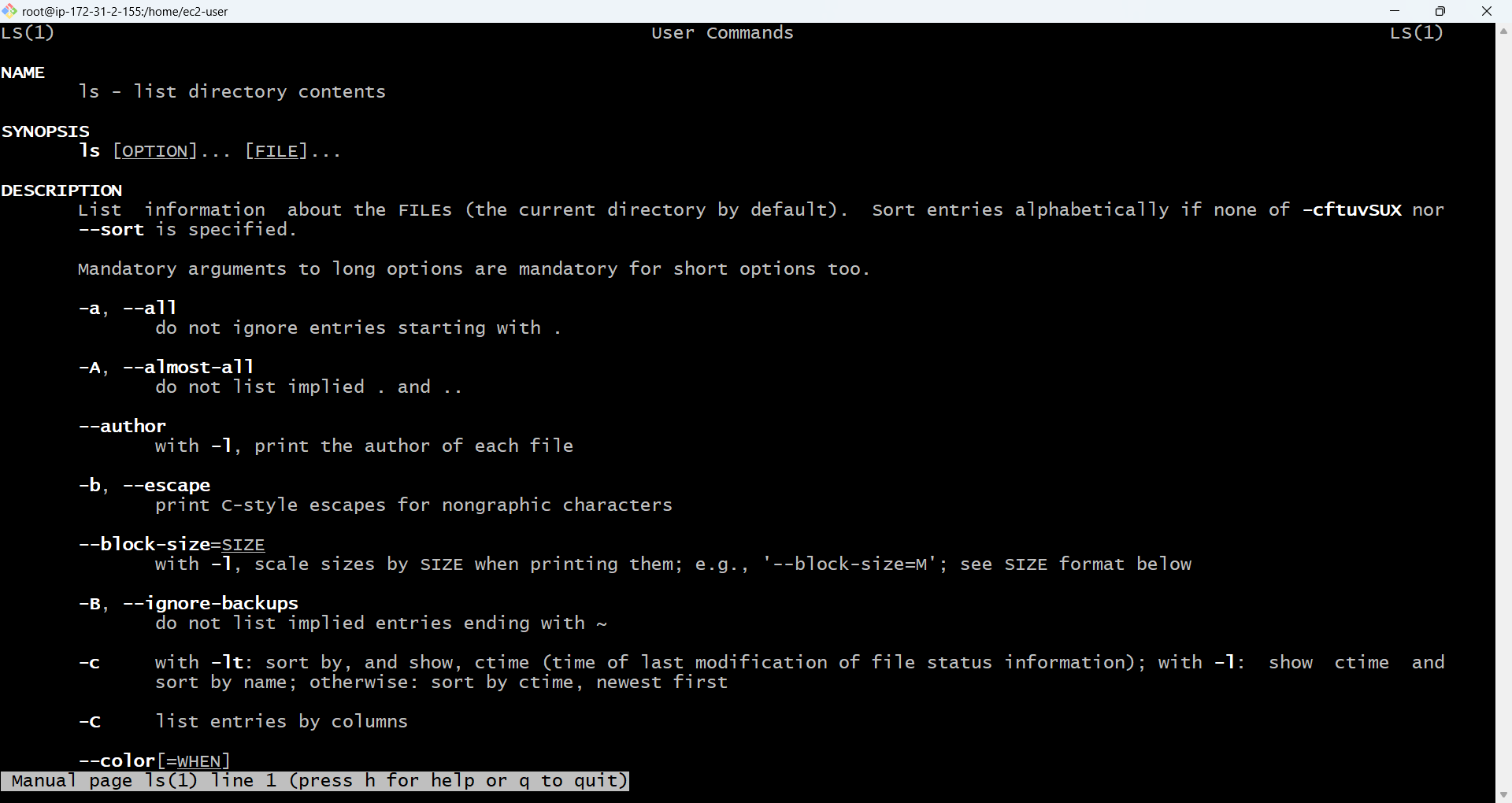
**LINUX COMMANDS**

**1.man**

🡪This man command will give the total information the commands in a huge amount of type of lists.

🡪for example: man ls

Output: 

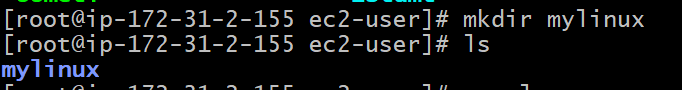
* The above screenshot is an just an example of it listing the commands there are many below.

**2. ls**

🡪 before using this ls command let us make one new directory.

**Making a directory:**

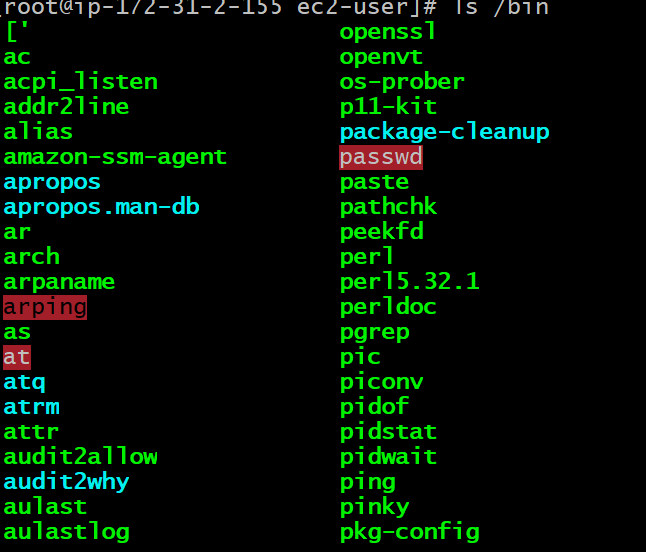
**Syntax:** mkdir <directory name>

**Output:** ****

* let us use ls command

**syntax: ls /foldername**

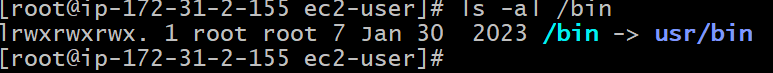
**example:** ls /bin

**output: **

**🡪**If you add a folder name or path, it will print that folder contents.

**Syntax: ls -al /bin**

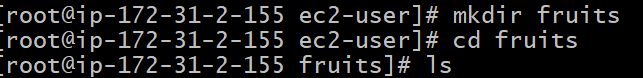
**🡪** compared to ls this will return the more information.

**Output: **

**3. change directory**

**🡪** Before going to change the directory let us create a new directory using mkdir .

**Syntax for cd: cd directory**

**Example : **

* In the above example we can see that we have created the fruits directory and changed the directory from ec2-user to fruits using cd.

**Syntax: cd ..**

* We can use this cd .. to directly move to the parent directory of folder.

Example: 

**Syntax : cd .**

* This does nothing but exists in the same directory unitl we change.



**Syntax : / dir name**

* We can use this absolute path that which starts from root folder.

**4 . pwd**

* This command will tell us about that in which folder path we are using and it will print that folder path.



**5. mkdir**

**🡪**  by using this mkdir we can create a new folder.



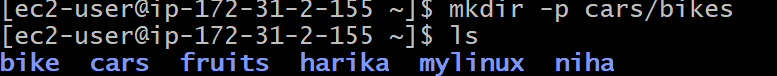
**Syntax: mkdir name1 name2**

* By using this command we can create the multiple folders under the single command.

****

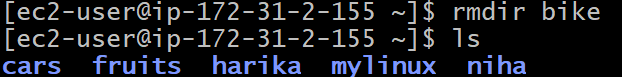
**Syntax: mkdir -p name1/name2**

* By using this command we can create a folder within the folder which means nested folders by adding -p.

****

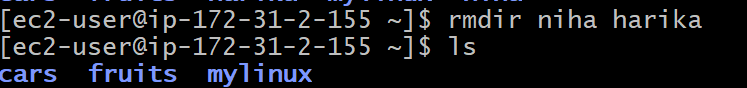
**6. rmdir**

🡪 by using this we can remove the directory which we have created.



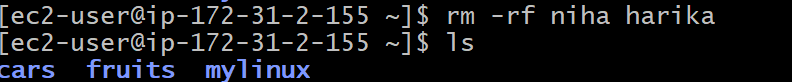
**Syntax: rmdir name1 name2**

* By using this we can remove the multiple directories.

****

* we can remove the files by using rm command , but for deleting the folders we will use the -rf command.

**Syntax: rm -rf name1 name2**

****

* **we** should be very careful while using this command because the files which we are accepting to delete for recovring ti will be complex and there is no bin.

**Mv**

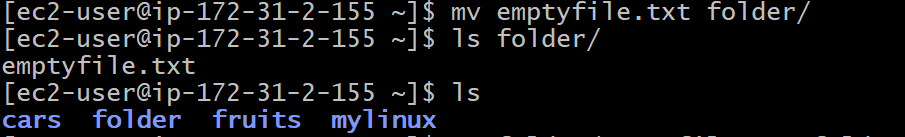
**🡪**the mv command is used to move or rename the files in linux.

🡪here file is created using touch

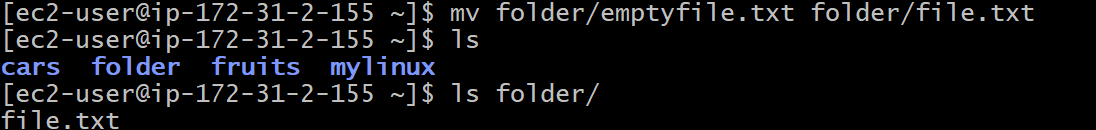


* touch test creates an empty file named **emptyfile.txt** in the current directory.
* And also create a directory name folder.

**Moving or Renaming a File with mv:**

****

**Verifying and renaming the file.**

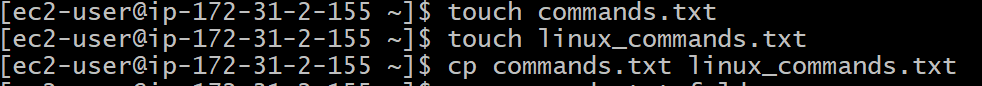
****

**Cp**

**🡪**The cp (copy) command is used to copy files and directories from one location to another.

**1. Copy a File**

**Command:**

****

* This copies commands.txt and creates a duplicate named linux\_commands.txt in the same directory.
* The original commands.txt remains unchanged**.**

**2. Copy a File to Another Folder**

**Command:**

****

**🡪**This copies commands.txt into the folder/ directory.

🡪The original commands.txt stays in its place.

**3. Copy a Folder (-r for Recursive)**

**Command:**

****

* -r is used to recursive to copy entrie directories.
* This copies commands.txt and all its contents into the folder/ folder

**4. Preserve File Attributes (-p option)**

**Command:**



🡪The -p flag keeps the original file’s **timestamps, ownership, and permissions**.

**5. Prompt Before Overwriting (-i option)**

**Command:**



🡪If commands.txt already exists in folder/, it asks **before overwriting**.

**6. Force Copy Without Prompt (-f option)**

**Command:**

****

**🡪**The -f (force) option overwrites the file without asking**.**

**7. Verbose Mode (-v option)**

**Command:**

****

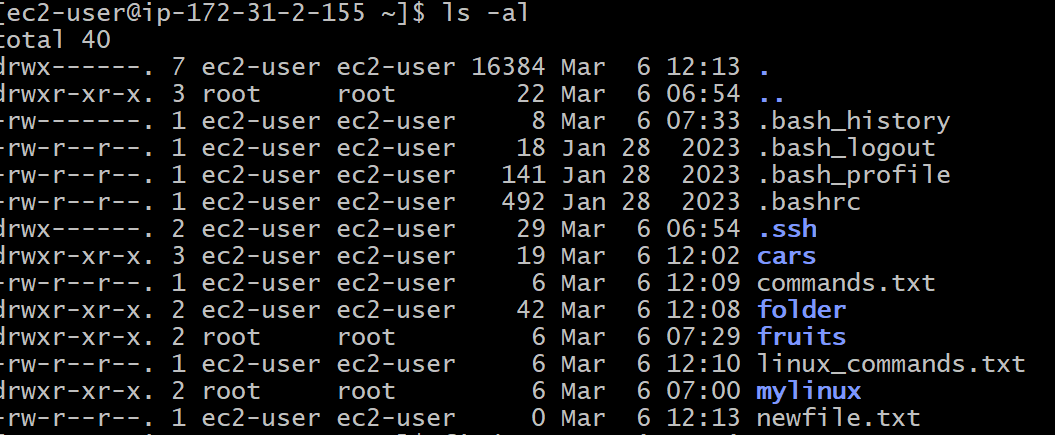
* It will show what is being copied.

**Touch**

🡪 You can create an empty file using the

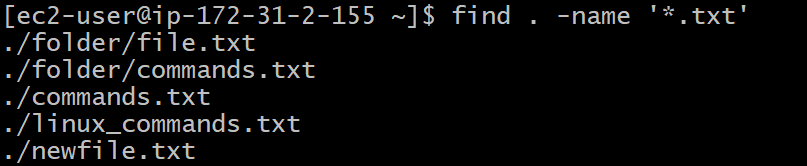
command: touch file.txt

🡪 If the file already exists, it opens the file in write mode, and the timestamp of the file is updated



**Find**

🡪The find  command can be used to find files or folders matching a particular search pattern. It searches recursively.

🡪Find all the files under the current tree that have the .txt  extension and print the relative path of each file matching: 

🡪It's important to use quotes around special characters like \*  to avoid the shell interpreting them. Find directories under the current tree matching the name "src":

find . -type d -name src.

🡪Use -type f  to search only files, or search symbolic links.-type l  to only

🡪-name  is case sensitive. use -iname  to perform a case-insensitive search. You can search under multiple root trees:

find folder1 folder2 -name filename.txt

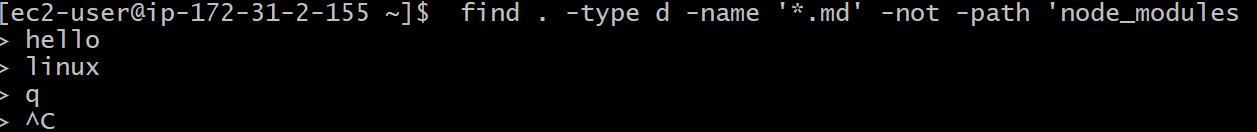
ex: 

🡪Find directories under the current tree matching the name "node\_modules" or 'public':

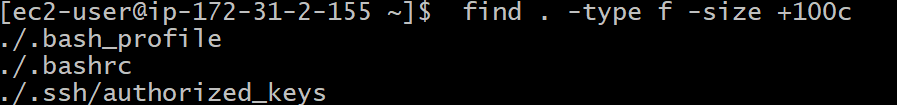


🡪You can also exclude a path, using -not -path :

find . -type d -name '\*.md' -not -path 'node\_modules

ex: 

🡪You can search files that have more than 100 characters (bytes) in them: find . -type f -size +100c

Ex: 

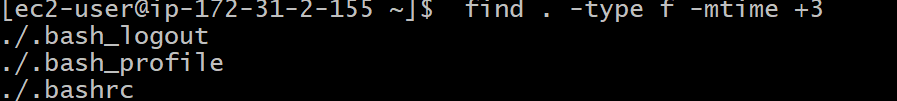
🡪Search files bigger than 100KB but smaller than 1MB:

find . -type f -size +100k -size -1M



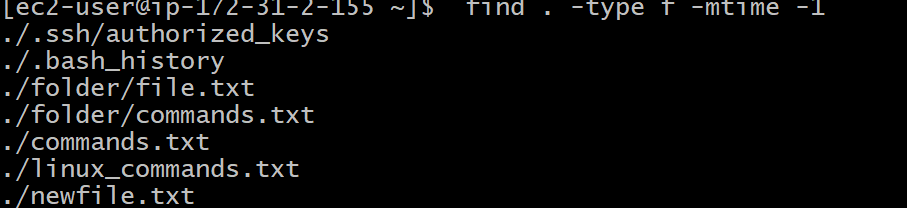
🡪Search files edited more than 3 days ago

find . -type f -mtime +3



🡪Search files edited in the last 24 hours

find . -type f -mtime -1



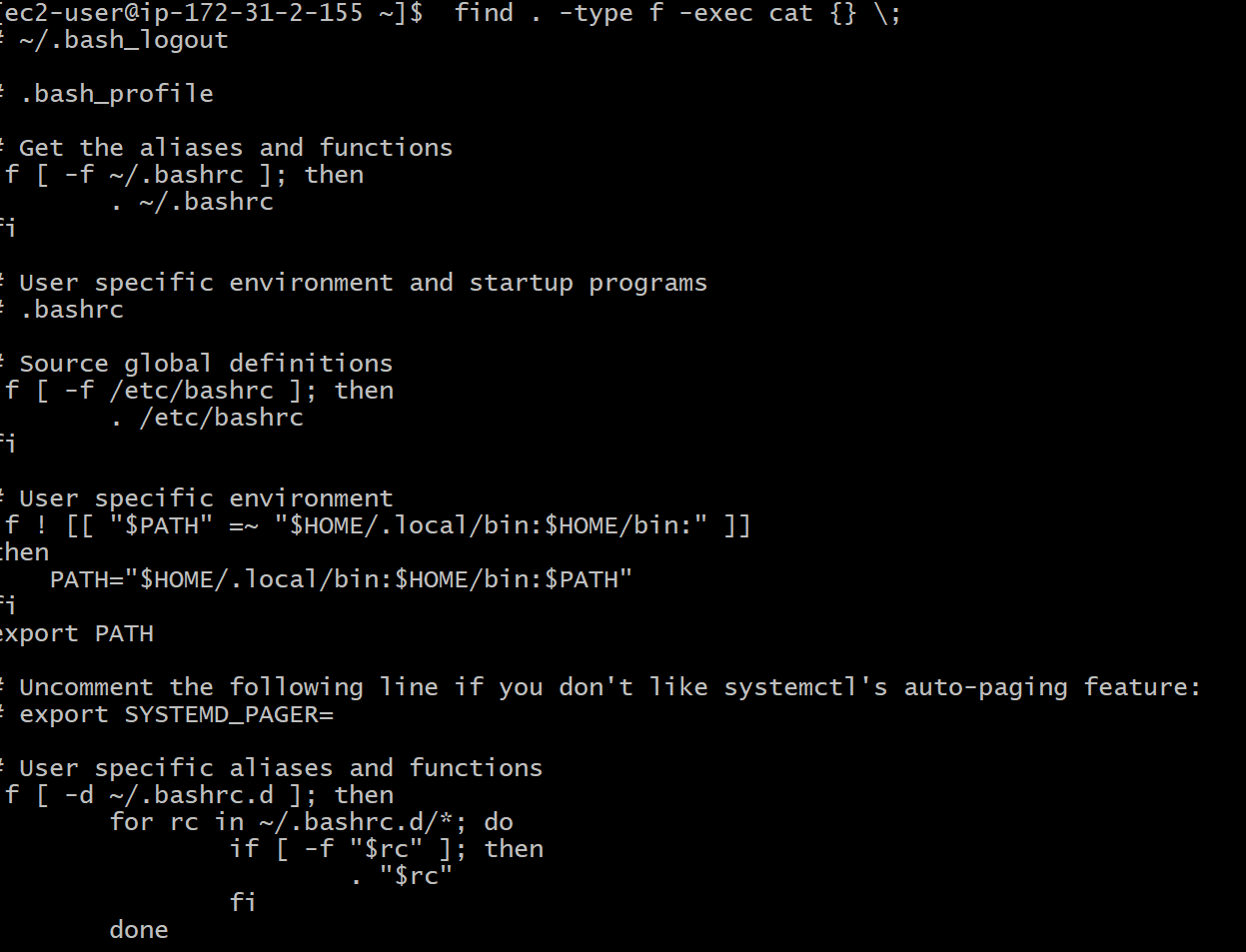
🡪You can delete all the files matching a search by adding the -delete  option. This deletes all the files edited in the last 24 hours:

find . -type f -mtime -1 -delete

🡪You can execute a command on each result of the search. In this example we run cat  to print the file content:

find . -type f -exec cat {} \;

notice the terminating \; . {}  is filled with the file name at execution time.



**gzip**

You can compress a file using the gzip compression protocol named LZ77 using the gzip  command. Here's the simplest usage:

gzip filename



🡪This will compress the file, and append a .gz  extension to it. The original file is deleted. To prevent this, you can use the -c  option and use output redirection to write the output to the

filename.gz

file:

gzip -c filename > filename.gz  


🡪use the -k  option:

gzip -k filename



specific level with the option:-

gzip -1 filename



🡪You can compress multiple files by listing them:

gzip filename1 filename2

🡪You can compress all the files in a directory, recursively, using the -r  option:

gzip -r a\_folder



🡪The -v  option prints the compression percentage information. Here's an example of it being used along with the -k  (keep) option:



🡪gzip can also be decompress a file, using the -d option.



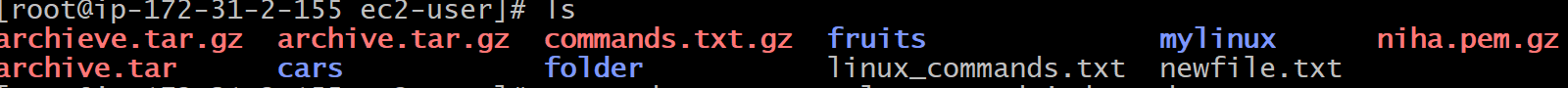


**Gunzip**

**🡪**The gunzip  command is basically equivalent to the gzip  command, except the enabled by default.-d  option is always The command can be invoked in this way:

gunzip filename.gz





🡪here the newfile.txt is unzipped.

🡪This will gunzip and will remove the putting the result in the .gz  extension, filename  file. If that file exists, it will overwrite that.

🡪You can extract to a different filename using output redirection using the -c  option.

gunzip -c filename.gz > anotherfilename

**tar**

🡪The tar  command is used to create an archive, grouping multiple files in a single file.

🡪 Its name comes from the past and means tape archive. Back when archives were stored on tapes.

🡪This command creates an archive named archive.tar  with the content of file1  and file2 :

tar -cf archive.tar file1 file2



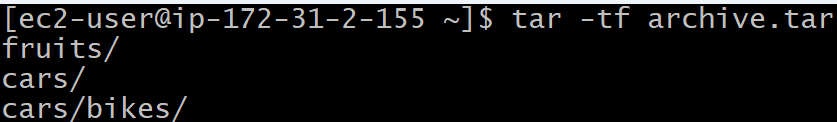
🡪The c  option stands for create. The f  option is used to write to file the archive.

🡪To extract files from an archive in the current folder, use:

tar -xf archive.tar



🡪list the files contained in an archive:



🡪tar  is often used to create a compressed archive, gzipping the archive.

This is done using the z  option:

tar -czf archive.tar.gz file1 file2



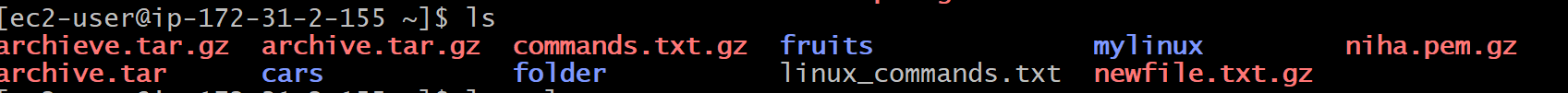
🡪This is just like creating a tar archive, and then running gzip  on it.

🡪 To unarchive a gzipped archive, you can use or gunzip , gzip -d , and then unarchive it, but tar -xf  will recognize it's a gzipped archive, and do it for you: tar -xf archive.tar.gz

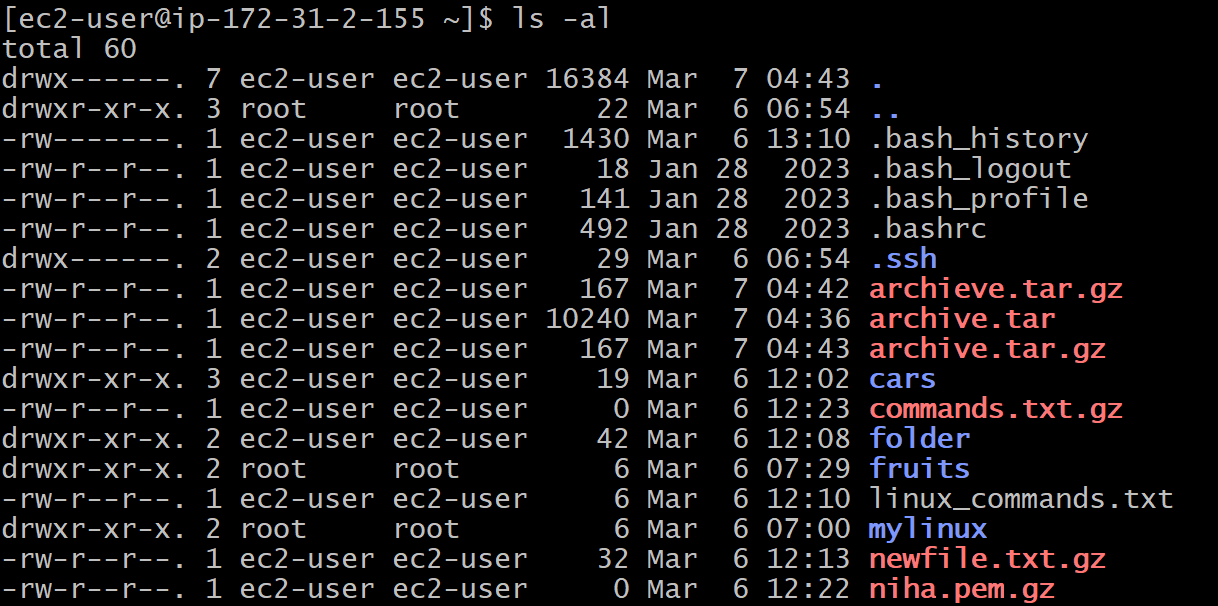


**Alias**

* By default it prints very little information by using ls command.



🡪while using the -al  option it will print something more useful, including the file modification date, the size, the owner, and the permissions, also listing hidden files (files starting with a . :

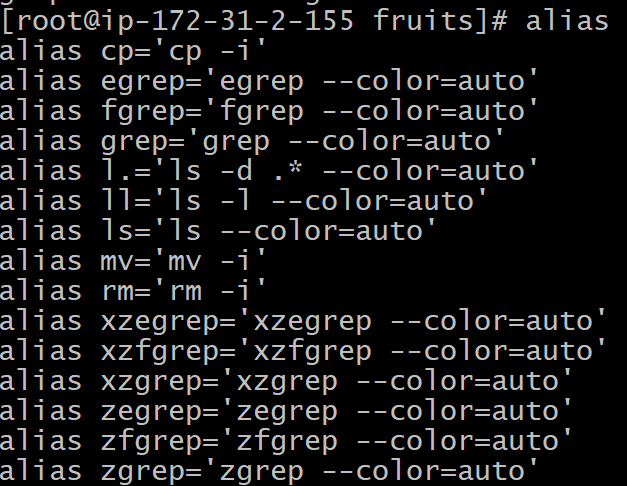


🡪You can create a new command, for example I like to call it ll , that is an alias to ls -al . You do it in this way:

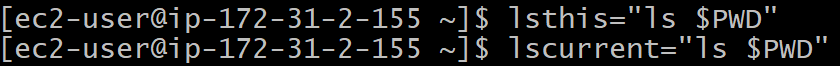
alias ll='ls -al



* Now calling alias  without any option will list the aliases defined.



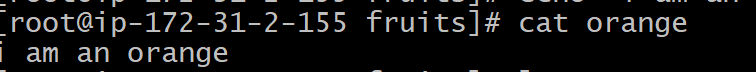
🡪$PWD refers to the current folder the shell is into. If you now navigate away to a new folder, lscurrent  lists the files in the new folder, lsthis  still lists the files in the folder you were when you defined the alias



**Cat**

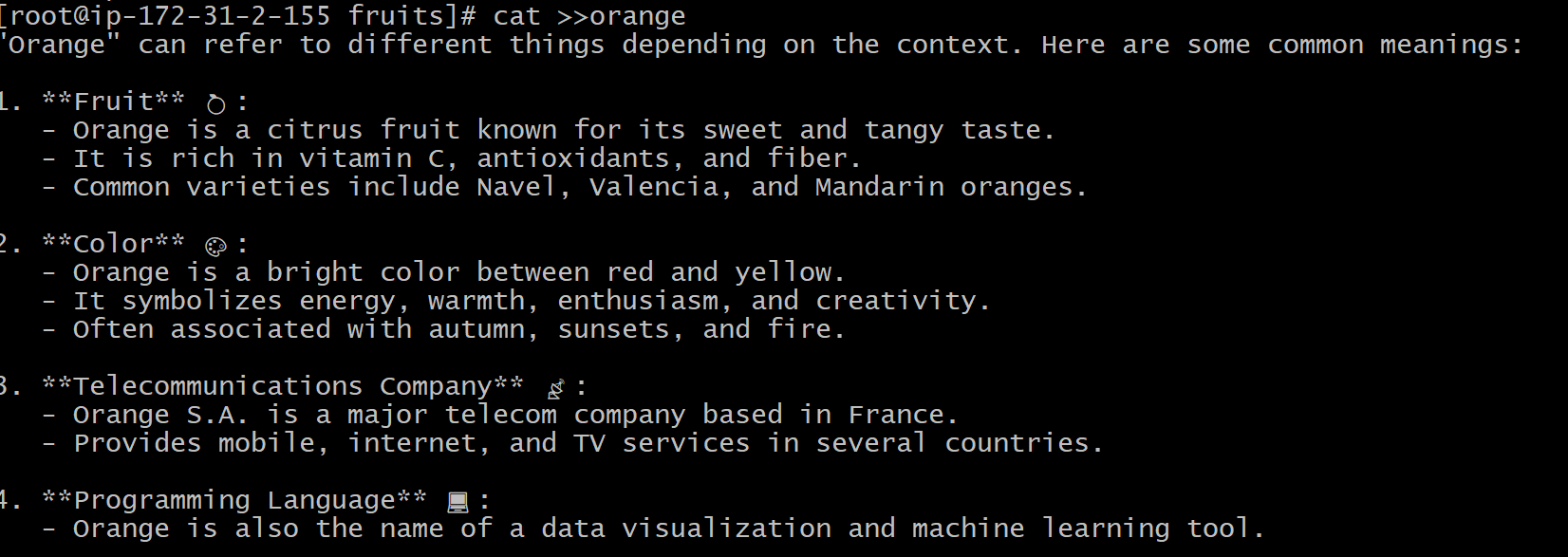
* cat  can also add content to a file, and this makes it super powerful. In its simplest usage, cat  prints a file's content to the standard output:

cat file

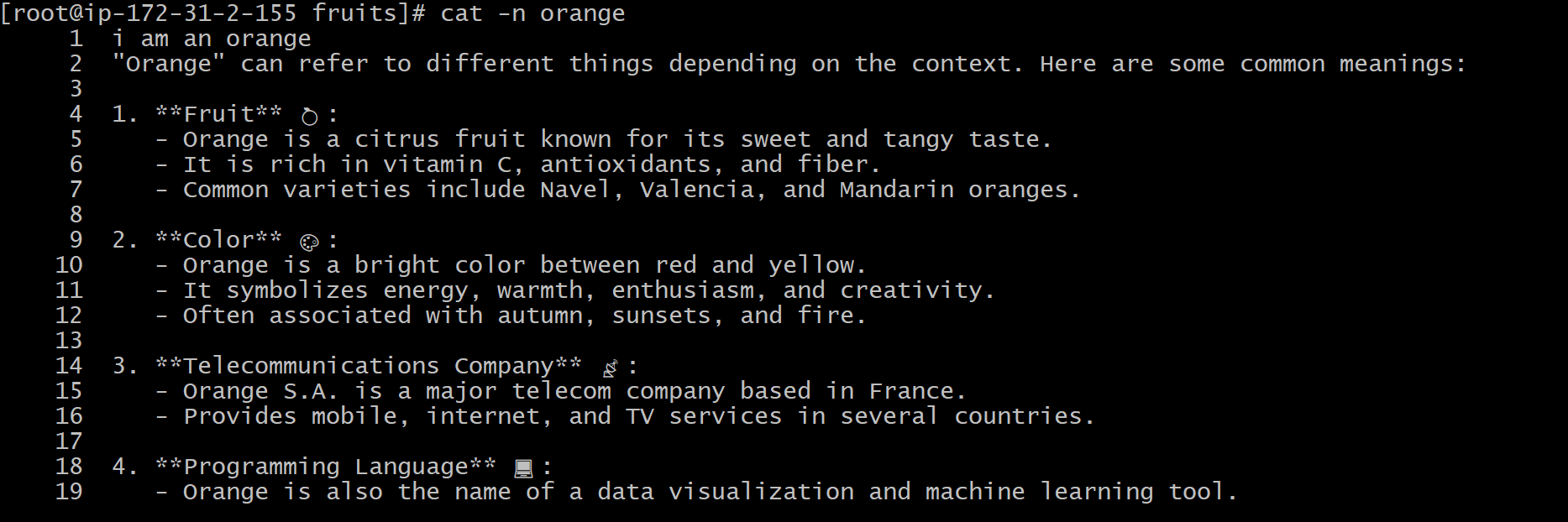


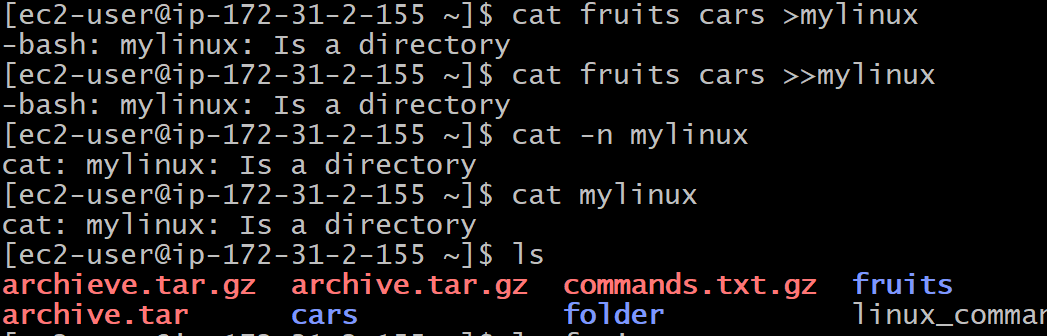
🡪redirection operator >  you can concatenate the content of multiple files into a new file:

cat file1 file2 > file3



🡪When watching source code files it's great to see the line numbers, and you can have cat  print them using the -n  option: cat -n file1





**Less**

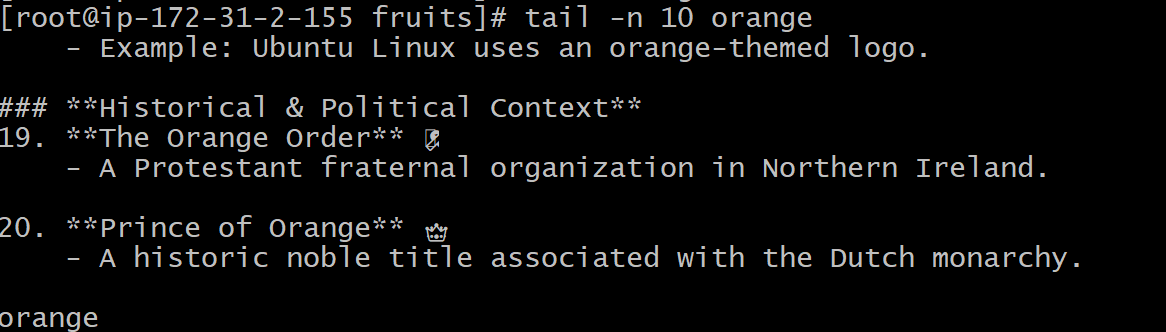
🡪 The less  command is one I use a lot. It shows you the content stored inside a file, in a nice and interactive UI. Usage: less  .



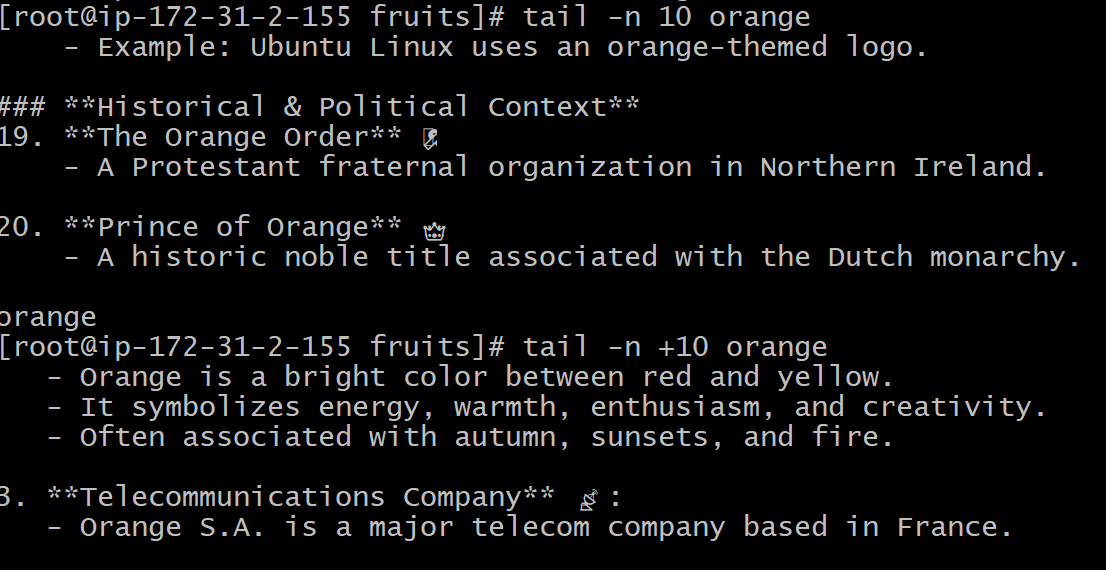
🡪Once you are inside a less  session, you can quit by pressing q .

Output: 

**Tail**

**🡪You can print the last 10 lines in a file**

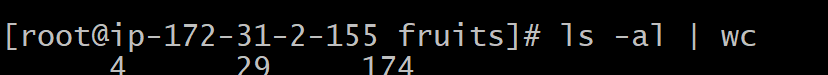
**🡪**You can print the whole file content starting from a specific line using +  before the line number



**Wc**

**🡪**The wc  command gives us useful information about a file or input it receives via pipes**.**

**🡪**we can count the output of running the ls -al  command



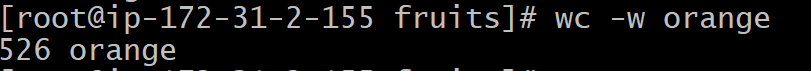
🡪The first column returned is the number of lines. The second is the number of words. The third is the number of bytes.

🡪We can tell it to just count the lines.

🡪wc – 1 orange



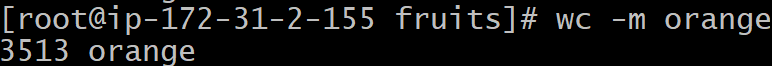
🡪just the words: wc -w orange



🡪just the bytes: wc -c orange



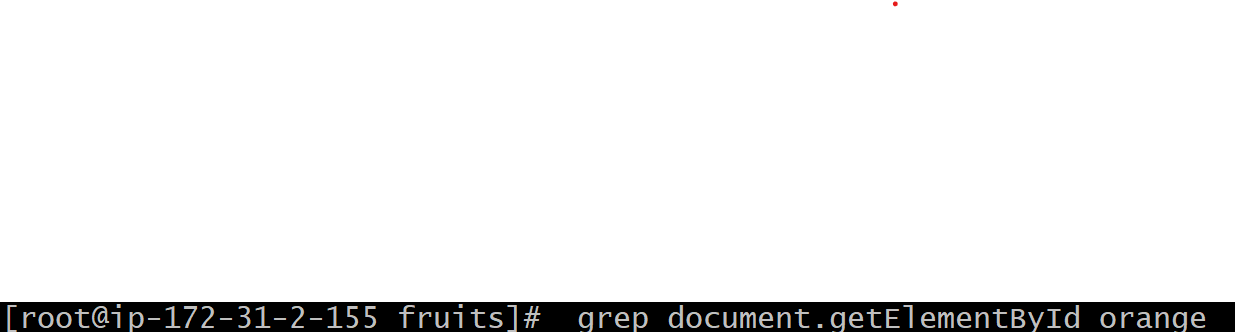
🡪the -m  flag will help getting the correct value: wc -m orange



**Grep**

**🡪**The grep  command is a very useful tool, You can use grep  to search in files, or combine it with pipes to filter the output of another command.

🡪For example here's how we can find the occurences of the document.getElementById  line in the index.md  file:

grep document.getElementById orange.

🡪Using the -n  option it will show the line numbers:

grep -n document.getElementById orange



🡪One very useful thing is to tell grep to print 2 lines before, and 2 lines after the matched line, to give us more context. That's done using the -C  option, which accepts a number of lines:

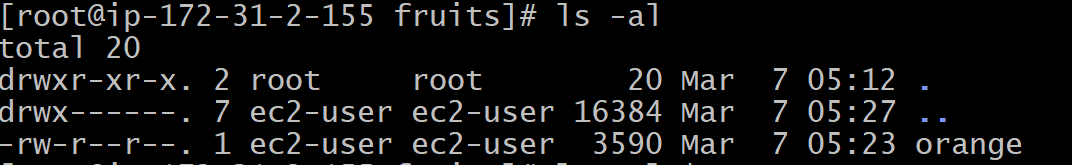
grep -nC 2 document.getElementById index.md

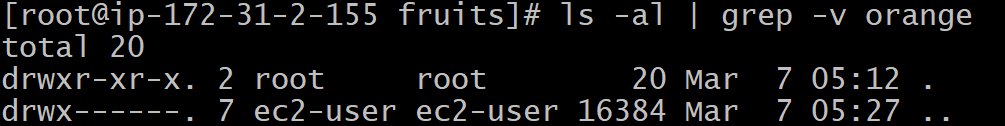




🡪The search string can be a regular expression, and this makes grep  very powerful.

🡪, using the -v  option:





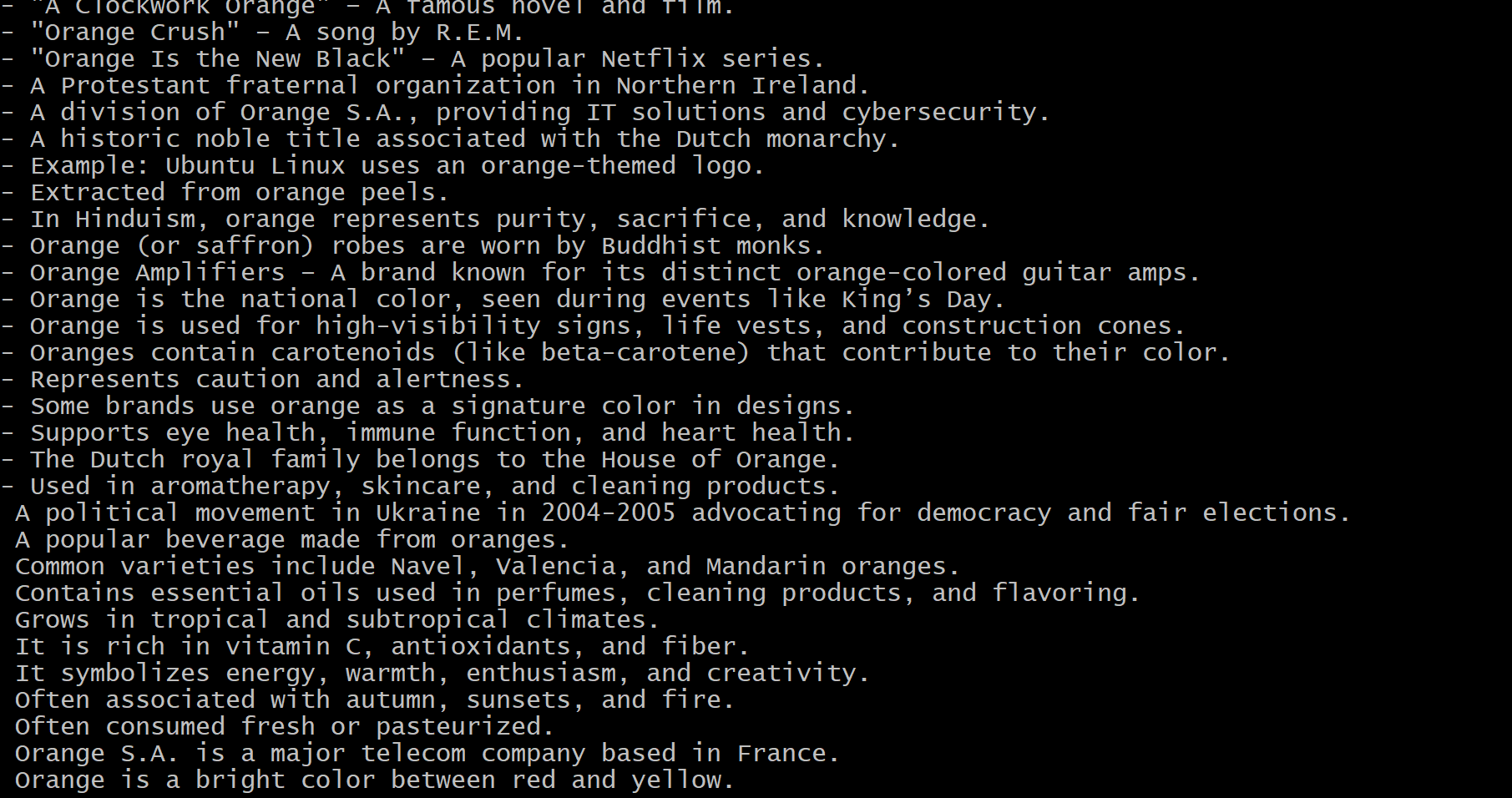
**Sort**

🡪This list is unordered.

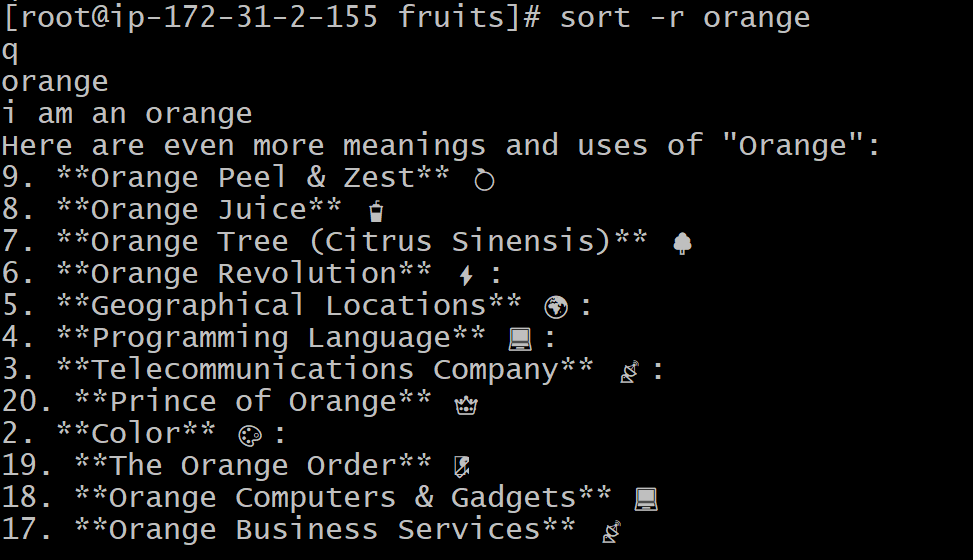
🡪The sort  command helps us sorting them by name.



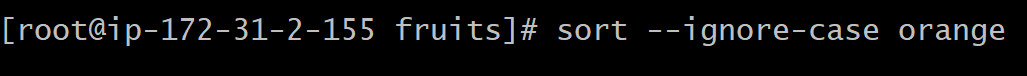
Output:

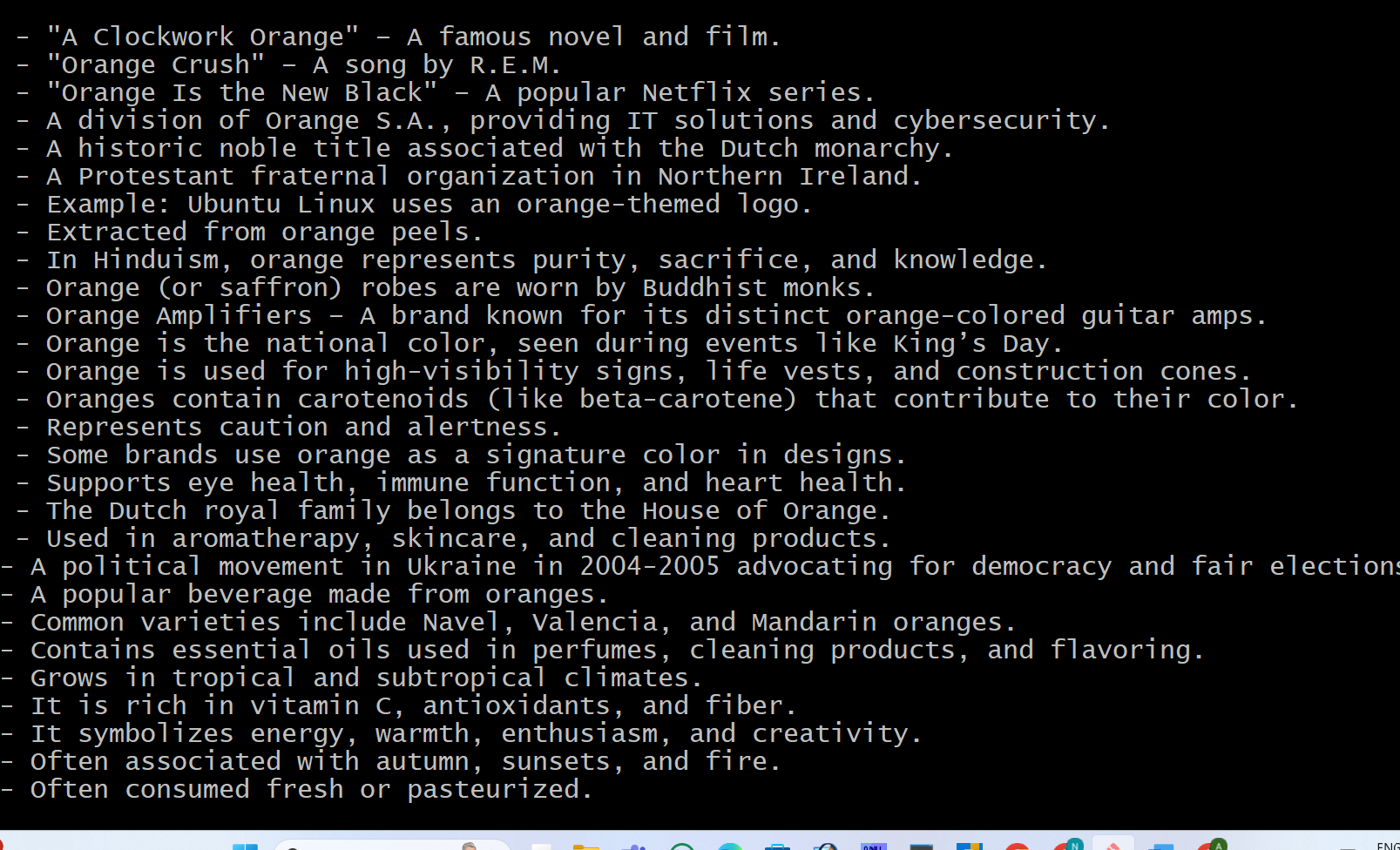


🡪Use the r  option to reverse the order:

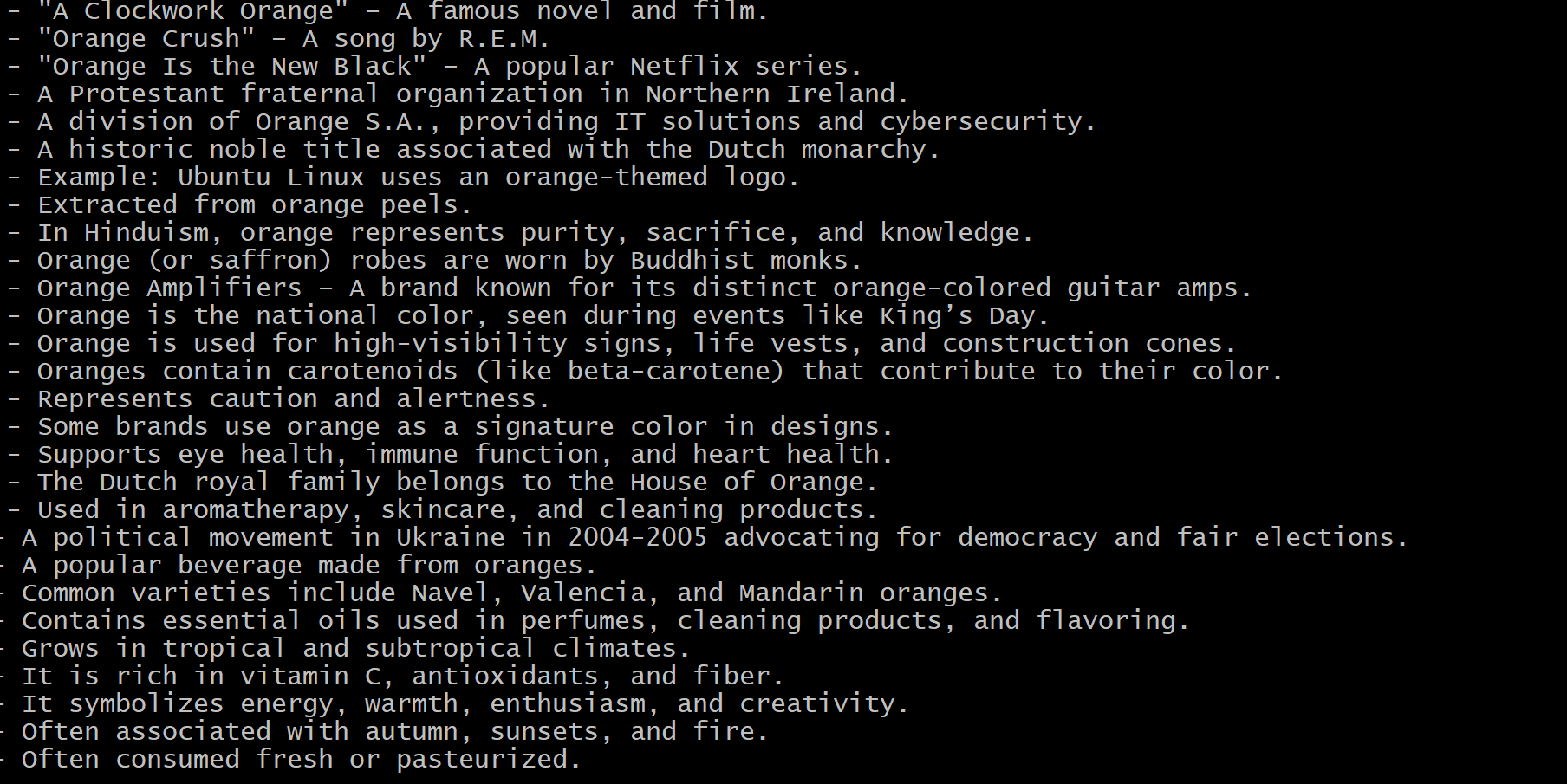


🡪Sorting by default is case sensitive, and alphabetic. Use the --ignore-case  option to sort case insensitive, and the -n  option to sort using a numeric order.



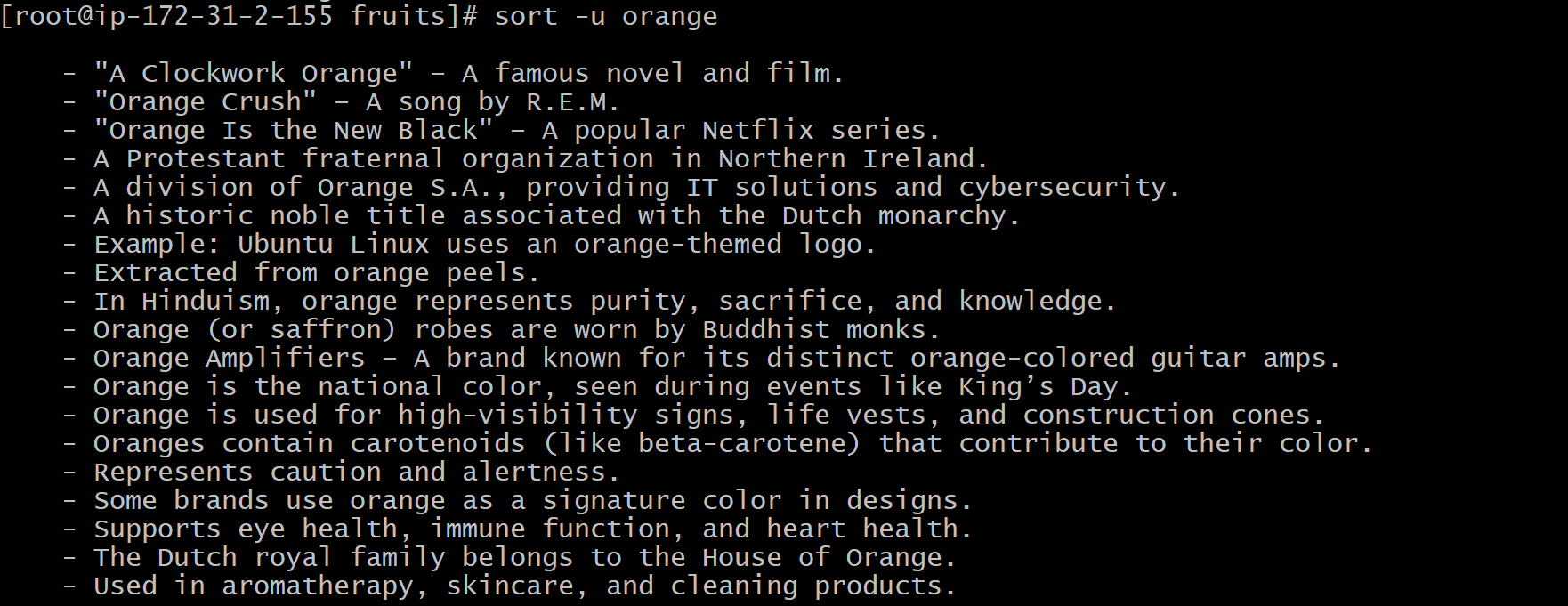






🡪If the file contains duplicate lines:

🡪You can use the -u  option to remove them:



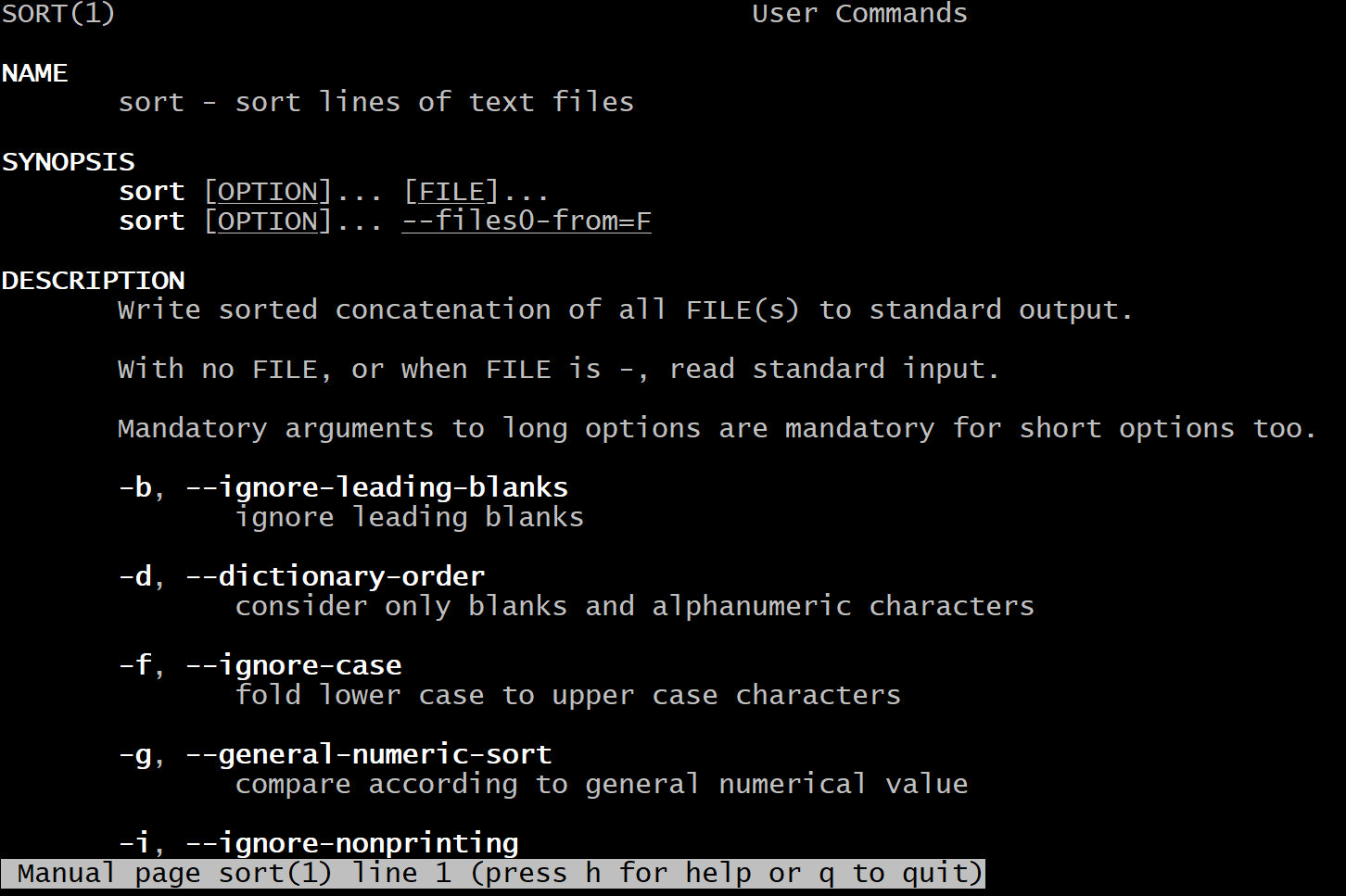
🡪can order the files returned by ls  with:

ls | sort



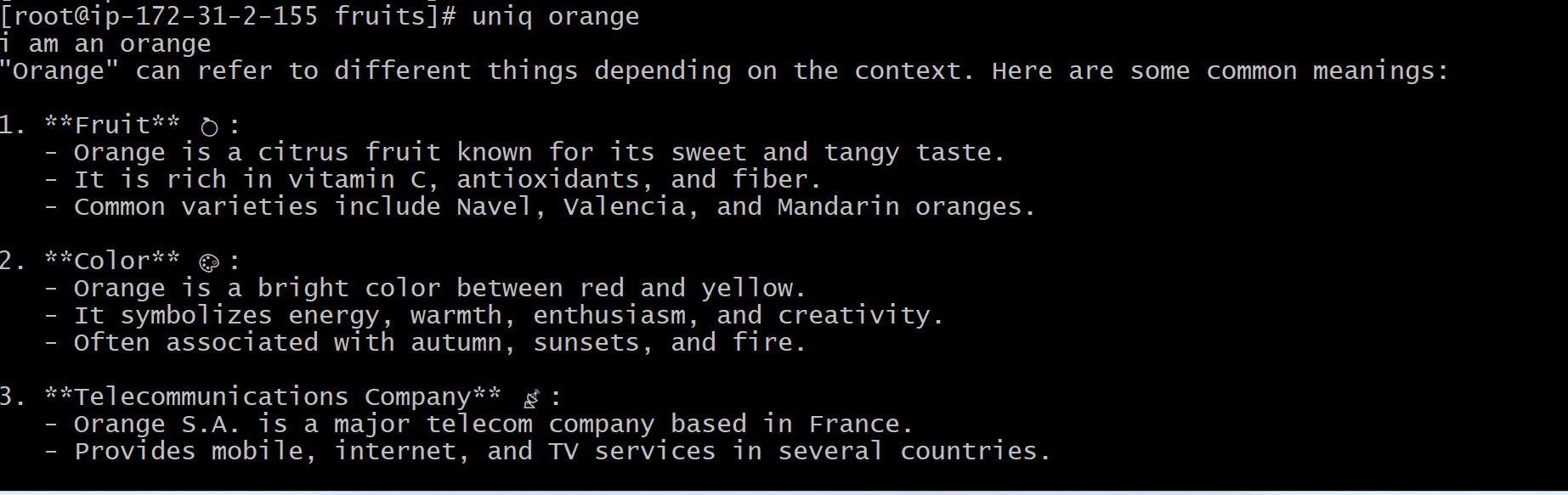
🡪sort  is very powerful and has lots more options, which you can explore calling man sort .



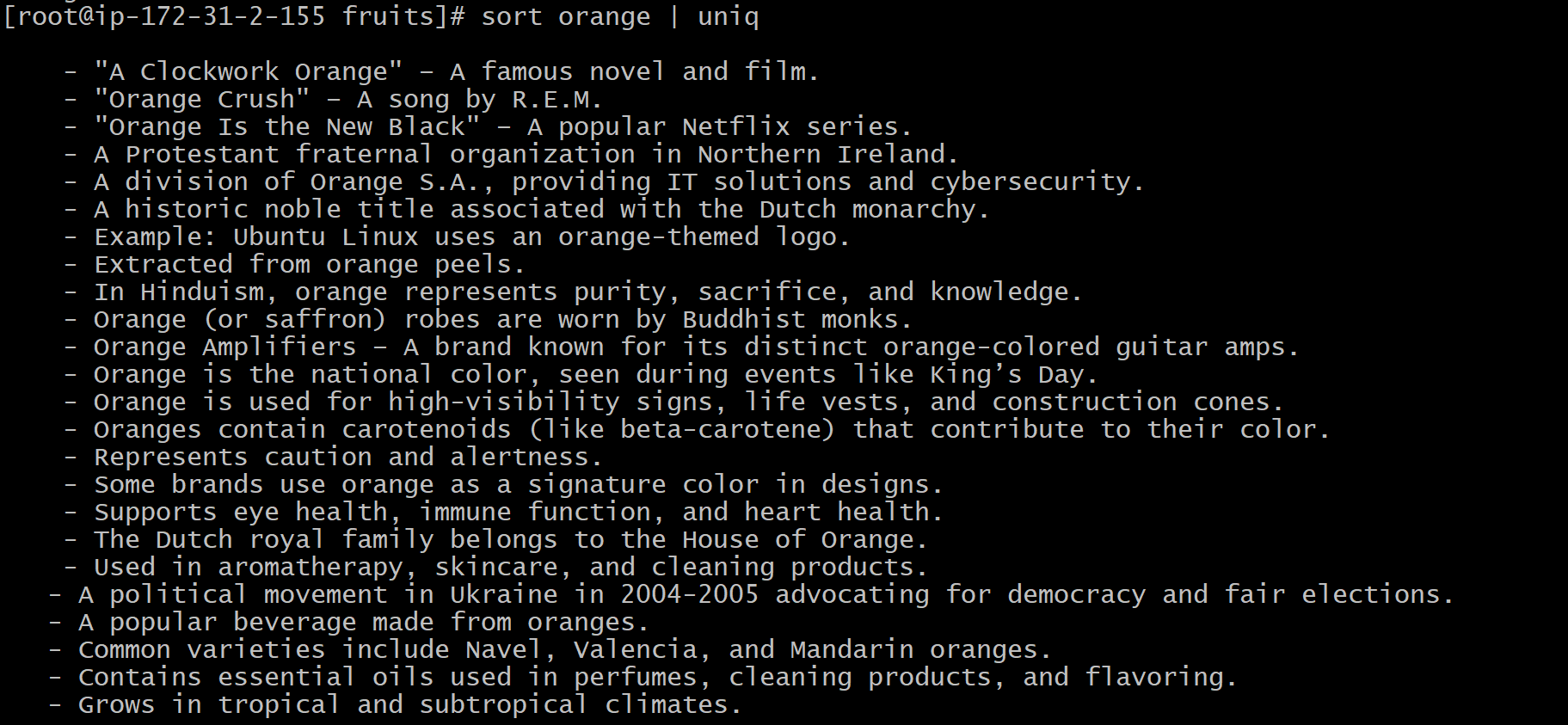


**Uniq**

🡪uniq  is a command useful to sort lines of text.



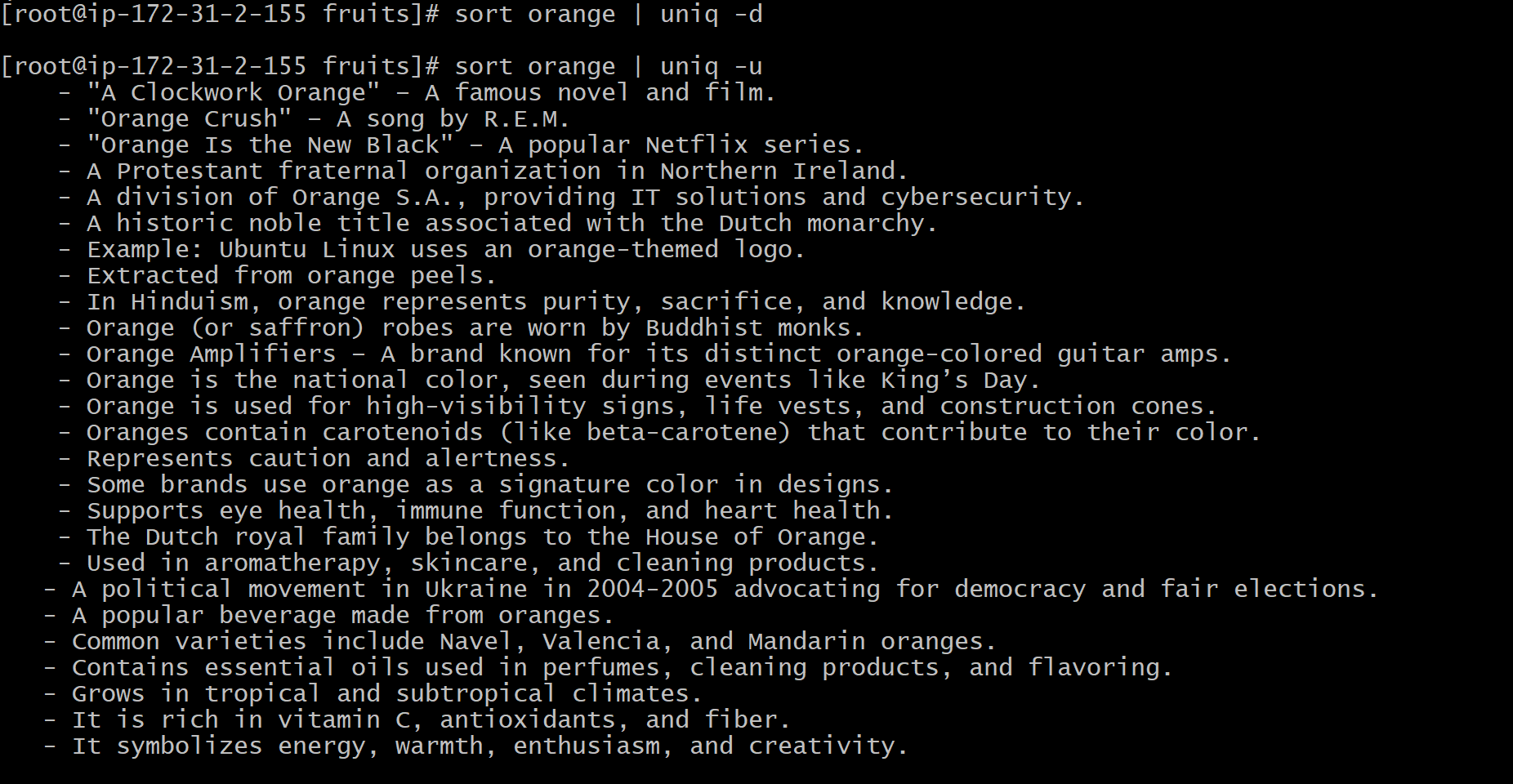
🡪This implies that you will most likely use it along with sort .



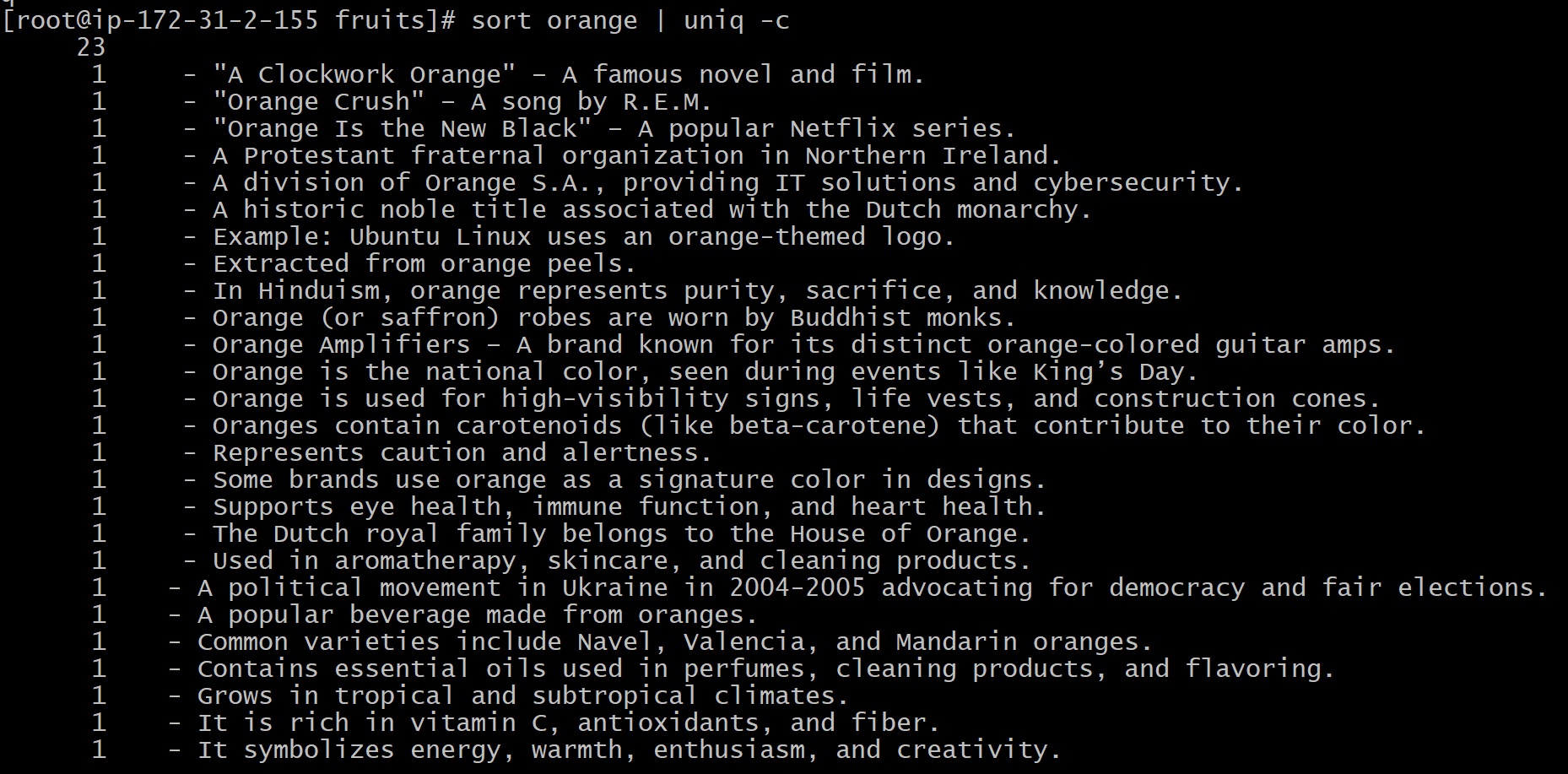
🡪You can tell it to only display duplicate lines, for example, with the -d  option:

sort orange | uniq -d

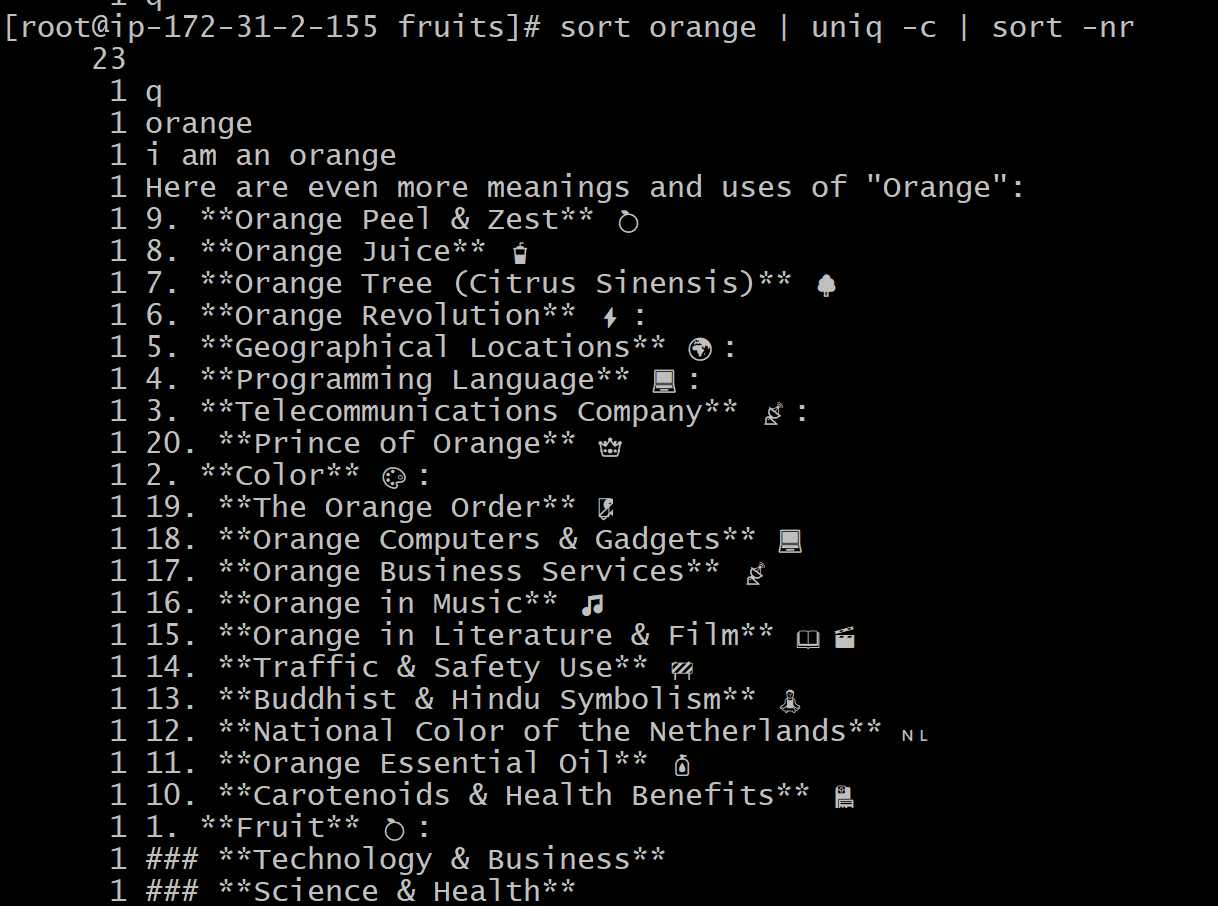
🡪You can use the -u  option to only display non duplicate lines



🡪You can count the occurrences of each line with the c  option.

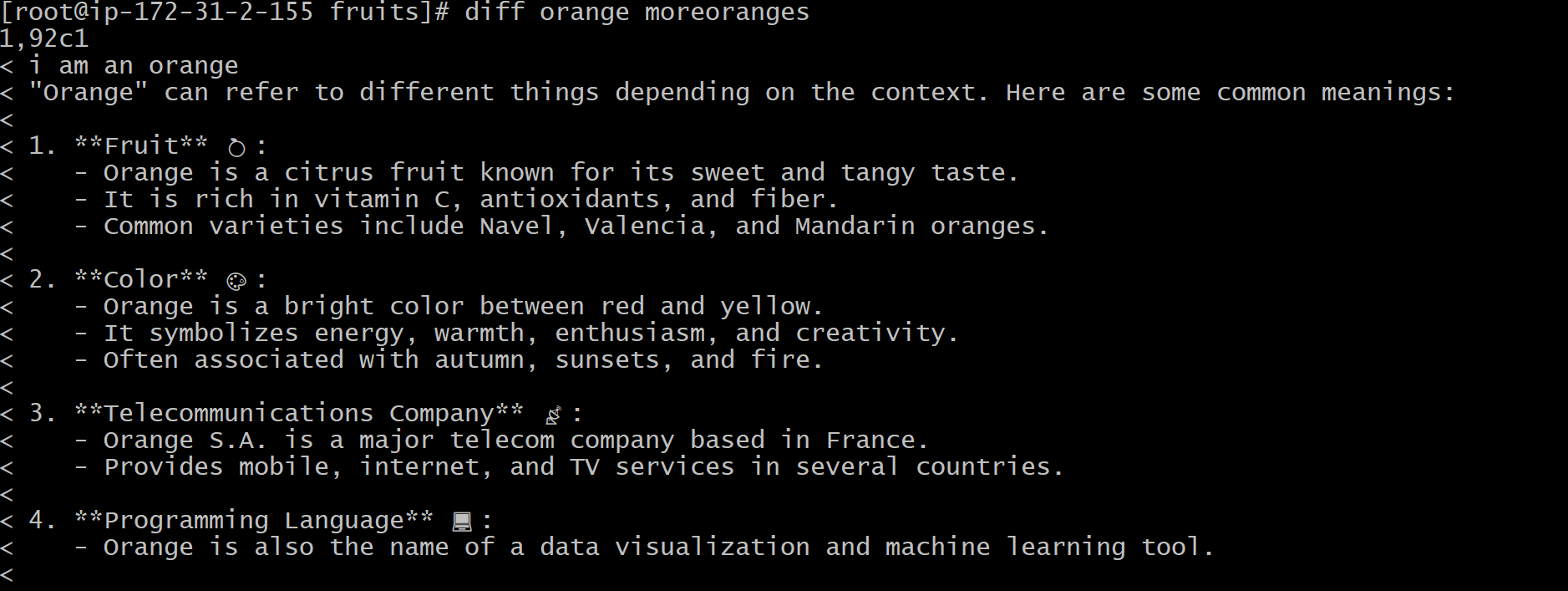


🡪Use the special combination: sort orange | uniq -c | sort -nr

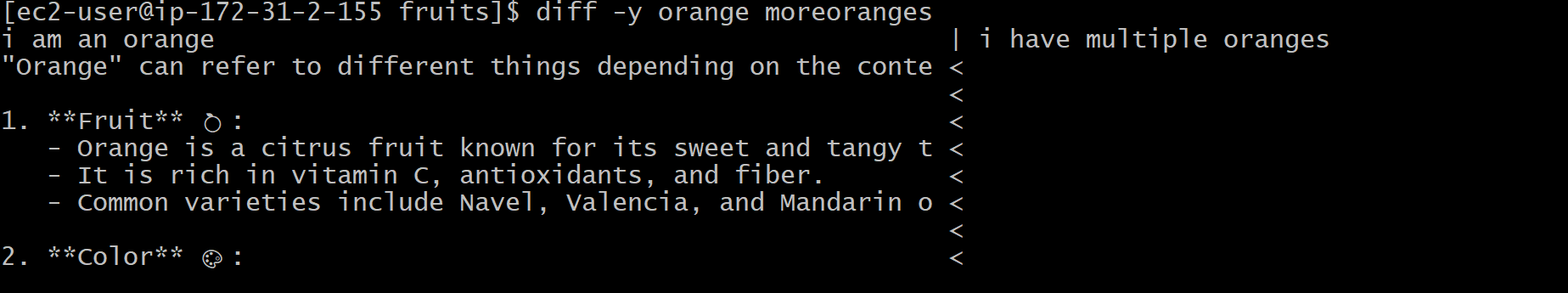


**Diff**

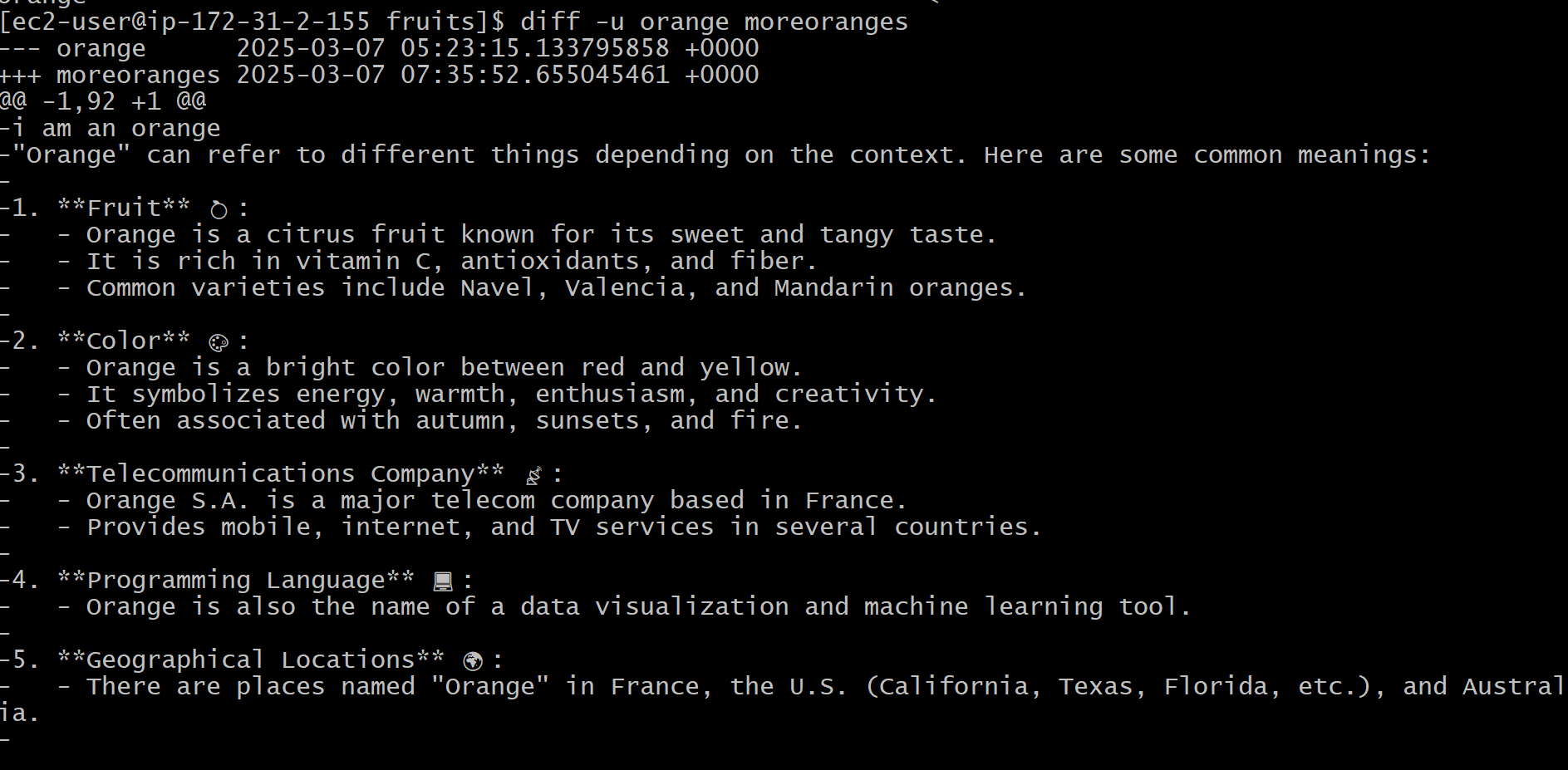
**🡪**diff  is a handy command. Suppose you have 2 files, which contain almost the same information, but you can't find the difference between the two. diff  will process the files and will tell you what's the difference. Suppose you have 2 files: oranges and moreoranges . The difference is that moredogs.txt  contains one more orange name:



🡪Using the -y  option will compare the 2 files line by line

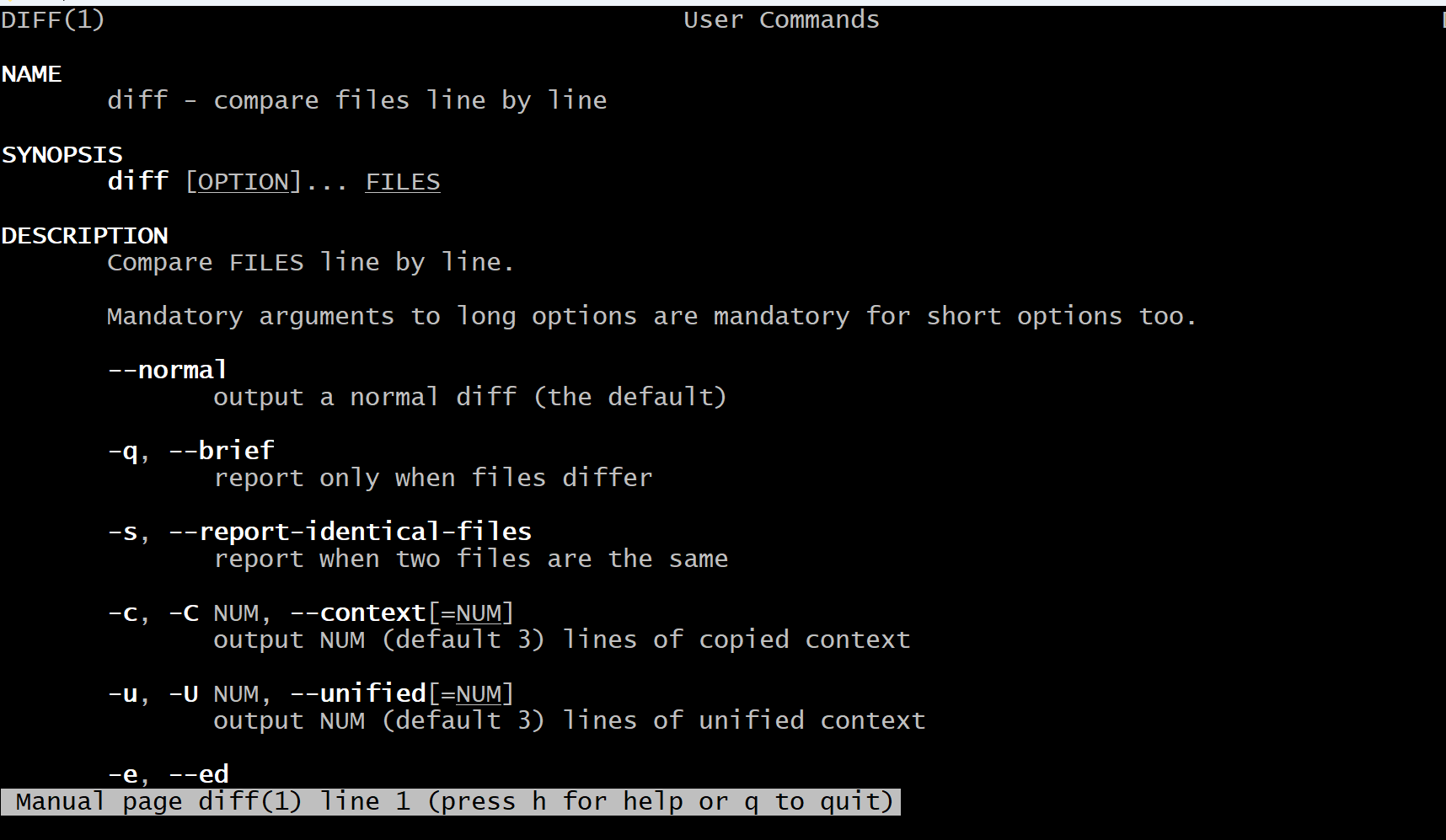


🡪The -u  option however will be more familiar to you, because that's the same used by the Git version control system to display differences between versions.



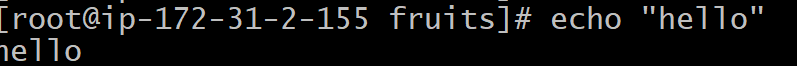


* it tell us in which files differ, rather than the content, use the r  and q  options:
* 
* There are many more options you can explore in the man page running man diff :



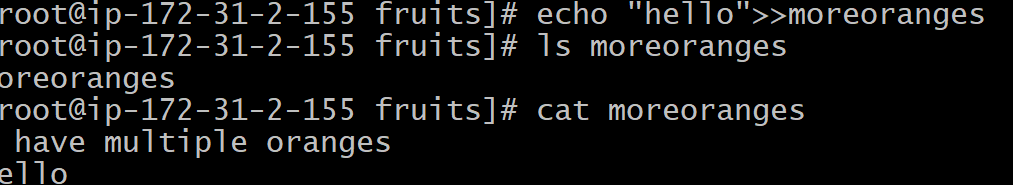
**echo**

🡪The echo  command does one simple job: it prints to the output the argument passed to it.

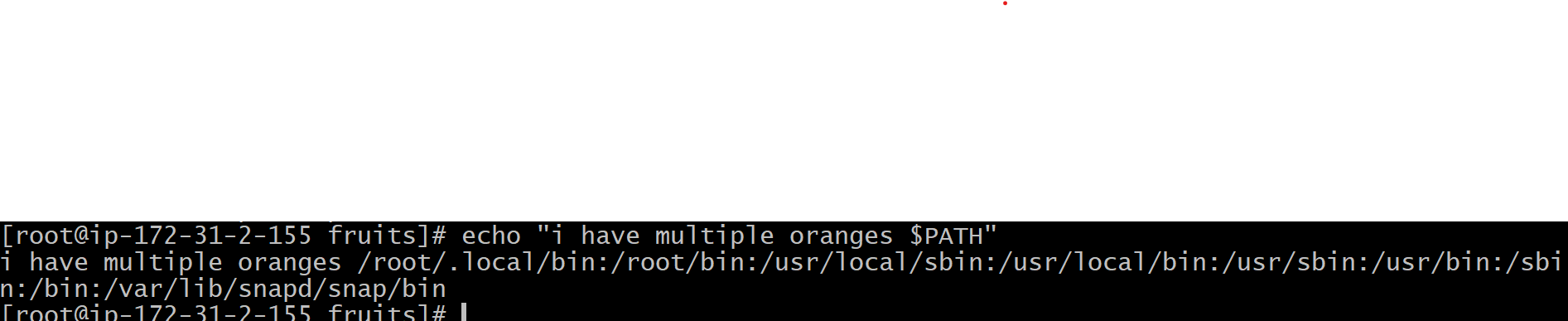
🡪

🡪This will print hello  to the terminal.

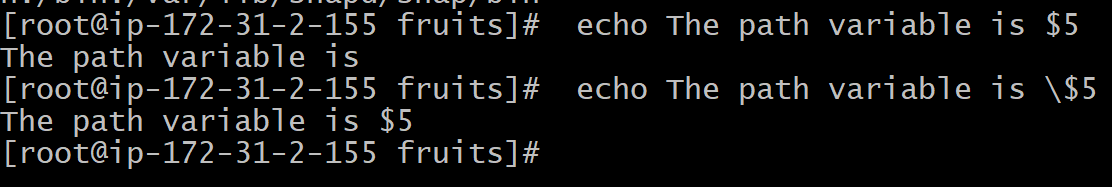
We can append the output to a file:



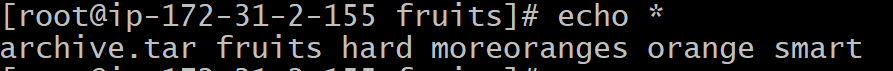
🡪We can interpolate environment variables:



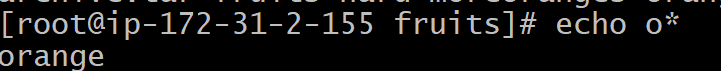
🡪Beware that special characters need to be escaped with a backslash \ .  $  for example



🡪We can echo the files in the current folder: echo \*



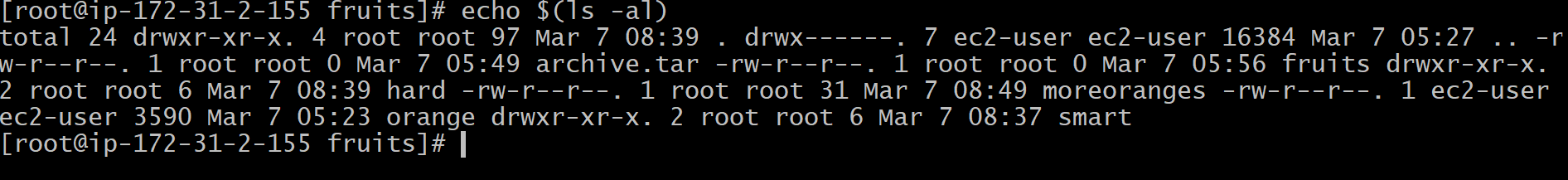
🡪We can echo the files in the current folder that start with the letter o : echo o\*



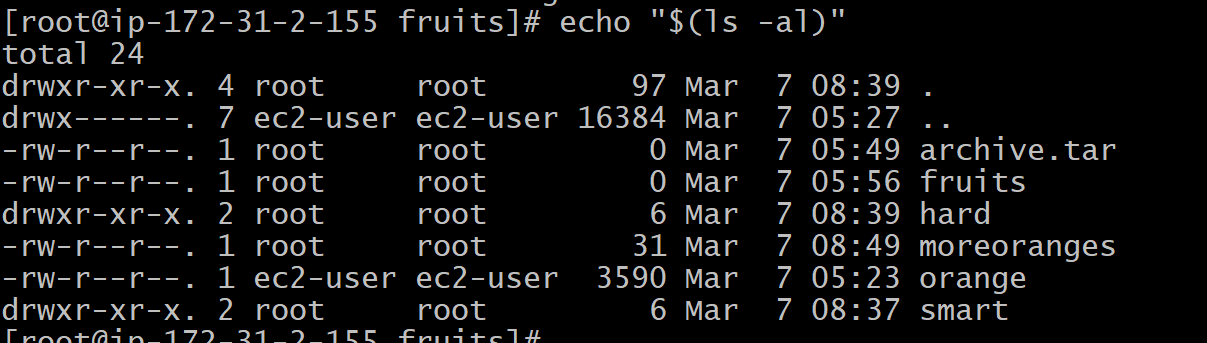
🡪Any valid Bash (or any shell you are using) command and feature can be used here. You can print your home folder path: echo ~



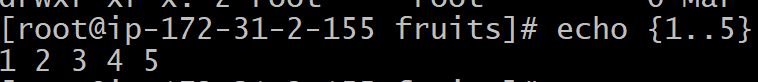
🡪You can also execute commands, and print the result to the standard output (or to file, as you saw): echo $(ls -al)



🡪Note that whitespace is not preserved by default. You need to wrap the command in double quotes to do so.



🡪You can generate a list of strings, for example ranges: echo {1..5}



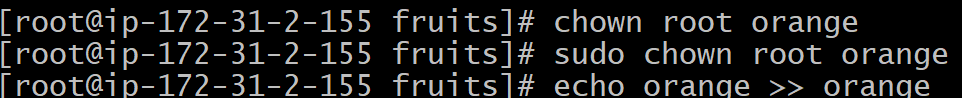
**Chown**

**🡪**Every file/directory in an Operating System like Linux or macOS (and every UNIX systems in general) has an owner. The owner of a file can do everything with it. It can decide the fate of that file. The owner (and the root  user) can change the owner to another user, too, using the chown  command:

Chown owner file



🡪You can use chown  to transfer the ownership to you.



🡪It's rather common to have the need to change the ownership of a directory, and recursively all the files contained, plus all the subdirectories and the files contained in them, too. You can do so using the -R  flag: chown -R

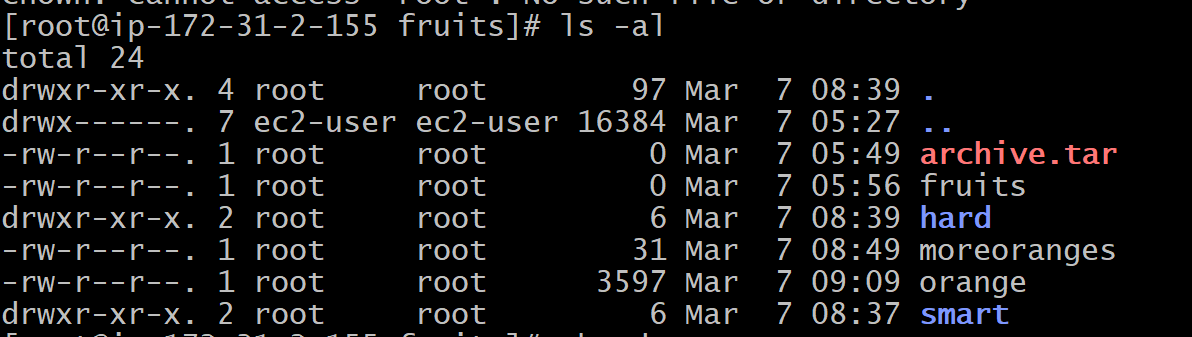


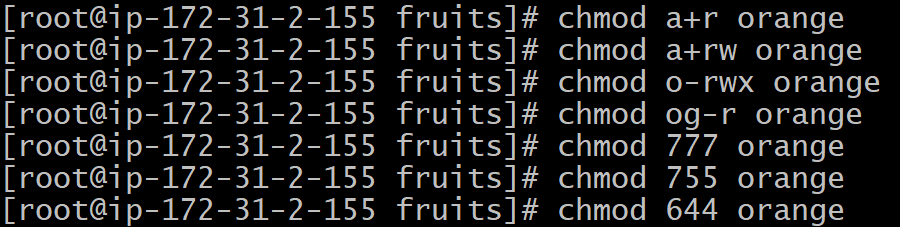
🡪You can also just change the group of a file using the chgrp  command:



**Chmod**

* Every file in the Linux / macOS Operating Systems (and UNIX systems in general) has 3 permissions: Read, write, execute. Go into a folder, and run the ls -al  command



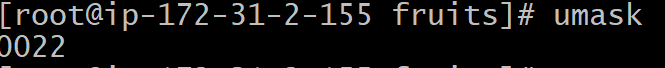


🡪the above are the commands used in chmod.

🡪to add a permission, or to remove it. Then you enter one or more permissions symbols ( r ,  w , x )

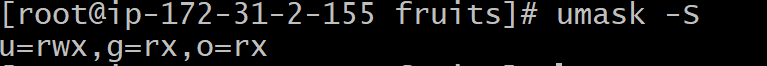
**Umask**

**🡪W**hen you create a file, you don't have to decide permissions up front. Permissions have defaults. Those defaults can be controlled and modified using the umask  command. Typing umask  with no arguments will show you the current umask, in this case 0022 :



🡪What does 0022  mean? That's an octal value that represent the permissions. Another common value is 0002

🡪Use umask -S  to see a human-readable notation



🡪Here's a list that gives a meaning to the number: 0  read, write, execute 1  read and write

2  read and execute

3  read only

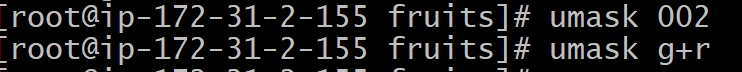
4  write and execute

5  write only

6  execute only

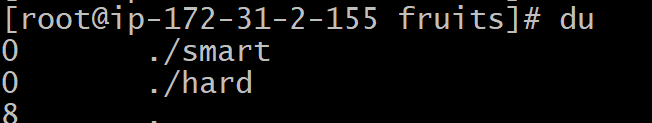
7  no permissions

🡪We can set a new value for the mask setting the value in numeric format: umask 002 or you can change a specific role's permission: umask g+r

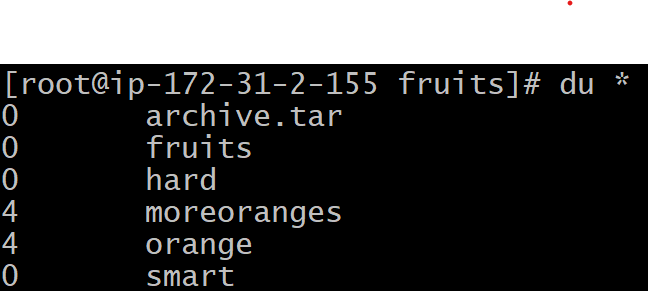


**Du**

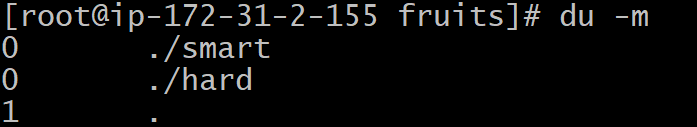
🡪The du  command will calculate the size of a directory as a whole: du



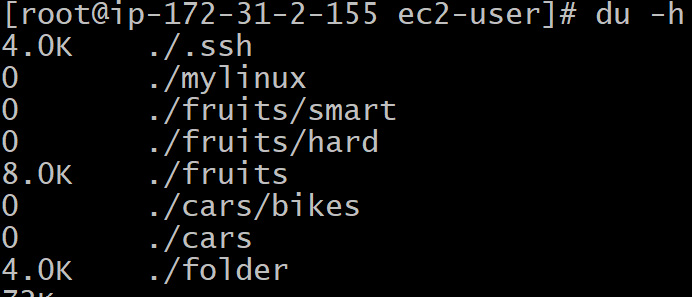
🡪Running du \*  will calculate the size of each file individually.

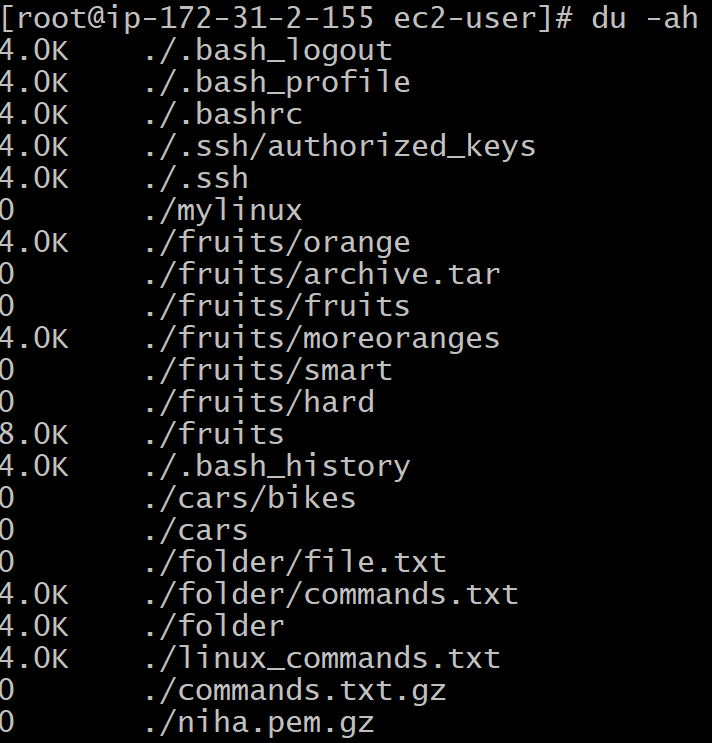


🡪You can set du  to display values in MegaBytes using du -m

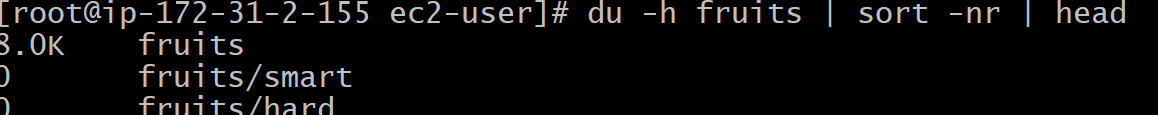


🡪The -h  option will show a human-readable notation for sizes, adapting to the size:



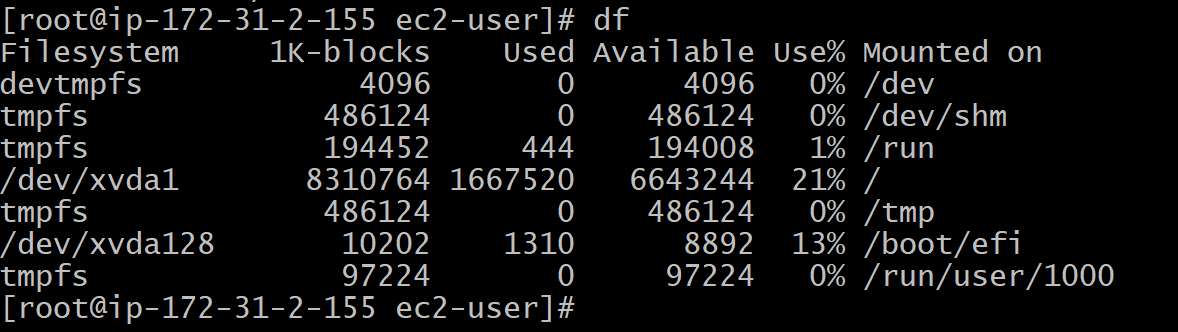
🡪Adding the -a  option will print the size of each file in the directories, too:

🡪A handy thing is to sort the directories by size: du -h | sort -nr and then piping to head  to only get the first 10 results

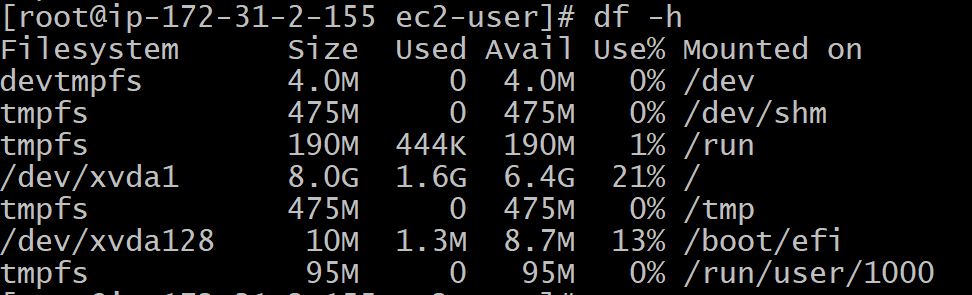


**Df**

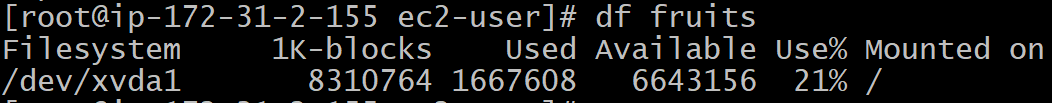
**🡪The df  command is used to get disk usage information. Its basic form will print information about the volumes mounted.**

****

**🡪Using the -h  option ( df -h ) will show those values in a human-readable format**

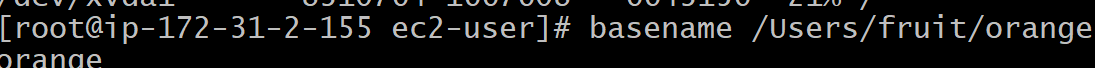


🡪You can also specify a file or directory name to get information about the specific volume it lives on:



**Basename**

**🡪**If you run basename  on a path string that points to a directory, you will get the last segment of the path. In this example, /Users/flavio  is a directory

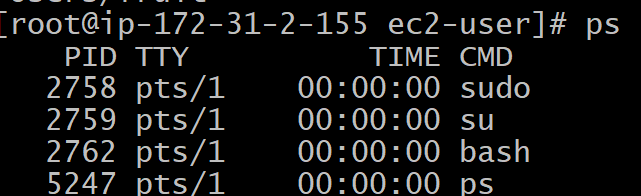
****

**Dirname**

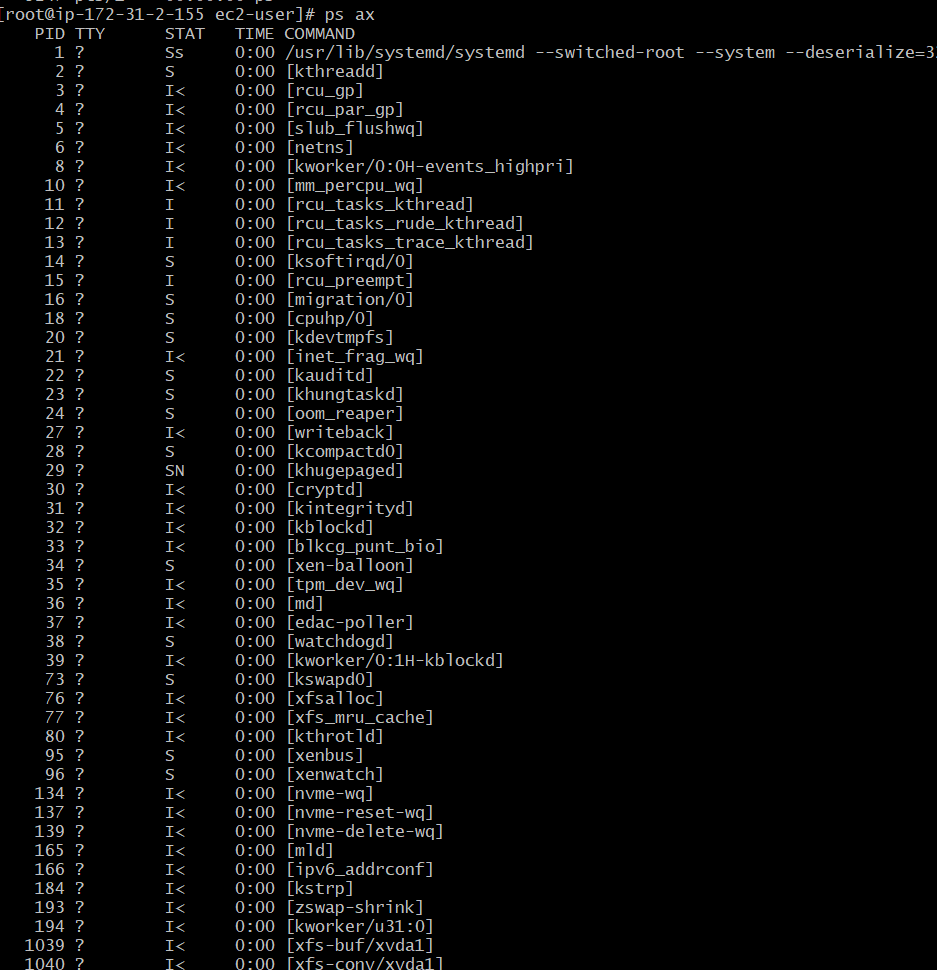
**🡪**Suppose you have a path to a file, for example /Users/flavio/test.txt . Running dirname /Users/flavio/test.txt will return the /Users/flavio  string****

**Ps**

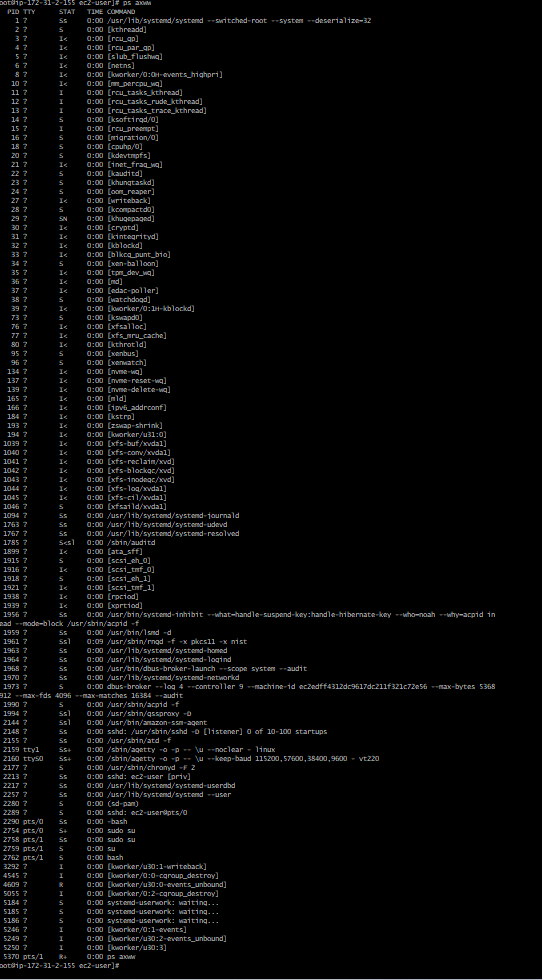
🡪Your computer is running, at all times, tons of different processes. You can inspect them all using the ps  command:



🡪To list all processes we need to pass some options to ps . The most common I use is ps ax



🡪Use the command ps axww  to continue the command listing on a new line instead of cutting it.

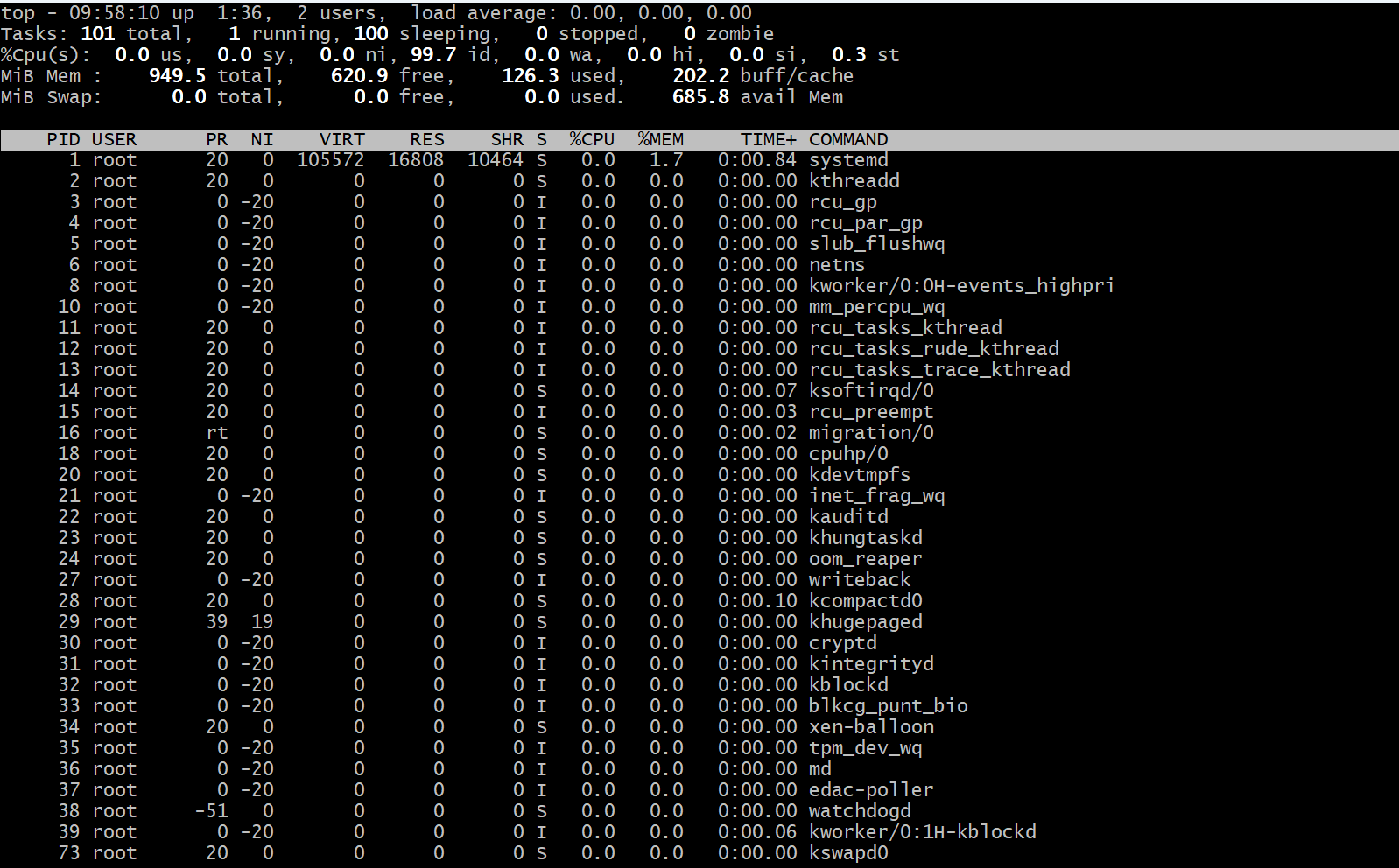


🡪You can search for a specific process combining grep  with a pipe, like this: ps axww | grep "Visual Studio Code"



**Top**

**🡪The top  command is used to display dynamic real time information about running processes in the system.**

****

🡪You can add a flag to sort processes by memory utilized.



**Kill**

**🡪**Linux processes can receive signals and react to them. That's one way we can interact with running programs. The kill  program can send a variety of signals to a program.



**Killall**

Similar to the kill  command, killall  instead of sending a signal to a specific process id will send the signal to multiple processes at once. This is the syntax: killall

**Jobs**

**🡪When we run a command in Linux / macOS, we can set it to run in the background using the &  symbol after the command.**





**Bg**

🡪When a command is running you can suspend it using ctrl-Z . The command will immediately stop, and you get back to the shell terminal.

**fg**

🡪When a command is running in the background, because you started it with &  at the end (example: top &  or because you put it in the background with the fg . bg  command, you can put it to the foreground using Running fg will resume to the foreground the last job that was suspended.

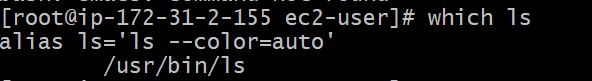
**Type**

🡪A command can be one of those 4 types: an executable a shell built-in program a shell function an alias The type  command can help figure out this, in case we want to know or we're just curious. It will tell you how the command will be interpreted.

🡪One of the most interesting things here is that for aliases it will tell you what is aliasing to. You can see the ll  alias, in the case of Bash and Zsh, but Fish provides it by default, so it will tell you it's a built-in shell function.

**Which**

🡪 Suppose you have a command you can execute, because it's in the shell path, but you want to know where it is located. You can do so using which . The command will return the path to the command specified:



🡪which  will only work for executables stored on disk, not aliases or built-in shell functions.

**Nohup**

**🡪**The way to make a command run even after you log out or close the session to a server is to use the nohup  command. Use nohup   to let the process continue working even after you log out.

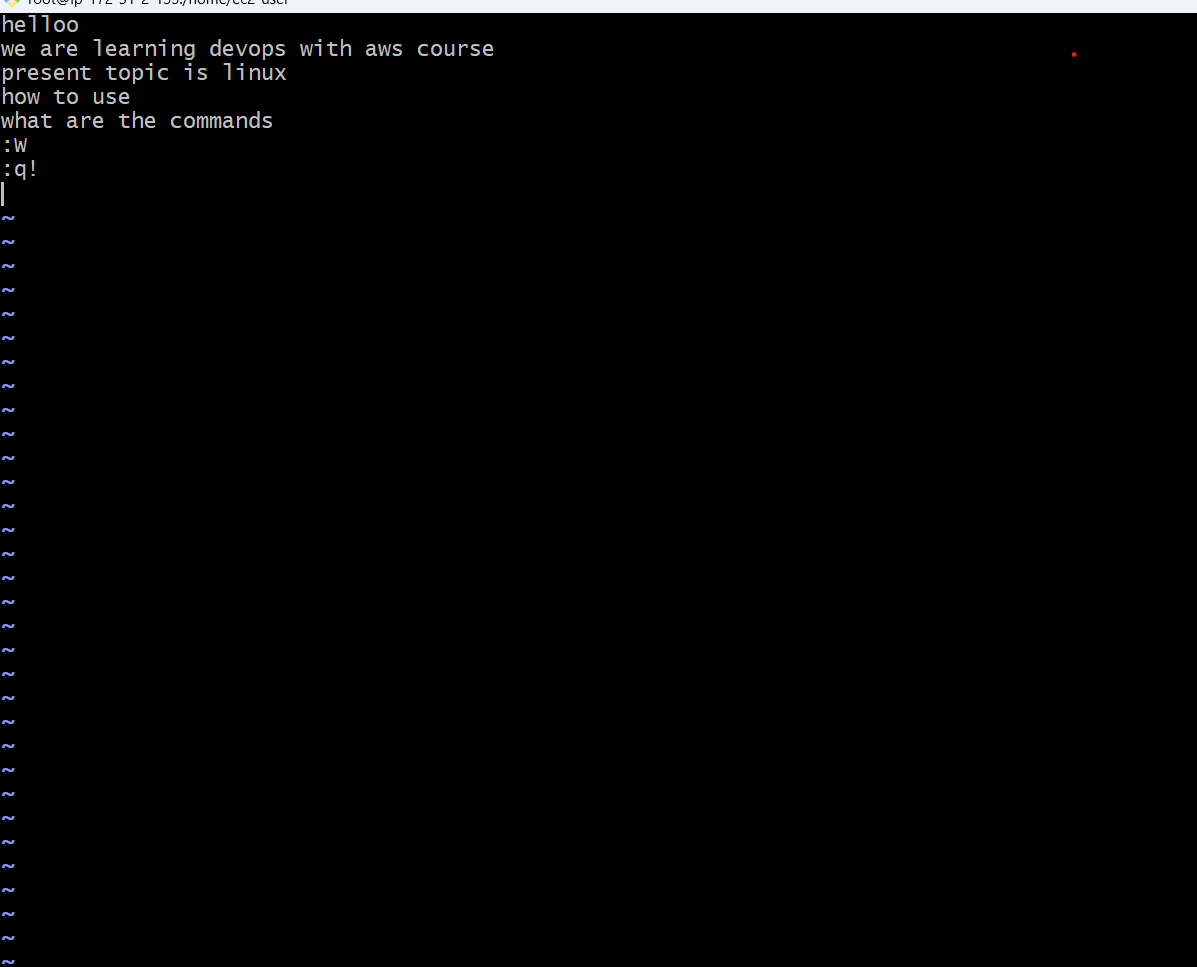
**xargs**

**🡪**The xargs  command is used in a UNIX shell to convert input from standard input into arguments to a command. In other words, through the use of xargs  the output of a command is used as the input of another command. Here's the syntax you will use: command1 | xargs command2.

**🡪**We will channel the output of cat todelete.txt  to the rm  command, through In this way: xargs .

cat todelete.txt | xargs rm

**vim**

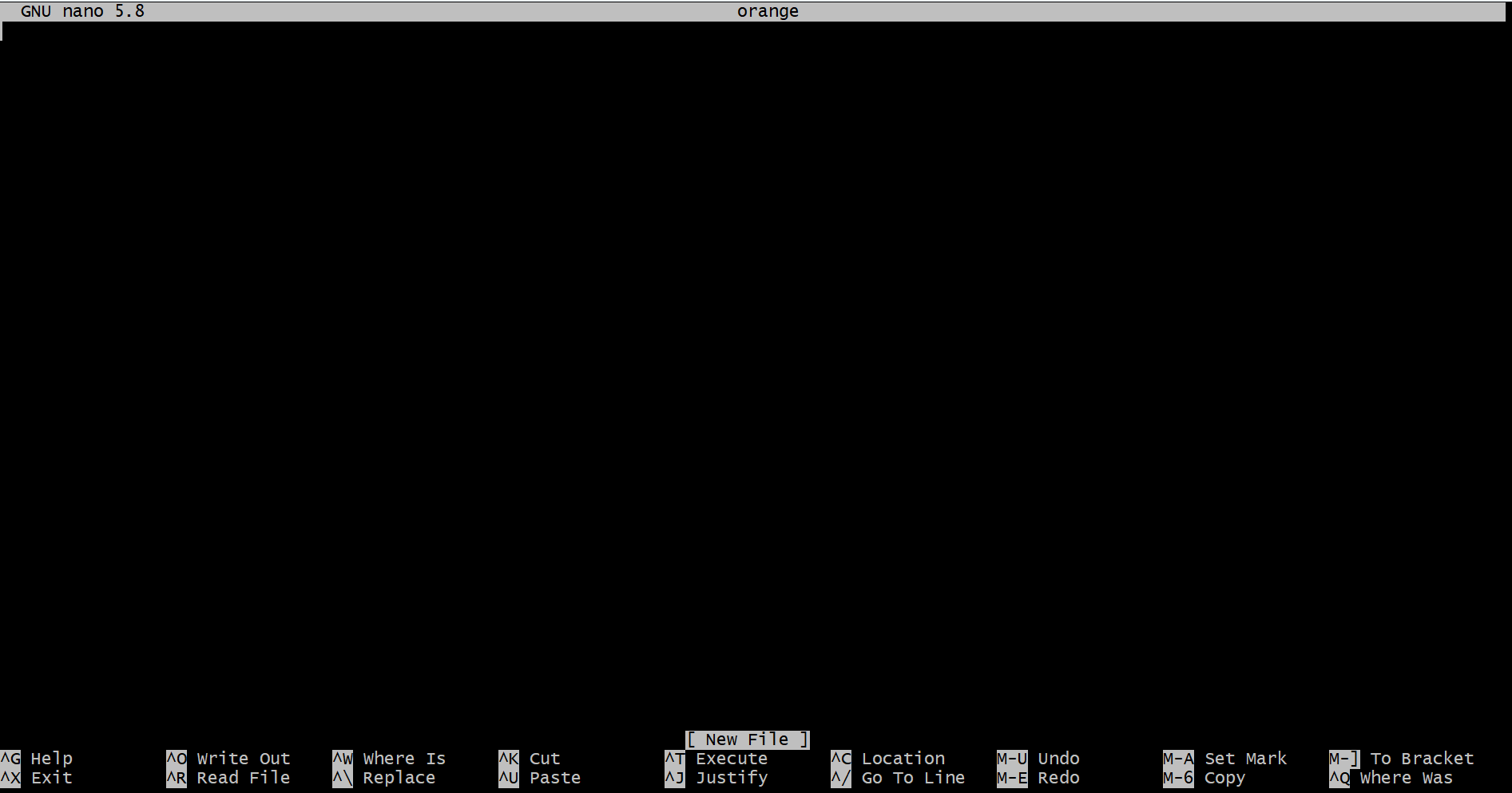
****

**🡪** vim  is a very popular file editor, especially among programmers. It's actively developed and frequently updated, and there's a very big community around it. There's even a Vim conference! vi  in modern systems is just an alias to means vi  i m proved. You start it by running vim , which vi  on the command line**.**

**🡪** When you start the editor, you are in command mode. You can't enter text like you expect from a GUI-based editor. You have to enter insert mode. You can do this by pressing the i  key. Once you do so, the INSERT --  word appear at the bottom of the editor:

🡪 One thing you might want to do now is saving the file. You can do so by pressing w . :  (colon), then 98 You can save and quit pressing :  then w  and q : :wq  You can quit without saving, pressing and ! : :q!  :  then q  You can undo and edit by going to command mode and pressing u . You can redo (cancel an undo) by pressing ctrl-r .

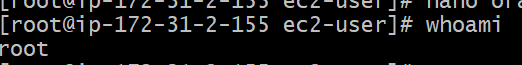
**Nano**

**🡪** ****

🡪 nano  is a beginner friendly editor. Run it using nano  . You can directly type characters into the file without worrying about modes. You can quit without editing using ctrl-X . If you edited the file buffer, the editor will ask you for confirmation and you can save the edits, or discard them. The help at the bottom shows you the keyboard commands that let you work with the file:

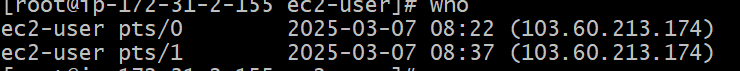
**Whoami**

**🡪** **Type whoami  to print the user name currently logged in to the terminal session:**



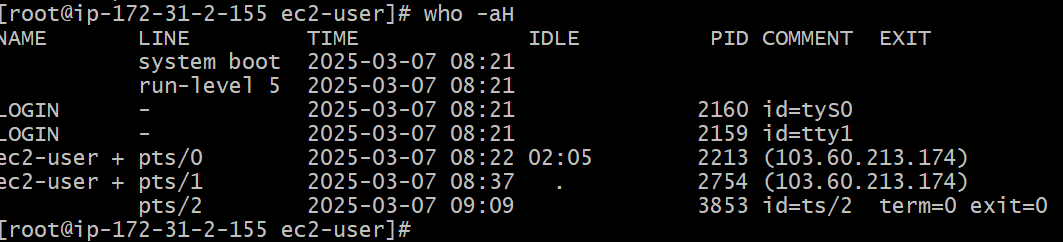
**Who**

The who  command displays the users logged in to the system.



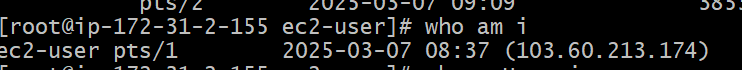
**Who aH**

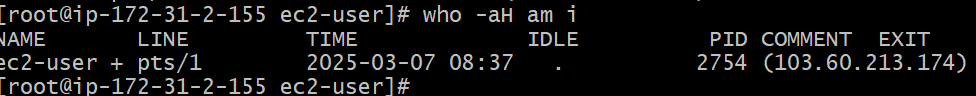
**🡪** **The -aH  flags will tell who  to display more information, including the idle time and the process ID of the terminal:**



**Who am i**

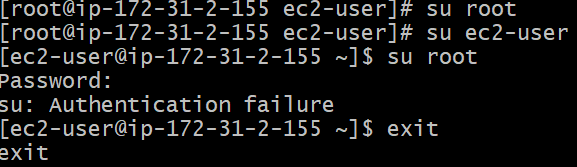
**🡪** **The special who am i  command will list the current terminal session details.**



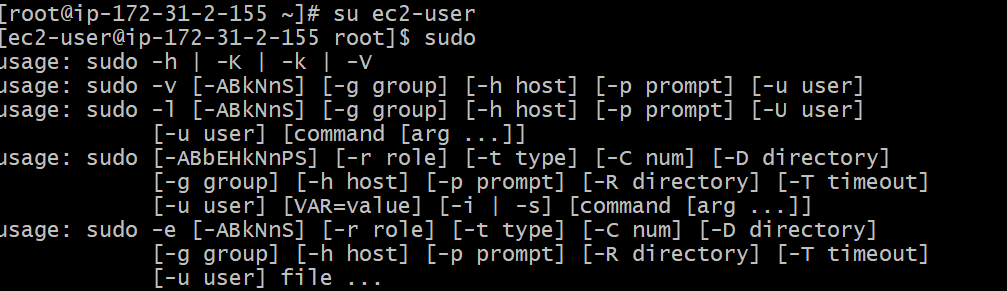


**Su**

🡪 While you're logged in to the terminal shell with one user, you might have the need to switch to another user. For example you're logged in as root to perform some maintenance, but then you want to switch to a user account. You can do so with the su su  command.

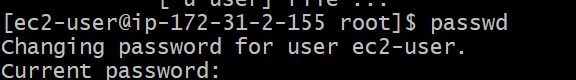


**Sudo**

🡪 sudo  is commonly used to run a command as root. You must be enabled to use sudo , and once you do, you can run commands as root by entering your user's password (not the root user password)

**Passwd**

**🡪** **Users in Linux have a password assigned. You can change the password using the passwd  command. There are two situations here. The first is when you want to change your password. In this case you type: passwd**



Ping

🡪 The ping  command pings a specific network host, on the local network or on the Internet. You use it with the syntax ping   where   could be a domain name, or an IP address. Here's an example pinging google.com

