The Unix Commandline

Jeetaditya Chatterjee

May 27, 2021

What is the Unix commandline?

The unix commandline is an interpreter known as the shell and a set of tools that come together to form a functional operating system. It is a central part of unix and for a very long time the only way to interact with unix. Not only do you call a command but you can chain them to form larger and more complex programs from simple building blocks. This is called pipelining and will be discussed later

Why do I need to know the Unix command line?

It is a good skill to have for many reasons

- Linux is the biggest and largest form of Unix like OS used today and its used on most servers. If you ever need to ssh into a server to fix something and you don't know the basics then you will be F*ucked
- The commandline unlike in windows is much more direct and powerful. This means that a lot of guides in linux land will use it over a gui tool you may be used to
  - chaining commands together feels a lot more satisfying than *python*

What do I need too know then?

A little bit of syntax

Variables

Shell is not a very nice language so syntax is important To set a variable you would set it like any other language but you can't have a space between the variable symbol and the equals sign
To call it you need to prepend the $ sign
Environment variables are global variables to set them you need to prepend export
You can also asign the output of commands to variables

```
var="some value"
export ENVAR="some global value"

command_out=$(some command)
echo $var
echo $ENVAR
echo $command_out
```

flags

A flag is an optional argument provided to a command to provide it with some sort of structured input or to tell it to do something they are usually denoted with a - hyphen and a single letter or 2 hyphens and a word you can usually string together multiple short form commands

```
command -l # short form flags
command -la # 2 short form flags
command --long # long form flags
command --input INPUT # example of input
```

The main commands

These are all of the commands that are kind of essential. I don't expect you to remember them all and this is not an exaustive list. This should give you a starting place. That being said there are quite a few

Moving around

pwd

Prints the current directory pwd prints the current working directory It stands for print working directory

```
$ pwd
```

cd

change directory Change the current directory

Some sybols that are helpful ~ which is a shortcut to your home dir . is the current directory .. is the previous directory

```
$ cd ~/your/directory
$ pwd
/home/$USER/your/directory
$ cd ~ # cd /home/$USER
$ pwd
/home/jeet
$ cd ..
$ pwd
/home
```

ls

list the current directory's contents

```
$ ls
$ ls -a
$ ls -l
$ ls -la
```

The ls command lists the current directory the -a flag will show hidden files (all files) these are denoted with a dot infront of there name the -l flag will list all files in the long format

Files

touch

touch is used to create an empty file. Its actually used to change the time stamp on the file but thats not used as much. if at all

you have to provide the date in the disgusting order of month day year

```
$ touch file
```

```
$ touch existing-file -d month/day/year
```

mv

mv is used to move files around the filesystem. It takes the file to move and the destination of that file. It is also used to rename a file

If you need to move directorys then you need to make sure that the names are unique. mv will not move that file by default but if you provide the -f then you will overwrite the previous directory There also the -i flag that will ask you to confirm

```
$ mv SOURCE DESTINATION

$ ls
file1
$ mv file1 file2
$ ls
file2

$ ls
dir1 dir2
$ mv dir1 dir2
mv: cannot move 'dir1' ro 'dir2' : Directory not empy
$ mv dir1 dir2 -f
$ ls
dir2
```

cp

cp copys a file leaving the source file intact It can overwite the destination file tho so be careful with your names you can use the -f and -i flag like you would for mv

```
$ cp SOURCE DEST

$ ls
file1
$ cp file1 file2
$ ls
file1 file2
```

ln

ln creates a symbolic link between two files. a symbolic link is reference to another file it does not actually copy the file and is very useful for a lot of different senarios. some being storing files spread out over the file system in a central location.

it take the arguments of source and destination and an optional flag -s which makes the link soft. by default the symbolic link will not look like a link but another file. if you want it to show up as a link you need to add the -s flag

```
$ ln source-file dest-file
$ ls -l
-rw-rw-r-- jeet jeet ... source-file
-rw-rw-r-- jeet jeet ... dest-file

$ ln source-file dest-file -s
$ ls -l
-rw-rw-r-- jeet jeet ... source-file
lrwxrwxrwx jeet jeet ... dest-file -> source-file
```

rm

delete a file. add the -i flag to ask for your confirmation. If you need to delete a directory then you need to add the -r (recursive) flag and if the file cannot be written to then you can use the -f option

```
$ rm file -i
rm: remove regular file 'file'? # your input here
# no more file
$ rm dir
rm: cannot remove 'dir' : Is a directory
$ rm dir -r
# all good
```

mkdir

make a directory.. use the -p flag to make a lot of child directorys quickly

```
$ mkdir dir

$ mkdir dir1/dir2/dir2
mkdir: cannot create directory 'dir1/dir2/dir3': No such file or directory
$ mkdir dir1/dir2/dir2 -p
```

Reading and writing files

cat

output and concatenate multiple files.

```
$ cat file
# contense of file
$ cat file1 file2
# contense of both files
```

head

Output the first 10 lines of a file the -n flag allows you to specify the amount of lines you want to print out

```
$ head file -n 20
```

tail

Same as head but for the last 10 lines again -n will allow you to specify the amount of lines

```
$ tail file -n 20
```

chmod

Change the file permissions. In linux files do not need extensions. instead attached to each and every file is a set of permissions. You saw them when we discussed ln. chmod changes those permissions. if you want to add a permission then do + followed by r w x or remove them replace the + with a minus

Notice the different groups. the first symbol is the type it can be d for directory l for link or - for file the first set of 3 is what root can do. root is the main user and they have free reign over the system. the second triple is the user permissions and the last one is other peoples permissions

```
$ ls -l
-rw-rw-r-- ... file
$ chmod +x file
$ ls -l
-rwxrwxr-x ... file

$ chmod -rwx file
---------- ... file
```

Transforming text

echo

shell print statement. It will print out a string of text. If you use

```
echo "a string"
a string
echo -e "this is a \n multiline string "
this is a
 multiline string
echo $VAR
# contense of VAR
```

grep

grep is a commandline regex matcher. It takes the arguments of a string (usually with a regular expression) and a file. some useful flags are -v which returns all of the lines that do not match -i which ignores case and -c which counts the number of occurrences

```
$ cat file
fox
fax
fex
$ grep "fox" file
fox
$ grep "f.x" file

$ grep "fox" file -v
fax
fex
```

sed

sed is used to make substutions in text it takes the arguments of a command and a file It has a little bit of a werid syntax so bear with me

in the string we start it with s this tells sed we are performing a substitution we then provide it with a regex this is what will be substituted then you provide the substitution. There is more syntax here but these are the basics

Now by default sed will only perform 1 substitution to tell it to replace all you need to add g to the end this is called a flag

You can add the -i flag to change the file in place. if you add an argument to it will create a backup of the file you are about to change

```
$ sed 's/REGEX/REPLACEMENT/FLAG' file

$ cat file
the fox did fox things to say fox you to big fox
$ sed 's/fox/wolf/' file
the wolf did fox things to say fox you to big fox

$ sed 's/fox/wolf/g' file
the wolf did wolf things to say wolf you to big wolf

$ sed 's/fox/wolf/g' file -i

$ cat file
the wolf did wolf things to say wolf you to big wolf
```

awk

awk is a swiss a text processing swiss army knife. it is its own language and can pretty much replace all of the text processing commands. Now do I recommend that? welll probably not. but its pretty cool. Now that being said most people use awk to get out column's of data.

awk programs are constructed from a set of pattern and actions.

Awk first splits your files into lines and then into fields and then store those fields in variables. they are denoted by a $ sign and the number of the field note that $0 holds the whole file and $NF holds the last. If you want the first field you would call $1 second so on

Now this can be useful but it may be more useful for a csv file. To change the field seperatior you need to use the -F flag

In this example we are restructuring a csv to make it into a nicer format

```
some-pattern { some action }
another-pattern { a different action }
{ all lines match this action }
```

```
$0 ~ /some regex/ {
    print $1, $2 # for example
}

$1 == 18 {
    print $2, $1
}
```

```
$ cat file
this file is has space seperated words
it has multiple lines
$ awk '{print $0}' file # we just made a slower less convenient cat!
this file is has space seperated words
$ awk '{print $1}' file # print the first element of each line
this
it
$ cat file.csv
name,age,job
Jeet,18,Being a Nerd

$ awk -F, '{printf("name:%s job:%s age:%s", $1, $3, $2)}' file.csv
name:Jeet job:Being a nerd age:18
```

sort

sort sorts a file. You can tell it to sort using a numeric sort -n a dictionary sort (whch is also known as alphanumeric) as well as others

```
$ cat file-o-numbers
1
15
2
30
$ sort -n file-o-numbers
1
2
15
30
```

uniq

uniq removes dupicates from a file. Nothing really more to say

```
$ cat file
this
this
has
some
some
duplicates

$ uniq file
this
has
some
duplicates
```

wc

wc stands for word count. it will count the amount of lines words and bytes in that order you can add -l flag to get the line count -w to get the word count or -c for the byte count

```
$ cat file
line 1
line 2
line 3

$ wc file
3 6 24
$ wc -l
3
$ wc -w
6
$ wc -c
24
```

User management

sudo

sudo stands for super user do. it allows you to execute commands as another user (usually root). This is used all over the place as root is the only one allowed to do certain things like installing system programs and changing files outside of your home directory. sudo allows you to execute one command like that as a sort of security mesure.

```
$ apt update
Permission denied
$ sudo apt update
# works
```

w

w shows you who is logged on and what they are doing it will give you information on who is logged on where there are logged in from and what they are doing.

```
$ w

 02:26:18 up  8:52,  2 users,  load average: 0.94, 1.29, 1.15
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
jeet     :1       :1               17:34   ?xdm?   4:07m  0.01s /usr/libexec/gdm-x-session --run-s
root     tty4     -                02:24    1:52   0.02s  0.01s -bash
```

Process management

pgrep

pgrep will give you the process id's that match the a string they give you

```
$ pgrep emacs
23878
23878

$ pgrep firefox
6997
```

pkill

pkill kills the process or processs given to it. You can give it a process id or the string you were going to give to pgrep anyway. use the -9 flag to kill the process instantly

```
$ pkill emacs # waits for emacs to finish

$ pkill 23878 -9 # kills emacs instantly
```

htop

htop is probably not installed by default but it makes all of this manual work interactive. Its better if I show you

```
$ htop
```

Others

man

man is the inbuilt unix documentation. I did not need the internet to find out about all of the commands I just bored you with. I just typed in man command and it gave me everything the command can do in a semi nice to read format

man uses a pager by default so you can scroll and have a look at your leasure

```
$ man command
```

Honorable mentions

nc The networking swiss army knife

calc like awk for numbers

lp Have you ever needed too print something? well now you can on the command line!

less Read files at your pleasure

Pipes

Output redirection

for the most part we have just been outputing all of our data onto the proverbial floor and we should really be putting it into a file. we can use output redirection to put the stdout into a file

Nothing will be outputed onto the screen and all of the output will be put into the file. This is great. but now we have over written the file.. maybe we just want to append to the file.

There are also some special places you can redirect to and some special outputs you can get.

/dev/null is a black hole. when you send data there it will never be saved this is useful when you have a program that sends out a lot of error output you don't need. This is also a good time to mention that errors are sent out on a different stream of text. called stderr You can redirect the errors from it by simply adding a 2 in front

```
$ cat file1 file2 > file1+file2

$ cat non-existant-file 2> /dev/null # no errors and no output
```

Output appendation?

To append we use double the ammount of more than signs. This is usefull for when you have a log file. Just be careful to make sure the amount of more than signs are correct

```
$ some-output >> some-log-file
```

Pipes

Pipes are a very powerful concept in unix. It allows you to build bigger programs from smaller ones. The way it works is you chain the std output of one command into the std input of the next. the command will do its work and pass it onto the next. Whenerver you use a print statement in python you are outputing to std output and its what these programs are doing. Lets take an example. I have a csv full of names and ages. I want to get the age of all the people who are over the age of 18 and have a name beginning with J

Now this example is not all to revolutinary but you can create very complex programms from simpler blocks. This way of thinking can help you frame code problems in a different way. As a

set of transformations of an input instead of a set of instructions. This is actually a tennant of functional programming

```
$ cat file
Jeet,18
Jen,19
Mark,17

$ cat file | grep -i "j." | awk -F, '$2 >= 18 {print "name:", $1, "age:", $2}'
name: Jeet age: 18
name: Jen age: 19
```

An actual example

This is an actual script I wrote to solve my needs. **show gh gist list** I use the github cli to handle some thinsga and github gists are just one of them. For the purposes of today they are just small code files with a hash attactched to them and to edit them you need to paste in the hash. I could do that manually but thats a pain so I automated

walk through the script

```
gist_list=$(gh gist list)
gist_to_find=$(echo "$gist_list" | awk '{print $2}' | fzf --layout=reverse)

gh gist edit $(echo "$gist_list" | grep $gist_to_find | awk '{print $1}')
```

What to do from here?

Well now you have this introduction you need to use it. You can practice with the linux environments I hope you now have. Have a play with it and have fun with it. Wrote memorisation like this will probably not help but apply these skills will help you a lot

Any Questions?