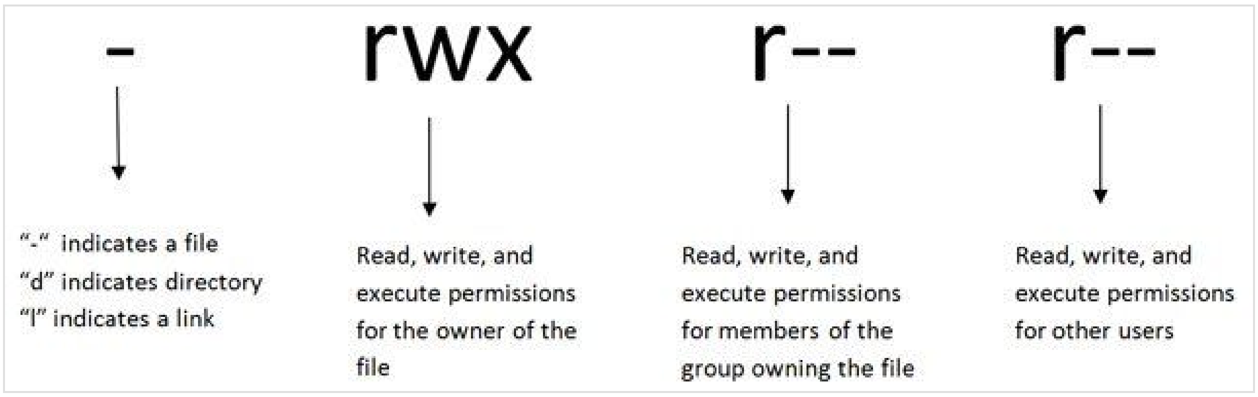
At least we aren't completely lost, there's some info for us to debug. It has something to do with permissions. Remember when we used `ls` to print out data in the long format earlier, it was also showing the file permissions. Let's revisit that



```
crio-user:~$ ls -l
total 8
-rw-r--r--  1 crio-user crio-user   48 Jul 28 13:24 run.sh
drwxr-sr-x 28 crio-user crio-user 6144 Jul 15 17:10 workspace
crio-user:~$ 
```

The highlighted text is the file permission for the **run.sh** file. Seems gibberish, right?

Let's decode it.


-
rwx
r--
r--
"-" indicates a file
"d" indicates directory
"l" indicates a link
Read, write, and execute permissions for the owner of the file
Read, write, and execute permissions for members of the group owning the file
Read, write, and execute permissions for other users

Linux file permissions (Source: https://www.pluralsight.com/blog)

As the image tells us, **r** means permission to read data from a file, **w** means permission to write/edit a file & **x** means permission to execute a file.

Aha! If we see the permissions for our **run.sh** file, it doesn't have any **x** in it & thus was throwing an error when we tried to run it. Just have to give e**x**ecutable permission & we'll be good.

`chmod` command lets us **ch**ange the access **mod**e of a file. To add executable permission, we use `chmod +x <filename>`

```
chmod +x run.sh
```

What do the file permissions now look like?

See if you can run the file now!

References

1. Intro to Linux file systems
2. ls command usage
3. Understanding ls command output
4. Creating symbolic links
5. Absolute and relative paths
6. Linux file permissions

Curious Cats

- Find a command that prints out all sub-directories and files recursively

- Try to remove the write permission for the `run.sh` file. Will you be able to edit the file after this?
- `chmod` changes the access mode for a file. Can you find out how we can change the owner of a file?
- `ls -l` lists the contents in long form whereas `ls -t` sorts the files according to their modification time. What if we need to do both?

MILESTONE #3

Navigating via the terminal

We're inside our home directory (which is **/home/crio-user** for me). Ok, how will you go to the parent directory ie, **/home**?

We need to do `cd /home`, right?

Okay, what if we were inside **/home/crio-user/Downloads/videos/series/english/got/season100/episode1** and we need to go back to the **season100** folder? It'll be quite tedious to type it out. Instead we can use `cd ..` for that. Double periods (..) is Linux lingo stands for parent directory. Try it out!

Cool! Are there any other shortcuts like these? Glad you asked :)

As you would've felt by now, the home directory of a user is an important directory and hence gets a shortcut. `cd ~` will get you to the home directory of the current user no matter where you are inside the Linux filesystem.

Remember how we ran the **run.sh** file earlier? We did `./run.sh`, right?

Similar to ".." denoting the parent directory, single period (**.**) denotes the current directory. Essentially, the earlier command was telling the terminal "Run this file present in the current directory"

CRUD - Create, Read, Update, Delete

We've been going back and forth between directories & listing out already available files. Let's take it up a notch & create files of our own!

Come back to your home directory if you were lost trying out variants of `cd` from the previous section. Verify using `pwd` as always.

I watched **K.G.F: Chapter 1** last night and I'm planning to store all movies I watch and some info about those, like synopsis, review etc. To start with, let's create a directory structure like this in the home directory

```
/home/crio-user
        |- kgf
        |    |- cast.txt
        |    |- part1
                |\- synopsis.txt
        |    |- part2
\- synopsis.txt
```

Hmm, first we'll create the directories required & then the files. `mkdir` is the command we need to **ma**ke **dir**ectories in Linux and we feed it the name of the new directory we want to create. So, we'd use `mkdir kgf` to create a directory called **kgf**. As we were in the home directory, the absolute path of the folder will be **/home/crio-user/kgf** (we could also do `mkdir /home/crio-user/kgf`). We'll create the subdirectories using the `mkdir` command as well. Instead of using `cd` to go to the **kgf** directory and then using `mkdir part1` we can use relative path to create the new directory from the home directory itself i.e., both the below sets of commands do the same thing.

```
cd kgf
mkdir part1
mkdir part2
```

Or

```
mkdir kgf/part1
mkdir kgf/part2
```

Cool. Now, we need to add files. That'd be the `touch` command. Come back to the **kgf** directory using `cd` if you are somewhere else. Execute `touch part1/synopsis.txt` to create a file named **synopsis.txt** inside **part1** directory.

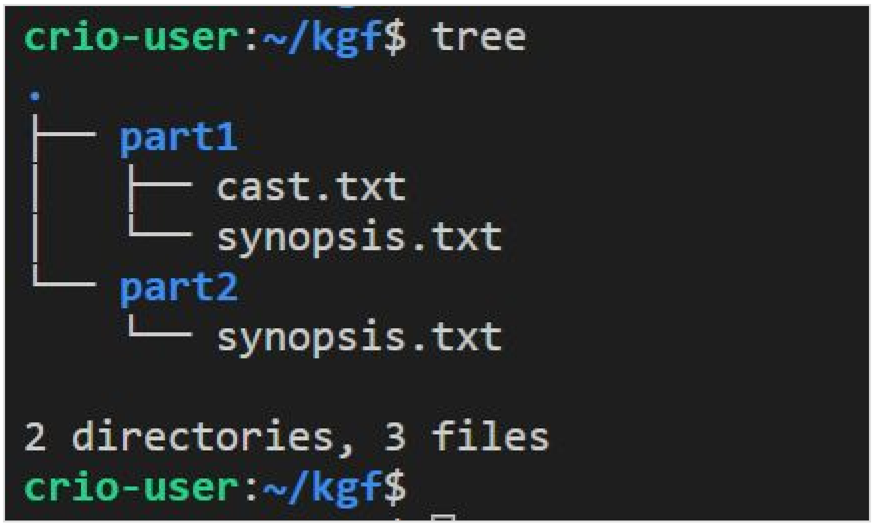
Similarly, create a **synopsis.txt** file inside **part2** directory as well.

I'll add the synopsis for part1 as it's already out. `nano` command lets us edit files and is one of the default editors on Ubuntu. There would be other editors as well. Using `nano part1/synopsis.txt` will open the file for us to edit. After adding the required text

1. Hit `Ctrl + x`
2. Enter **y** when asked if to save
3. Hit Enter as we don't need to change the file name we're writing to

To verify the data was indeed saved, we can use the `cat` command to print out contents of a file. Execute `cat part1/synopsis.txt`

We have one more file to create - **cast.txt**. Execute `touch part1/cast.txt` to create it. With that, we have created all the directories and files as planned. It's always good to verify when we're done with something. `cd` to the **kgf** directory & execute `tree` command to print out a layout of files & directories under it.



```
crio-user:~/kgf$ tree
.
├── part1
│   ├── cast.txt
│   └── synopsis.txt
└── part2
    └── synopsis.txt

2 directories, 3 files
crio-user:~/kgf$
```

Oops! The **cast.txt** file should've been inside the **kgf** directory and not inside **kgf/part1**.

We can copy the file to the **kgf** directory and then remove **part1/cast.txt**. From the **kgf** directory, use

1. `cp part1/cast.txt .` - to **co**py the **part1/cast.txt** file to directory denoted by "**.**" which we know from our earlier discussion is the current directory
2. `rm part1/cast.txt` - to **rem**ove the **part1/cast.txt** file

Use the `tree` command now to verify the structure again

References

1. Everything is a file in Linux

Curious Cats

- To create the dir1/subdir1 directory, we can use `mkdir dir1`, `cd` to **dir1** and then execute `mkdir dir1/subdir1`. Can you find out if it's possible to create the **dir1/subdir1** directories using a single `mkdir` call?
- Does Linux provide some command to just move files from one directory to another rather than having us to do a copy first & then remove the original?
- On second thoughts, we decide to remove the **part2** directory. How would you delete a directory?

MILESTONE #3

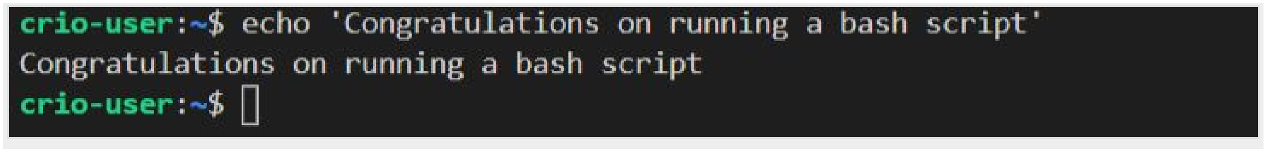
Writing to files via output redirection

In milestone 2, we used the below command to create a file called **run.sh** & then looked at how to run it.

```
echo "echo 'Congratulations on running a bash script'" > run.sh
```

But, what was it actually doing?

`echo` command is used to print out the value provided to it. By default, it prints the value to the terminal which is the "standard output". Try running `echo 'Congratulations on running a bash script'`



```
crio-user:~$ echo 'Congratulations on running a bash script'
Congratulations on running a bash script
crio-user:~$ 
```

Now, we want this command to be run when the **run.sh** file is executed (or run). For this, we need to add the whole command `echo 'Congratulations on running a bash script'` itself to the file. As we didn't know about using the `nano` editor earlier (or `echo` for that matter :)), we used the `echo` command itself to write the whole command to the file.

`echo "echo 'Congratulations on running a bash script'"` prints **echo 'Congratulations on running a bash script'** to the terminal. We can redirect the output instead to a file using the redirection operator **>**. You can verify this indeed was happening earlier by `cat`ing the content of the `run.sh` file from earlier to the terminal.

Feeding Output of a command to the Input of another

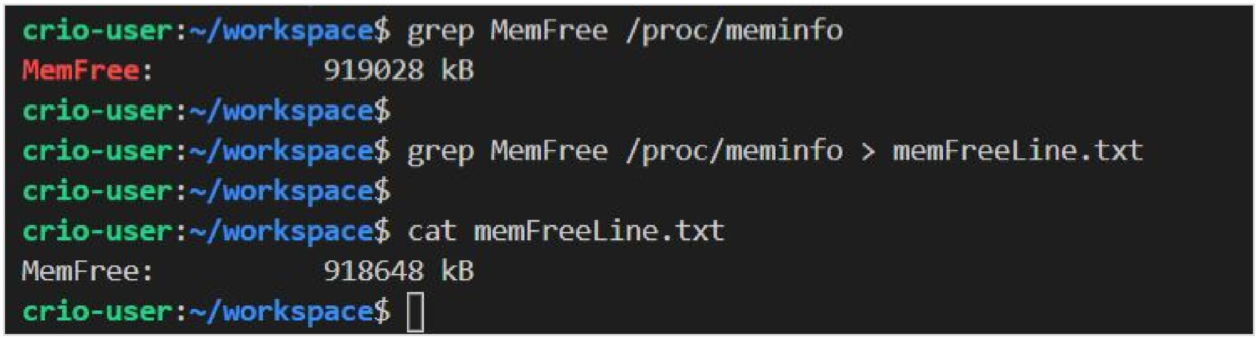
The **/proc/meminfo** file has info related to the RAM installed in our system. Try using `cat` to print its content. The first three lines are the attributes **MemTotal**, **MemFree** & **MemAvailable**. What if we needed to print out just the value of the **MemFree** attribute which denotes the amount of free memory?

We'll be required to perform a couple of actions.

1. Filter for the line in **/proc/meminfo** containing the attribute **MemFree**
2. Fetch the numerical value in that line by separating that column

`grep` command is used to filter/search for text using strings or patterns. It's usage is `grep <pattern> <file>`. For us, it'd be `grep MemFree /proc/meminfo`. You'll see only the line containing **MemFree** from **/proc/meminfo** filtered out. Let's write it to a file using the redirection operator (**>**) we learnt in the previous section.

(The MemFree value is dynamic & hence can be different every time we print it out)



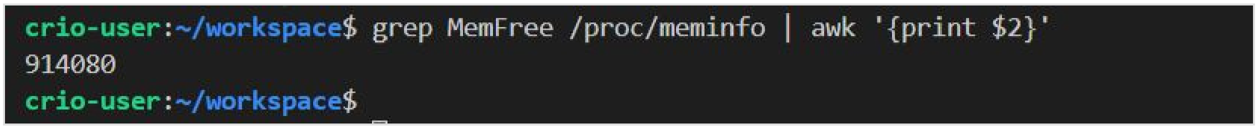
```
crio-user:~/workspace$ grep MemFree /proc/meminfo
MemFree:          919028 kB
crio-user:~/workspace$
crio-user:~/workspace$ grep MemFree /proc/meminfo > memFreeLine.txt
crio-user:~/workspace$
crio-user:~/workspace$ cat memFreeLine.txt
MemFree:          918648 kB
crio-user:~/workspace$ 
```

Now, how will we fetch only the numerical value?

`awk` can do that for us. It uses space as a field separator (by default space is used as field separator but other field separators can be specified) and separates text into columns. That'd mean, we have 3 words in our line - **MemFree:**, **918648** & **kB**. To print out the 2nd word, the usage is `awk '{print $2}' memFreeLine.txt` where `$2` denotes the second word in the line.

That did the work for us and we have just the free memory in kB printed out to the terminal. Looking back, the **memFreeLine.txt** file was a by-product which we didn't actually need. Is there some way to do this without having to create this file?

Piping is a type of redirection in Linux used to send output of one command to input of another command. We had earlier written the output of `grep` to a file & then used it as input to `awk`. Instead of this, we can use the Linux pipe operator (**|**) to redirect output of `grep` directly to input of `awk`.


```
crio-user:~/workspace$ grep MemFree /proc/meminfo | awk '{print $2}'
914080
crio-user:~/workspace$
```

As you can see, `awk` command isn't given a file to read from but rather get its input from out of `grep` because of the use of the pipe operator (**|**).

You can continue linking together more commands like this, using multiple pipes, to achieve any of your goals!

Like we saw here, there can be multiple ways we can do a task in Linux

References

1. Piping and Redirection

Curious Cats

- We came across a couple of ways to redirect the output of a command. **>** operator writes output of a command to a file. Try running `echo "Hello" > hello.txt` command two times and `cat` content of **hello.txt**. Does it contain one **Hello** or two? How can we append data rather than overwrite it here when using redirection?
- The | operator sends output of a command as input to another. What if we needed to print the output of the first command to the terminal as well as redirect it to a file?

- What if I need to see the first 20 lines of a file? Is there some command that lets us fetch some lines from the starting of the file?
- What if you want to see contents of a file getting printed to screen as and when the file is being written to?

MILESTONE #4

Data analysis from the command line

With the Linux terminal power behind us, we'll apply this to gather statistics/data from standard software logs.

Hadoop log Analysis

- Hadoop (https://hadoop.apache.org) is a big data processing framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
- We have a Hadoop log bundle here
- And this is the data we want to pull up from the logs
    - How many log files do we have?
    - Number of "ERROR"s seen across all the logs
        - Similarly, how many "FATAL", "WARN" and "INFO" messages do we see
    - How many "Exceptions" do we see?
        - Which of these exceptions are related to the disk being full?

        - Print out a list of log filenames and the count of how many exceptions are seen in each file, if the count is > 1
        - Print a Histogram of these counts per file
    - Filter out across the log files, all events that occured at a particular minute
        - E.g. "2015-10-19 14:25"
        - List all log file names where we can see log messages for this minute
    - How many "MapTask metrics system" and “ReduceTask metrics system” shutdowns were completed?

Let’s look at how this can be achieved easily with Linux commands.

Download the script and execute it using these commands

```
crio-user:~/workspace$ wget
https://gitlab.crio.do/crio_bytes/me_linux1/-/raw/master/hadoop_log_analys
is.sh -O hadoop_log_analysis.sh
```

```
crio-user:~/workspace$ chmod +x hadoop_log_analysis.sh
```

```
crio-user:~/workspace$ ./hadoop_log_analysis.sh /tmp/log_analysis
```

The script prints out all the data we need, as a report.

Take a detailed look at the commands in `hadoop_log_analysis.sh` and you'll see how each task is achieved using a combination of simple commands.
Note: You can achieve the same with a Python script (or other alternatives) from the command line as well.

That's about Hadoop Analysis. Are you ready to analyse some more log files on your own?

Now you are given some log files and no script this time, Go through the log files and try to come up with a script similar to the one provided for Hadoop Analysis.

OpenSSH log Analysis

- OpenSSH is the premier connectivity tool for remote login with the SSH protocol.
- We have an OpenSSH log here
- And this is the data we want to pull up from the logs:
    - How many break in attempts do we see? ("POSSIBLE BREAK-IN ATTEMPT!")
        - Reference: Brute Force and Hail Mary attacks
    - How many invalid certificate attempts do we see?
        - Are there particular IP addresses from which these invalid requests are coming in?
    - What are the ports that the OpenSSH server is listening on? (E.g. "Server listening on 0.0.0.0 port 902")

    - Plot the number of Error messages across time. This will show the time periods where errors were higher.
    - How many accepted connections do we see? (E.g. "Accepted password for curi from 137.189.90.232 port 20631 ssh2")
- How would you go about achieving this from the Linux Terminal?

Apache log Analysis

- Apache HTTP Server (https://httpd.apache.org) is one of the most popular web servers. Apache servers usually generate two types of logs: access logs and error logs.
- We have an Apache error log here
- Can you come up with interesting statistics for this log file and do the analysis?

References

- https://github.com/logpai/loghub
- https://zenodo.org/record/3227177#.XzQXYXUzbq8
- https://en.wikibooks.org/wiki/OpenSSH/Logging_and_Troubleshooting
- https://www.cyberciti.biz/faq/bash-loop-over-file/

Task Automation

Good developers don't like to do the same tasks repeatedly. Scripts and/or automation is their answer to these repetitive tasks.

Let's use the Hadoop log analysis script from the previous section and automate its execution.

If we need to run this script every 5 mins, given that new logs are being generated, let's see how to achieve this. (In the real world, you may want to run it less frequently. For demonstration purposes, let's run it every 5 mins).

Note: This script downloads the files from a static web link. On a live server, this script would fetch files from a directory on the Hadoop server(s) or run on the server itself.

We will use an in-built utility on Linux called `cron` to schedule execution of this script.

The **Cron runs processes on your system at a scheduled time**. Cron reads the **crontab** (cron tables) for predefined commands and scripts that need to be run.

By using a **specific syntax**, you can configure a cron job which schedules scripts or other commands to **run automatically**.

Run the below commands to set the cron job

```
# Open the cron tables with this command

crio-user:~/workspace$ crontab -e
```

```
# This will open the crontab file in an editor. Choose nano if prompted
for choice of editor.

# Go the last line of this file and add the below line

5 * * * * /home/crio-user/workspace/hadoop_log_analysis.sh
/tmp/log_analysis > /home/crio-user/workspace/log_report

# Substitute the above paths to suit your setup

# Save file and exit  ("Ctrl X", followed by “Y”, followed by “Enter” to
exit nano editor)
```

Reference links have been provided below to understand the crontab structure in detail.

This is what we’ve currently done

- Specified that the the job should be run automatically, every `5` minutes
- The job to be run is the `/home/crio-user/workspace/hadoop_log_analysis.sh` script with `/tmp/log_analysis` as a command line parameter (this directory will be used to save log files).
- The output of the job (which is the log analysis report) gets stored in the file `/home/crio-user/workspace/log_report`. Modify this path accordingly if you are not in the Crio Workspace.

Verify that the `/home/crio-user/workspace/log_report` is being created every 5 minutes.

Automation was easy!

The script can be enhanced to send a text message or an email when a particular value in the report is not as expected. For example, when too many FATAL messages are seen.

Other example tasks you could accomplish using cron jobs

- Backup files or database periodically
- Cleanup disk space when it fills up beyond a certain limit
- Monitor processes and restart them if they go down

References

- https://www.whoishostingthis.com/resources/cron/
- https://phoenixnap.com/kb/set-up-cron-job-linux
- https://www.thegeekstuff.com/2009/06/15-practical-crontab-examples/

Useful shortcuts

- `up arrow` key → will bring up the last command that was executed. Each press will bring up the previous command executed.
- `history` → will print out a list of the previous commands executed.
- `tab` key → pressing the tab key can be used to auto-complete the directory and file names while typing paths or filenames.

References

- Find pointers to Curious Cats questions here
- Handy Linux cheat sheet - Cheat Sheet 1 and Cheat Sheet 2
- Examples for Linux usage - Examples
- Detailed Linux command list - Linux Programming Commands

Why Linux?

A majority of systems around the world run some form of Linux. These range from enterprise and desktop servers to smartphones.

- 60 to 70% of all Web Servers in the world run some form of Linux/Unix and ~90% of all cloud computing happens on Linux based servers.
- Most of the smartphones in the world run on Linux.
- Mac is based on Unix and supports a terminal where these commands can be run.

In the Real World

- What does Linux power - 25 Awesome unexpected things powered by Linux
- What you could use Linux additionally for - Alternative examples of Linux usage

Newfound Superpowers

- Knowledge of an all powerful Linux terminal
- Most commonly used Linux commands in your arsenal

Now you can

- Explain where the terminal fits in & how it acts as an alternative to the GUI
- Run commands in the terminal & modify their behaviour using flags
- Navigate easily across directories and create/read/update/delete files & directories
- Filter data stored in files and redirect output of commands to either a file or as input to another command