

Linux: an introduction

A cheat sheet like approach

KU Leuven ICTS 2023-01-26

Contents

Introduction	3
Operating System: Linux	4
Kernel	5
Shell	5
Getting started	6
Linux command syntax	6
Getting help	7
Useful features	8
Tab autocompletion	8
Command history	8
Some shortcuts	8
Various commands	9
The File system	10
ls command	12
Navigating the file system	14
Working with files and directories	15
Naming tips	15
Creating files	15
Nano	15
File creating commands	17
Viewing file content	17
Viewing commands	17
Less command	18
Moving, removing files and directories	18
cp	18
mv	18
rm	19

More file handling.....	20
Finding files	20
Archiving and compressing files.....	20
Links	22
File permissions.....	23
The shell revisited	25
IO Redirection	25
Pipes.....	26
Command chaining	26
Command substitution	27
(very) Basic Bash scripting.....	27
Basic steps.....	27
Bash variables	27
Extra processing commands	29
sort.....	29
tr.....	29
wc.....	30
uniq	30
comm	31
split.....	31
cat.....	32
cut	32
paste.....	33
wget	33
curl	33
grep	34
Using regular expressions	34
Monitoring resources.....	37
Check disk space	37
Process foreground / background	37
Process Management	38
Useful links.....	40

Introduction

Fundamentally, there are two different ways to work with an operating system:

- through a graphical user interface (GUI), in which the user uses a pointing device to manipulate windows, a menu-based approach
- through the command line interface (CLI), in which the user types commands at a prompt.

The command line (or the terminal) has been around for ages. In the early days it was the only way to operate a computer. With the rise of graphical interfaces the widespread use of computers really took off.

Many of the graphical programs you use on a daily basis are actually nothing more than a graphical shell for a command line tool. This is certainly the case for many tools used in the scientific community. Most of the supercomputers run on Linux and more often than not all you get to interact with those is a command line.

Some warning on the Linux command line interface:

- There is no Recycle Bin, anything that you execute (run) in the command prompt would need to be fixed manually. Double Check EVERY command that you type, to make sure it is correct.
- The command line interface is case-sensitive, one wrongly typed character can cause a lot of problems.
- The command line is a very powerful tool, and it requires a little more finesse and learning than the GUI (Graphic User Interface).

Operating System: Linux

"An operating system (OS) is software system that manages computer hardware, software resources, and provides common services for computer programs". (Wikipedia)

Operating systems (OS) were created to simplify the use of computers: any given program can focus on its core features and leave all basic system functionalities to the OS.

Linux is an open source operating system. It will manage basic tasks such as file management, memory management, process management, input-output management, and controlling peripheral devices: makes the connections between the software and the physical resources that do the work.

Every Linux-based OS (or distribution) involves the Linux kernel—which manages hardware resources—and a set of software packages that make up the rest of the operating system. The OS includes some common core components, like the GNU tools. These tools give the user a way to manage the resources provided by the kernel, install additional software, configure performance and security settings, and more. All of these tools bundled together make up the functional operating system. Because Linux is an open source OS, combinations of software can vary between Linux distributions.

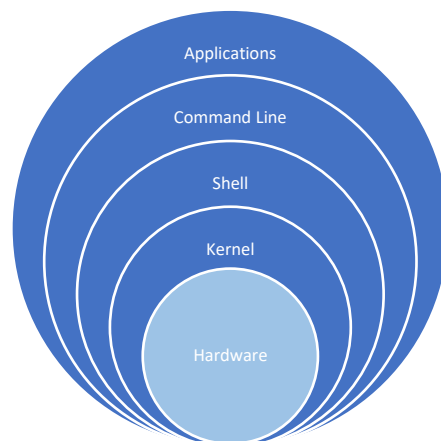


Figure 1 components in the nix architecture

Kernel

Check:

- <https://www.kernel.org/>
- <https://www.redhat.com/en/topics/Linux/what-is-the-Linux-kernel>

The kernel takes user input via the shell and accesses the hardware to perform things like:

- Memory management. Physical memory is extended virtually. Unused programs or program sections are offloaded to the hard disk and loaded into RAM when required for execution.
- File management. Linux has a hierarchical file system whose internal structure may vary. Disk space from other systems can be mounted.
- Device management. Access to devices is through files that form the interface between the device drivers and the kernel.
- Program and process management. Linux ensures that each program runs independently.
- Data access. Data stored in the file system must be protected from unauthorized access.

Shell

The shell is the most basic link between the user and the operating system. The commands entered on the command line will be interpreted by the shell and passed on to the operating system. Once the work is done, the command line will return to a prompt and await further input.

There are several different shells, and syntax might vary between them. All shells support similar basic functions.

- **bash** This is the standard Linux shell (Bourne Again Shell).
- **sh** The original Bourne shell.
- **csh** The C shell uses a different programming interface to bash.
- **ksh** The Korn shell is one of the most popular shells on Unix. It is compatible with bash .
- **tcsh** The enhanced C shell.
- **zsh** A bash compatible shell.

Getting started

Linux command syntax

Telling the computer what to do is done by passing commands to the computer at the prompt. The prompt: usually \$

A Linux command consists of 3 parts: the command itself, the command options, and its arguments. The anatomy of a command line interface is as follows:

```
command [OPTIONS prefixed with - or --] [ARG1] [...ARGX]
```

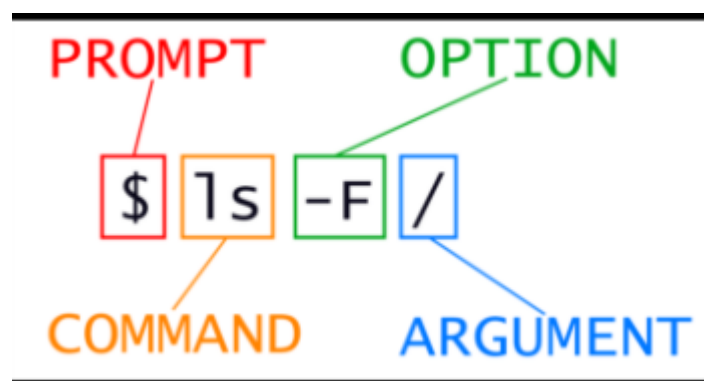


Figure 2. General syntax (source: <https://swcarpentry.github.io/shell-novice/02-filedir/index.html>)

Under the hood, a command can be either: (<https://datascienceatthecommandline.com/2e/chapter-2-getting-started.html>)

- a shell built-in, are command-line tools provided by the shell.
- a binary executable, is created by compiling source code to machine code.
- an interpreted script. An interpreted script is a text file that is executed by a binary executable. i.e. Python script, bash script.

Getting help

Note: The built-in help is complete, but lacks examples.

Command	Useful options	Common usage	What does it?
run the command with the -h or --help switch	-h --help	anycommand -h anycommand --help	Show usage information and a list of options to be used with the command
<TAB> completion			When in doubt about a specific command's name, an option, or a file name, use tab completion to help.
man		man anycommand	Loads manual page for the anycommand command The traditional documentation system
help		help anycommand	Shows help on the bash built-in commands
info		info anycommand	Another documentation system, behaving more like a digital book
apropos		apropos keyword	searches for man pages containing the keyword similar to <code>man -k</code>
whatis		whatis anycommand	a one-line summary of a command, taken from its man page.
which		which anycommand	locate the executable file associated with the given command
type	-a	type anycommand type -a anycommand	It shows how the command would be interpreted when entered on the command line.
	-t	type -t anycommand	print a single word describing the type of the command which can be one of the following: <ul style="list-style-type: none"> • alias (ex. <code>ls</code>) • function (ex. <code>conda</code>) • builtin (ex. <code>echo</code>) • file (ex. <code>cut</code>) • keyword (ex. <code>for</code>)

Table 1. Getting help (<https://www.howtogeek.com/108890/how-to-get-help-with-a-command-from-the-Linux-terminal-8-tricks-for-beginners-pros-alike/>)

Useful features

Tab autocompletion

hit Tab while typing a command, option, or file name and the shell environment will automatically complete what you're typing or suggest options.

Command history

The `history` command displays a user's command line history. Arrow UP and Arrow DOWN keys walk through the history

Command	Options	Common usage	What does it?
history			See your previously entered commands, the commands are saved to a file in your home directory (<code>.bash_history</code>) Use the up/down arrows to scroll through your previous commands
		<code>history tail</code>	View last commands in the history list
		<code>history grep keyword</code>	Search for a keyword in the commands
	number	<code>history num</code>	list the last num entries from the history

Table 2. *history* command

Check the maximum number of lines kept in history: `echo $HISTSIZE`

Check the file containing the history: `echo $HISTFILE`

Some shortcuts

Command	Options	Common usage	What does it?
CTRL-C			Stop current command
CTRL-R			Recall the last command matching the characters you provide
!!			Repeat last command
!abc			Run last command starting with <i>abc</i>
!number			Retrieve command on line <i>number</i> in the history

Table 3. *Some shortcuts* (<https://cheatography.com/davechild/cheat-sheets/Linux-command-line/>)

Various commands

Command	Options	Common usage	What does it?
uname	-a		Show system and kernel
date			Show system date
uptime			Show uptime
whoami			Show your username
clear			Clear terminal
exit			Exit the current shell
logout			This will end the terminal session and return to the login screen. Some systems may have a file called .logout or .bash_logout in the user's home directory that executes when logging out


Table 4. Various commands (<https://cheatography.com/davechild/cheat-sheets/Linux-command-line/>)

The File system

“In Linux, everything is a File”. It is a generalization concept, in Unix and its derivatives such as Linux, everything is considered as a file. If something is not a file, then it must be running as a process on the system.

Linux has a hierarchical directory structure: files are organized in a tree-like pattern of directories (folders), which may contain files and other directories, etc.

Common Linux directories at the root level.



```
bin -> usr/bin
boot
dev
etc
home
init
lib -> usr/lib
lib32 -> usr/lib32
lib64 -> usr/lib64
libx32 -> usr/libx32
lost+found
media
mnt
opt
proc
root
run
sbin -> usr/sbin
snap
srv
sys
tmp
usr
var
```

Figure 3. Linux basic directory structure

directory	
/	The root directory contains all subdirectories.
/boot	When the system boots, the boot program will examine the /boot. Among the objects it looks for is the map file, by which the Linux boot manager will determine the location of the kernel on the hard disk.
/bin	The binaries directory contains the most important commands.
/dev	The device directory contains device files through which you communicate with the devices connected to the computer.
/etc	Only the configuration files should be there: passwd, group, hosts, etc.
/home	The users home directory will often be under the /home/yourname directory. The advantage of this is that the user will have their own file system.
/lib	Directory for shared libraries.
/opt	Directory for additional programs that are not part of Linux.
/root	The administrator's directory. Not to be confused with the root /.
/sbin	This is the directory for administrative commands.
/tmp	The temporary files directory. It is accessible for read and write.
/var	This is the directory for variable data in which Linux stores variable, rapidly or frequently changing data.
/usr	The sensitive data directory is a directory that contains a series of directories in which where Linux keeps very important data. Note: /usr/doc contains the Linux documentation

Table 5. Linux basic directory description

Command	Options	Common usage	What does it?
file		file filename	Show what kind of file
tree			List contents of directories in a tree-like format.
	-L	tree -L 1	List only the first level
pwd		pwd	Print working (current) directory

Table 6. Useful commands on files

ls command

lists the contents of the directory, it will stick to the current directory.

Command	Options	Common usage	What does it?
ls	-a		Show all (including hidden)
ls	-R		Recursive list
ls	-r		Reverse order
ls	-t		Sort by last modified
ls	-S		Sort by file size
ls	-l		Long listing format
ls	-1		One file per line
ls	-m		Comma-separated output
ls	-Q		Quoted output
ls	-F		Visually display a file's type '*' executable '/' directory '@' link

Table 7. ls command (<https://cheatography.com/davechild/cheat-sheets/Linux-command-line/>)

Tip:

- options can be combined
ls -ltr: long listing, most recent files at the end
- use wildcards

Wildcard	Common usage	What does it ?
?	hel?	any <i>single</i> character (a-z, 0-9) .
*	hel*	any number of characters (zero or more characters).
[]	k[a,e,x]m k[f-h]m	specifies a range. This is an OR operation
[!]	list[!9]	similar to the [] construct, except rather than matching any characters inside the brackets, it'll match any character, as long as it is not listed between the [and]. This is a logical NOT.
{ }	ls {*.doc, *.pdf}	list anything ending with .doc or .pdf. terms are separated by commas and each term must be the name of something or a wildcard. No spaces are not allowed after the commas.

Table 8. globbing - wildcards (<https://tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm>)

Globbering : the shell will first attempt to expand the wildcard pattern to match any files that are present before passing their expanded names to the program we want to run. Globbing is an operation that is performed by the shell.

What do the fields in ls -al output mean?

```
frankvp@CRD-L-08004:~/LinuxDev$ ls -al
total 3712
drwxr-xr-x 13 frankvp frankvp 4096 Feb  8  2021 .
drwxr-xr-x 13 frankvp frankvp 4096 Sep 14 17:24 ..
-rwxr-xr-x  1 frankvp frankvp 260948 Feb  6  2021 Linux_basic_for_iridis_v1.pdf
-rwxr-xr-x  1 frankvp frankvp 62341 Feb  6  2021 MOCK_DATA.csv
-rwxr-xr-x  1 frankvp frankvp 62341 Feb  6  2021 MOCK_DATA.r.txt
-rwxr-xr-x  1 frankvp frankvp  52 Feb  6  2021 cat_text
-rwxr-xr-x  1 frankvp frankvp 1341 Feb  6  2021 diff-out
drwxr-xr-x  3 frankvp frankvp 4096 Feb  8  2021 dir100
drwxr-xr-x  2 frankvp frankvp 4096 Feb  8  2021 dir1000
drwxr-xr-x  6 frankvp frankvp 4096 Feb  6  2021 dir101
drwxr-xr-x  2 frankvp frankvp 4096 Feb  6  2021 dir102
drwxr-xr-x  2 frankvp frankvp 4096 Feb  6  2021 dir3
-rwxr-xr-x  1 frankvp frankvp  20 Feb  6  2021 file1
-rwxr-xr-x  1 frankvp frankvp  20 Feb  6  2021 file1_link
-rwxr-xr-x  1 frankvp frankvp  0 Feb  6  2021 filenow
-rwxr-xr-x  1 frankvp frankvp 678848 Feb  6  2021 hands-on-linux_intro-all.pdf
-rwxr-xr-x  1 frankvp frankvp  32 Feb  6  2021 hello_world_1.sh
-rwxr-xr-x  1 frankvp frankvp 11594 Feb  6  2021 index.html
-rw-r--r--  1 frankvp frankvp 3204 Feb  8  2021 less.txt
-rw-r--r--  1 frankvp frankvp 3204 Feb  8  2021 less1
lrwxrwxrwx  1 frankvp frankvp  11 Feb  6  2021 link_testdir -> testdir.txt
-rw-r--r--  1 frankvp frankvp 3880 Feb  8  2021 listfile
-rwxr-xr-x  1 frankvp frankvp 114 Feb  6  2021 lnklLorem117
-rwxr-xr-x  1 frankvp frankvp 114 Feb  6  2021 lnklLorem15
-rwxr-xr-x  1 frankvp frankvp 115 Feb  6  2021 Lorem1
-rwxr-xr-x  1 frankvp frankvp 114 Feb  6  2021 Lorem_15.txt
-rwxr-xr-x  1 frankvp frankvp 469 Feb  6  2021 Lorem_2
-rwxr-xr-x  1 frankvp frankvp 469 Feb  6  2021 Lorem_2_sorted
-rwxr-xr-x  1 frankvp frankvp  0 Feb  6  2021 Lorem_empty
```

Figure 4. ls -al

Column	
1	File type + Permissions <i>File type :</i> https://Linuxconfig.org/identifying-file-types-in-Linux – : regular file d: directory c: character device file b: block device file s: local socket file p: named pipe l: symbolic link <i>Permissions:</i> User Group Other w: read w: write x: execute
2	Number of hard links
3	Owner name
4	Group name
5	Size in bytes
6	Last modification date and time
7	File/directory name

Table 9. Fields from ls -al

Hidden files: files starting with a . (dot), these are usually files and directories that are used to configure different programs on your computer. The prefix . is used to prevent these configuration files from cluttering the terminal when a standard ls command is used.

Navigating the file system

A **directory**: a directory is just a file containing names of other files.

A **path** is a sequence of nested directories with a file or directory at the end, separated by the / character

Relative path: documents/fun/file1

Relative to the current directory

Absolute path: /home/user/leuven/file2

Start at the root.

/ : root directory.

Notice that there are two meanings for the / character. When it appears at the front of a file or directory name, it refers to the root directory. When it appears inside a path, it is a separator.

Command	Options	Common usage	What does it?
pwd		pwd	Print working (current) directory
cd		cd dir	Change directory Changes the current position to the specified directory
		cd ..	Go up one level (parent)
		cd dir1/dir2	Relative pathname, relative to the current directory
		cd /dir1/dir2	Absolute path, starting from the root
		cd -	Switch between current dir and previous dir

Table 10. directories navigation commands

Symbol	What does it mean?
.	Current directory
..	Parent directory
~	Home directory
/	Root directory

Table 11. special symbols related to directories

Working with files and directories

Naming tips

Source: <https://swcarpentry.github.io/shell-novice/03-create/index.html>

Some useful tips for the names of your files and directories.

1. Do not use spaces.
Spaces can make a name more meaningful, but since spaces are used to separate arguments on the command line it is better to avoid them in names of files and directories. You can use - or _ instead
2. Do not begin the name with - (dash).
Commands treat names starting with - as options.
3. Stick with letters, numbers, . (dot), - (dash) and _ (underscore).
Many other characters have special meanings on the command line.

File extension or not: this is just a convention, in Linux file extensions are not necessary. Files contain bytes: it's up to us and our programs to interpret those bytes according to the rules for plain text files, PDF documents, configuration files, images, and so on. However, two-part names are used to help keep different kinds of files apart.

Creating files

Use an editor to create a file.

Nano

Nano is very much like Windows 'Notepad' but without mouse support. All commands are executed through the use of the keyboard, using the Ctrl-key.

It can only work with plain character data, not tables, images, or any other human-friendly media. It is one of the least complex text editors. On Unix systems (such as Linux and macOS), many programmers use Emacs or Vim (both of which require more time to learn), or a graphical editor such as Gedit.



Figure 5. Nano editor

- **Ctrl+X** Exit the editor. If you've edited text without saving, you'll be prompted as to whether you really want to exit.
- **Ctrl+O** Write (output) the current contents of the text buffer to a file. A filename prompt will appear; press Ctrl+T to open the file navigator shown above.
- **Ctrl+R** Read a text file into the current editing session. At the filename prompt, hit Ctrl+T for the file navigator.
- **Ctrl+K** Cut a line into the clipboard. You can press this repeatedly to copy multiple lines, which are then stored as one chunk.
- **Ctrl+J** Justify (fill out) a paragraph of text. By default, this reflows text to match the width of the editing window.
- **Ctrl+U** Uncut text, or rather, paste it from the clipboard. Note that after a Justify operation, this turns into unjustify.
- **Ctrl+T** Check spelling.
- **Ctrl+W** Find a word or phrase. At the prompt, use the cursor keys to go through previous search terms, or hit Ctrl+R to move into replace mode. Alternatively you can hit Ctrl+T to go to a specific line.
- **Ctrl+C** Show current line number and file information.
- **Ctrl+G** Get help; this provides information on navigating through files and common keyboard commands.

Manual: <https://www.nano-editor.org/dist/v4/nano.pdf>

Cheat sheet: <https://www.nano-editor.org/dist/latest/cheatsheet.html>

File creating commands

Command	Options	Common usage	What does it?
touch		<code>touch anyfile</code>	Create empty file anyfile
>		<code>>anyfile</code>	Use redirection to create an empty file anyfile
cat >		<code>cat > anyfile</code>	run the <code>cat</code> command followed by the redirection operator <code>></code> and the name of the file you want to create. Press Enter type the text and once you are done press the CTRL+D to save the file.

Table 12. File generating commands

Viewing file content

Viewing commands

Basic commands

Command	Options	Common usage	What does it?
cat		<code>cat anyfile anotherfile</code>	Concatenate files and output
head	<code>-n</code>	<code>head -n nn anyfile</code>	Show first nn lines of anyfile
	<code>-n</code>	<code>head -n -nn anyfile</code>	Remove trailing lines, do not show last nn lines
tail	<code>-n</code>	<code>tail -n nn anyfile</code>	Show last nn lines of anyfile
	<code>-n</code>	<code>tail -n +nn anyfile</code>	Remove the header lines, show from line nn on
less		<code>view anyfile</code>	View a file

Table 13. Viewing commands

Less command

View and paginate any file

Action in less	What does it?
Down arrow, enter, e, j	Move forward one line.
Up arrow, y, k	Move backward one line.
Space bar, f	Move Forward one page.
b	Move Backward one page.
/pattern	Search forward for matching patterns.
?pattern	Search backward for matching patterns.
n	Repeat previous search.
N	Repeat previous search in reverse direction.
g	Go to the first line in the file.
G	Go to the last line in the file.
q	Exit less

Table 14. less command (<https://Linuxize.com/post/less-command-in-Linux/>)

Moving, removing files and directories

cp

cp is used to copy files and directories. It requires at least two arguments.

- To copy file file1 to a new file file2
`cp file1 file2`
 file2 will have the same contents as file1, but it will have new date stamp. Modifying one of the files will not affect the other file.
- To copy file1 and file2 to a different directory
`cp file1 file2 <path_to_new_dir>`
 File <path_to_new_dir>/file1 will have the same contents as file1, and file <path_to_new_dir>/file2 will have the same contents as file2, but they will have new date stamps.
- To copy directory dir1 with all the files and subdirectories to a different directory
`cp -r dir1 <path_to_new_dir>`

mv

mv is used to move or rename files and directories. It requires at least two arguments.

- To rename file file1 to file2
`mv file1 file2`
 Note that after being renamed, the file will still have old time stamp.
- To move file1 and file2 to a different directory
`mv file1 file2 <path_to_new_dir>`
- To move directory dir1 with all the files and subdirectories to a different directory
`mv dir1 <path_to_new_dir>`

rm

rm is used to remove files and directories. It requires at least one argument.

Be careful with this command so that you don't delete files that you meant to keep.

To remove directory dir1 with all the files and subdirectories: `rm -r dir1`

Tips:

- Whenever you use wildcards with rm (besides carefully checking your typing!), test the wildcard first with `ls`. This will let you see the files that will be deleted.
- Use `-i` flag, to work interactively and confirmation will be asked for.
- Use `-v` flag to show what is happening.

Command	Options	Common usage	What does it?
cp		<code>cp anyfile anotherfile</code>	Copy anyfile to anotherfile, overwriting if the file already exists or multiple sources to directory
	<code>-i</code>	<code>cp -i anyfile anotherfile</code>	Interactive option, prompts the user before overwriting the destination file
	<code>-v</code>		Verbose, displays what is being copied
	<code>-r</code>		
mv		<code>mv old new</code>	Move or rename files and directories
rm		<code>rm anyfile</code>	Delete anyfile

Table 15. file manipulation commands

Command	Options	Common usage	What does it?
mkdir		<code>mkdir dir</code>	Make a directory dir. Relative and absolute path specification possible.
	<code>-p</code>	<code>mkdir -p dir/sdir1/sdir2</code>	Create sub-directories of a directory. It will create parent directory first, if it doesn't exist.
rmdir		<code>rmdir dir</code>	Remove a directory dir, this directory must be empty.

Table 16. directories creational commands

More file handling

Finding files

Use `find` to find files and directories whose names match simple patterns.

Command	Options	Common usage	What does it?
find	-name	<code>find dir -name name*</code>	Find files starting with <i>name</i> in <i>dir</i>
	-user	<code>find dir -user name</code>	Find files owned by <i>name</i> in <i>dir</i>
	-size	<code>find dir -size + 10M</code>	Find all files larger than 10M
	-mmin	<code>find dir -mmin num</code>	Find files modified less than <i>num</i> minutes ago in <i>dir</i>
	-type	<code>find dir -type d -name 'pattern'</code>	Find all directories whose name is <i>name</i> in <i>dir</i> . Use single quotes for the pattern
		<code>find dir -type f -name 'pattern'</code>	Find all files whose name is <i>name</i> in <i>dir</i> . Use single quotes for the pattern
	-perm	<code>find dir -type f -perm 0777</code>	Find all files in <i>dir</i> with the permission 777
whereis		<code>whereis command</code>	Find binary / source / manual for <i>command</i>
locate		<code>locate file</code>	Find <i>file</i> (quick search of system index)
file		<code>file anyfile</code>	Get type of anyfile
stat		<code>stat anyfile</code>	Stat command gives information such as the size of the file, access permissions and the user ID and group ID, birth time access time of the file.
	f	<code>stat -f anyfile</code>	file system information

Table 17. finding files

Archiving and compressing files

Tar stands for tape archive, the archive itself is a single file that can represent many files. tar files do not have to be compressed but they often are, even a not zipped tar file will often use less disk space than the files stored individually.

Syntax:

```
tar [function] [options] [pathname ...]
```

Command	Function	Common usage	What does it?
tar	-c		create a new archive.
	-r	tar -rf my_archive.tar example.txt	Files or directories can be added/updated to the existing archives
	-t		list the contents of the archive.
	-u		append files, but only those that are newer than the copy in the archive.
	-x		extract an archive. The files will expand within the current directory.
	--delete	tar --delete my_archive.tar workshop/file1	To remove files/directories from an existing tar file. Make sure the path of the file/directory to delete is the same as the pathname displayed by tar -tvf

Table 18. tar command functions

Command	Options	Common usage	What does it?
tar	-v		verbose
	-f		specifies the file name and must be followed by this name. Use the extension .tar to clearly indicate the file type.
	-k		do not overwrite existing files when extracting files from an archive, and return an error if such files exist.
	-z		use gzip for compression and gunzip for decompressing the file.
	-j		uses bzip2 for compression and bunzip2 for decompressing.

Table 19. tar command options

Recipes:

Explicitly archive and compress afterwards

- Put all the files to archive in the same folder.
- Create the tar file:
 - tar -cvf my_archive.tar workshop/
archive the workshop directory within the current directory
 - c : creates a .tar archive
 - v : tells what is happening (verbose)
 - f : assembles the archive into my_archive.tar
- Once the tar file created, verify the file contents with the -t option

- a. `tar -tf my_archive.tar`
- 4. Compress the tar, create gzip file (most current)
 - a. `gzip my_archive.tar` (will add .gz extension)
 - b. to decompress:
 - `gunzip my_archive.tar.gz`

Archive and compress data the fast way

- 1. Compress with gzip:
 - a. `tar -zcvf my_archive.tar.gz workshop/`
 - b. decompress:
 - i. `tar -xzvf my_archive.tar.gz -C /path/to/directory`
- 2. Compress with bzip2:
 - a. `tar -jcvf my_archive.tar.gz workshop/`
 - b. decompress:
 - i. `tar -xjvf archive.tar.bz2 -C /path/to/directory`

Extract a file from a tar file

- 1. Check with -t option the path of the file
- 2. `tar -xf my_archive.tar workshop/file1`

Add a file to an archive

- 1. Add the file documents/work/file1 to the archive archive.tar only if it's newer than the version already in the archive (or does not yet exist in the archive).
- 2. `tar -uvf archive.tar documents/work/file1`

tip:

the content of a tar file may be read directly with `less`, without extracting the file.

Links

A symbolic or soft link is an actual link to the original file (similar to a shortcut in Windows), whereas a hard link is a mirror copy of the original file.

If you delete the original file, the soft link will be useless since it points to a non-existent file, the hard link will still have the data of the original file because it acts as a mirror copy of the original file.

Command	Options	Common usage	What does it?
ln	-s	<code>ln -s source_file link_2_file</code>	Create a soft (symbolic) link
		<code>ln source_file link_2_file</code>	Create a hard link

Table 20. In command

Tip:

what is the difference between Hard link and the normal copied file?

- If you copy a file, it will just duplicate the content. So if you modify the content of a one file (either original or hard link), it has no effect on the other one.
- However if you create a hard link to a file and change the content of either of the files, the change will be seen on both.

Source: <https://ostechnix.com/explaining-soft-link-and-hard-link-in-Linux-with-examples/>

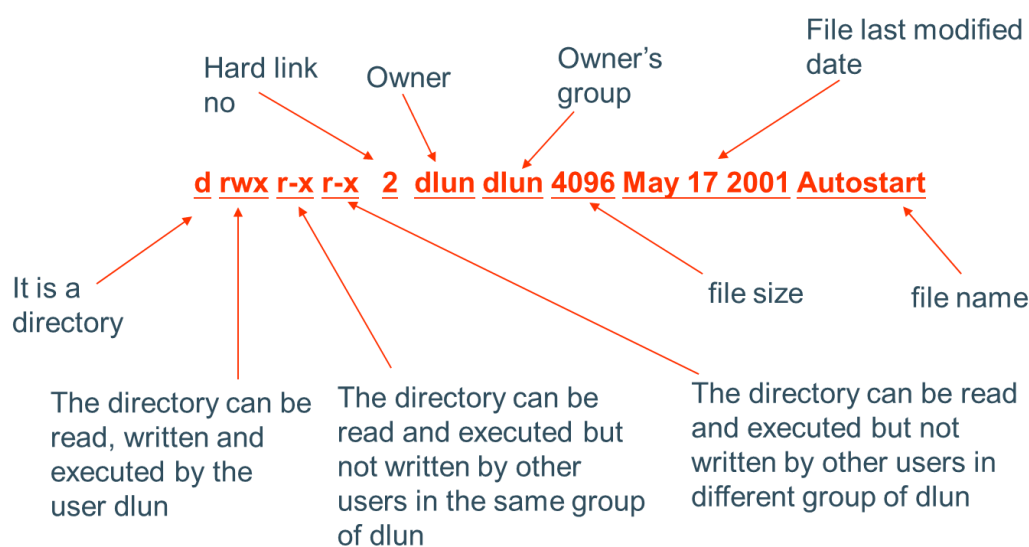
File permissions

Linux File Access permissions:

- Linux is a multiuser system, the files of all users are stored in a single file structure
- Mechanism is required to restrict one user to access the files of another user. User can impose access permission to each file to restrict its access

3 types of access rights

- Read access (r)
 - reading, opening, viewing, and copying the file is allowed
- Write access (w)
 - writing, changing, deleting, and saving the file is allowed
- Execute rights (x)
 - executing and invoking the file is allowed. This is required for directories to allow searching and access.



Command	Options	Common usage	What does it?
chmod		chmod 775 file	Change mode of <i>file</i> to 775
	-R	chmod -R 600 folder	Recursively chmod folder to 600
		chmod go+r	add read permissions to group and others.
		chmod u-w	remove write permissions from user.
		chmod a-x	remove execute permission from all (a: all).
chown		chown user:group file	Change file owner to user and group to group
		chown user file	Change file owner to user
		chown :group file	Change file group to group

Table 21. chmod chown

2 formats for permissions:

- octal format (3 digit octal form)

Calculate permission digits by adding numbers below.	
4	read (r)
2	write (w)
1	execute (x)

Table 22 permission digits

- symbolic format
 - action: r (read), w (write), x (execute)
 - u (user), g (group), o (other), a (all)

The shell revisited

IO Redirection

`stdin`, `stdout`, and `stderr` are three data streams.

- `stdin(0)`: standard input stream. This accepts text as its input (keyboard)
- `stdout(1)`: text output from the command to the shell is delivered via the `stdout` (standard out) stream (the screen)
- `stderr(2)`: Error messages from the command are sent through the `stderr` (standard error) stream (error messages output to the screen)

The `>` redirection symbol works with `stdout` by default. You can use one of the numeric file descriptors to indicate which standard output stream you wish to redirect.

Command	Options	Common usage	What does it?
<code>></code>		<code>cmd > file</code>	redirect standard output of <i>cmd</i> to <i>file</i> . Existing content is overwritten
		<code>cmd > /dev/null</code>	discard stdout of <i>cmd</i>
<code>>></code>		<code>cmd >> file</code>	append stdout to <i>file</i>
<code><</code>		<code>cmd < file</code>	Input of <i>cmd</i> from <i>file</i>
<code>2></code>		<code>cmd 2> file</code>	Error output (<code>stderr</code>) of <i>cmd</i> to <i>file</i>
<code>1>&2</code>		<code>cmd 1>&2</code>	stdout to same place as <code>stderr</code>
<code>2>&1</code>		<code>cmd 2>&1</code>	<code>stderr</code> to same place as stdout
<code>&></code>		<code>cmd &> file</code>	Every output of <i>cmd</i> to <i>file</i>

Table 23. redirection

Source: <https://www.howtogeek.com/435903/what-are-stdin-stdout-and-stderr-on-Linux/>

Pipes

Piping is used to pass output to another program or utility, multiple pipes can be used.

Redirecting is used to pass output to either a file or a stream.

Key feature of the unix philosophy (https://en.wikipedia.org/wiki/Unix_philosophy)

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

The real power of Linux commands comes from combinations through piping.

Command	Options	Common usage	What does it?
		cmd1 cmd2	stdout of <i>cmd1</i> piped to <i>cmd2</i>

Table 24 pipe

Recipes:

Search for a keyword in the file names: `ls | grep keyword`

Search for a specific keyword in the history: `history | grep keyword`

tee command, effectively duplicates its input. It is primarily used in conjunction with pipes

Command	Options	Common usage	What does it?
tee		cmd1 tee filename	this command reads standard input and writes it to both standard output and one or more files.
	-a	ls-a tee -a ls_out.txt	append to existing file
		ls-a tee file1 file2	write output into multiple files

Table 25 tee

Command chaining

Command	Options	Common usage	What does it?
cmd1 ; cmd2			Run <i>cmd1</i> then <i>cmd2</i>
cmd1 && cmd2			Run <i>cmd2</i> if <i>cmd1</i> is successful
cmd1 cmd2			Run <i>cmd2</i> if <i>cmd1</i> is not successful
cmd &			Run <i>cmd</i> in background

Table 26 chaining commands

Command substitution

Use `$ ()` to insert the output of one command into the arguments for another command

Example:

```
which ls
# the classic way of working
cp -v /usr/bin/ls .
# instead use substitution
cp -v $(which ls) .
```

(very) Basic Bash scripting

Check: <https://www.networkworld.com/article/3610722/basic-scripting-on-unix-and-linux.html>

Basic steps

1. Create a file using an editor(Nano). Name script file with extension `.sh`
By convention, bash scripts end with a `.sh`. However, bash scripts can run perfectly fine without the `sh` extension.
2. Start the script with `#!/bin/bash`
The first line of the script identifies the shell to be used, that shell will run the commands in the script. It starts with a shebang (`# !`) followed by the shell path.
3. Write some code.
Any command you run on the Linux command line can be run in a script provided it's compatible with the specified shell.
It's a good idea to add comments to scripts to explain what the script is meant to do.
The syntax of a comment: `# comment`
4. Save the script file
5. For executing the script type: `bash filename.sh`

Bash variables

Variables stores any user-defined or system-defined information that can be accessible within the shell.

- useful for passing data to programs or being used in shell scripts.
- Defining a variable is very simple (do not put spaces around `=` symbol)

Variables are assigned in one way and referenced in another. Assign a variable with just its name, but precede the name with a `$` to use it.

```
$ my_num=11
$ echo $my_num
```

echo command displays line of text/string that are passed as an argument . This is a built in command that is mostly used in shell scripts and batch files to output status text to the screen or a file.

Command	Options	Common usage	What does it?
env			Show environment variables
set			Get a list of all shell variables, environmental variables, local variables, and shell functions. Wise to use <code>set less</code>
echo		<code>echo \$NAME</code>	Output value of <code>\$NAME</code> variable
export		<code>export NAME=value</code>	Set <code>\$NAME</code> to <i>value</i> export command promotes a shell variable to an environment variable
\$PATH			Executable search path
\$HOME			Home directory
\$SHELL			Current shell

Table 27. Linux variables

Check: <https://www.digitalocean.com/community/tutorials/how-to-read-and-set-environmental-and-shell-variables-on-Linux>

Extra processing commands

Some useful commands that are used to handle, filter, ... data.

Source: <https://blog.robertelder.org/data-science-Linux-command-line/>

sort

sort is the tool to sort a file. Sorting by itself isn't that useful, but it is an important pre-requisite to a lot of other tasks.

Command	Options	Common usage	What does it?
sort		sort anyfile	Sort anyfile lexicographically
	-n		Sort in numeric order
	-r		sort things in reverse order
	-R		Re-arrange the lines of the input randomly

Table 28. sort

Note: -R can be useful for developing a large number of test cases.

tr

Translate, a tool that can:

- convert character case
- squeeze repeating characters
- delete specific characters
- do basic text replacement

The `tr` command is often an essential addition for special cases fixes and cleanups that may be necessary in some data processing stage.

Command	Options	Common usage	What does it?
tr		tr SET1 [SET2]	it will replace each character in SET1 with each character in the same position in SET2. When SET2 is shorter than SET1, the tr command will, by default, repeat the last character of SET2
	-t		truncate SET1 to the length of SET2:
	-s		Squeeze, remove repeated instances of a character
	-d	tr -d SET1	delete characters in SET1.
	-c	tr -c SET1	search for a complement of SET1. i.e. searching for the inverse of SET1.

Table 29. tr

Recipes:

Convert to upper case: `tr 'a-z' 'A-Z' < file1`

Convert to upper case: `tr '[:lower:]' '[:upper:]' < file1`

simple Caesar cipher as algorithm of encryption:

`cat message | tr 'a-z' 'e-zabcd' > secret`

decrypt:

`cat secret | tr 'e-zabcd' 'a-z' > message`

delete all lowercase letters from the input text: `cat file1 | tr -d 'a-z'`

avoid multiple tabs and whitespaces: `cat file1 | tr -s [:space:] '\t'`

WC

A tool that to obtain word counts and line counts.

Line counts are usually a measure for counting a number of elements.

Command	Options	Common usage	What does it?
wc		<code>wc anyfile</code>	print newline, word, and byte counts for anyfile
	<code>-c</code>		print byte counts
	<code>-m</code>		print character counts
	<code>-w</code>		print word counts
	<code>-l</code>		print newline counts

Table 30. *wc*

Wildcards can be used to check multiple files at once.

`wc -l *.csv`

uniq

The `uniq` command is the tool to detect the adjacent duplicate lines and also deletes the duplicate lines.

Note: `uniq` is not able to detect the duplicate lines unless they are adjacent to each other, therefore it is often seen in a pipe coupled with `sort`. Or simply use `sort -u` instead of `uniq` command.

Command	Options	Common usage	What does it?
uniq		<code>uniq anyfile</code>	the default behaviour is to discard duplicate lines
	<code>-c</code>		count number of occurrences
	<code>-d</code>		only print duplicate lines
	<code>-u</code>		only print unique lines
	<code>-i</code>		ignore case

Table 31. *uniq*

comm

`comm` is a tool for computing the results of set operations: unions, intersections and complements, based on the lines of text in the input files. The command requires 2 files, which it then compares, and returns three columns depending on the uniqueness of the data. The first column is for unique values in file1, the second for unique values in file2, and third for the values that are the same in both files.

The command is useful when you want to learn something about the lines that are either common or different in two different files.

Command	Options	Common usage	What does it?
comm		<code>comm file1 file2</code>	compare two (sorted) files line by line column 1: only in file1 column 2: only in file 2 column 3: in file1 and file2
	<code>-1</code>		suppress column 1
	<code>-2</code>		suppress column 2
	<code>-3</code>		suppress column 3
	<code>--total</code>		Output a summary

Table 32. *comm*

Note: We must sort files using the same sorting standards for the `comm` command to work properly.

```
comm <(sort file1) <(sort file2)
```

split

`split` a file into smaller chunks. Sometimes a file is too large and it needs to be split into smaller parts, `cat` can be used afterwards to combine the pieces back together.

Command	Options	Common usage	What does it?
split		<code>split filename prefix</code>	split filename and prefix is the name you wish to give the small output files
	<code>-b</code>		the number of bytes in each of the smaller files.
	<code>-l</code>		the number of lines of the smaller files (default is 1,000).
	<code>-n</code>		the number of parts to split the original file

Table 33. *split*

cat

concatenate files together into a larger file. Sometimes a file is too large and it needs to be split into smaller parts, `cat` can be used afterwards to combine the pieces back together.

Command	Options	Common usage	What does it?
cat		<code>cat file1 file2 filen</code>	display the content of multiple files
	<code>-s</code>		Omit the empty lines
	<code>-n</code>		display the line number of file content

Table 34. *cat*

Usually used to combine files with the redirection `>` option

```
cat file1.txt file2.txt > mergefile.txt
```

cut

The `cut` command extracts a column of columns of information from a file.

Command	Options	Common usage	What does it?
cut			cut option filename
	<code>-d</code>	<code>cut -d '\,' -f 1 filename</code>	specify a delimiter to use instead of the default TAB delimiter
	<code>-f</code>	<code>cut -f n cut -f n1-n2 cut -f n1,n2,n3</code>	a specified field, a field set, or a field range
	<code>-b</code>	<code>cut -b n cut -b n1-n2 cut -b n1,n2,n3</code>	cut by byte position cut by byte position range cut by byte position list
	<code>-c</code>	<code>cut -c n cut -c n1-n2 cut -c n1,n2,n3</code>	Similar to <code>b</code> , but by character position, some characters take more than a byte for the representation
	<code>--output-delimiter</code>	<code>cut -c 2-5,8 -- output-delimiter=@</code>	Add a custom delimiter for output

Table 35. *cut*

Recipes:

Extract the first byte from each line: `cut -b 1`

Extract everything from character 10 until the end of the line from each line of file:

```
cut -c 10- employees.txt
```

paste

combine files as columns into 1 file. With no FILE, or when FILE is -, read standard input.

Command	Options	Common usage	What does it?
paste			
	-d	paste -d ',' file1 file2 filen	specify a delimiter to use instead of the default TAB delimiter. A list of delimiters can be specified
	-s	paste -s file1 file2 filen	It reads all the lines from a single file and merges all these lines into a single line with each line separated by tab.

Table 36. paste

wget

wget makes file downloads easy, wget is just a command-line tool without any APIs, built for robustness.

Command	Options	Common usage	What does it?
wget		wget url	provide it with the location of a file to download
	-i	wget -i file_with_url	Specify different file-url's in a file and submit this file to wget
	-m		Mirror a site

Table 37. wget

curl

curl (short for "Client URL") is a command line tool that enables data transfer over various network protocols. curl supports a much larger range of protocols, making it a more general-purpose tool than wget.

Command	Options	Common usage	What does it?
curl		curl anyfile	provide it with the location of a file to download

Table 38. curl

The main difference between wget is that curl will show the output in the console. On the other hand, wget will download it into a file.

grep

grep (Global Regular Expression Print) is a tool that can be used to extract matching text from files. While most everyday uses of the command are simple, more advanced uses including regular expressions and more, can make the command quite complicated.

Grep allows to search fast for a particular pattern in a large number of files.

Command	Options	Common usage	What does it?
grep		<code>grep pattern file</code>	Prints all lines in file matching a particular search pattern
	<code>-i</code>	<code>grep -i pattern files</code>	Case insensitive search
	<code>-r</code>	<code>grep -r pattern path</code>	Recursive search through the subdirectories
	<code>-v</code>	<code>grep -v pattern files</code>	Inverted search, print all lines NOT containing the specified pattern
	<code>-o</code>	<code>grep -o</code>	Show matched part of file only
	<code>-n</code>	<code>grep -n pattern file</code>	Include line numbers
	<code>-w</code>	<code>grep -w pattern files</code>	only returns lines where the sought-for string is a whole word and not part of a larger word.
	<code>-L</code>	<code>grep -L pattern files</code>	outputs the names of files that do NOT contain matches for the search pattern
	<code>-l</code>	<code>grep -l pattern files</code>	outputs the names of files that do contain matches for the search pattern

Table 39. grep

Using regular expressions

Taken from <https://www.opensourceforu.com/2012/06/beginners-guide-gnu-grep-basics/>

Use regular expressions with caution. The complexity of regex carries a cost. Regular expressions can be extremely powerful for quickly solving problems, where future maintenance is not a concern.

- KISS – Keep It Simple, Stupid
- Do that, and you will rarely have much trouble
- Start to be clever, and you will shoot yourself in the foot

One point to note is that **regular expressions are NOT wildcards**. The regular expression `'c*t'` does not mean 'match "cat", "cot"' etc. In this case, it means 'match zero or more 'c' characters followed by a t', so it would match 't', 'ct', 'ccct' etc.

There are two types of characters to be found in regular expressions:

- literal characters
- metacharacters

Common regex meta characters (taken from <https://ubuntu.com/blog/regex-basics>) with the corresponding examples from <https://www.maketecheasier.com/cheatsheet/regex/>

Express ion	What does it?	Example
.	Matches any single character (except newlines, normally)	c.t matches "cat", "cut" or "cot"
\	Escape a special character (e.g. \. matches a literal dot)	a*c matches "a*c"
?	The preceding character may or may not be present (e.g. /hell?o/ would match hello or helo)	ma?ke matches "make", "mke"
*	Any number of the preceding character is allowed (e.g. .* will match any single-line string, including an empty string, and gets used a lot)	12*3 matches "13", "123", "1223", "12223". It can be use together with . (dot) such as m.*ne matches "machine". Using .* by itself is meaningless as it matches everything and return the full result.
+	One or more of the preceding character (.+ is the same as .* except that it won't match an empty string)	12+3 matches "123", "1223", "12223"
	"or", match the preceding section or the following section (e.g. hello mad will match "hello" or "mad")	col(o ou)r matches "color", "colour"
()	group a section together. This can be useful for conditionals ((a b)), multipliers ((hello)+), or to create groups for substitutions (see below)	(ak) matches "make", "take", ,
{ }	Specify how many of the preceding character (e.g. a{12} matches 12 "a"s in a row)	
	{x}	Matches the preceding element x times
	{x,y}	Matches the preceding element between x and y times
	{x,}	Matches the preceding element at least x times
	{,y}	Matches the preceding element between 0 and y times
		12{3}5 matches "12225"

[]	Match any character in this set. - defines ranges (e.g. [a-z] is any lowercase letter), ^ means “not” (e.g. [^,]+ match any number of non-commas in a row)	[abc] matches "a","b" or "c" in the string "abc". a[^b]c matches "aec", "acc", "adc", but not "abc"
^	Beginning of line	^he matches "hello", "hell", "help", "he is a boy"
\$	End of line	ed\$ matches "acted", "bed", "greed"

Table 40. regex meta characters

POSIX Character Classes. [:class:] (an alternate method of specifying a range of characters to match.)

(<https://tldp.org/LDP/abs/html/x17129.html>)

[:alnum:]	matches alphabetic or numeric characters. This is equivalent to A-Za-z0-9.
[:alpha:]	matches alphabetic characters. This is equivalent to A-Za-z.
[:blank:]	matches a space or a tab.
[:cntrl:]	matches control characters.
[:digit:]	matches (decimal) digits. This is equivalent to 0-9.
[:graph:]	(graphic printable characters). Matches characters in the range of ASCII 33 - 126. This is the same as [:print:], below, but excluding the space character.
[:lower:]	matches lowercase alphabetic characters. This is equivalent to a-z.
[:print:]	(printable characters). Matches characters in the range of ASCII 32 - 126. This is the same as [:graph:], above, but adding the space character.
[:space:]	matches whitespace characters (space and horizontal tab).
[:upper:]	matches uppercase alphabetic characters. This is equivalent to A-Z.
[:xdigit:]	matches hexadecimal digits. This is equivalent to 0-9A-Fa-f.

Table 41. posix character classes

Important: POSIX character classes generally require quoting or double brackets ([[]]).

Monitoring resources

Check disk space

Command	Options	Common usage	What does it?
du			Disk usage Be careful when running this command on directories with large amounts of data
	-s		Simplify the output
	-h	du -sh	Human readable output
		du -sh directory	Specify a directory
df			disk free, shows the amount of space taken up by different drives.
		df -h	Human readable output

Table 42. check disk space

Process foreground / background

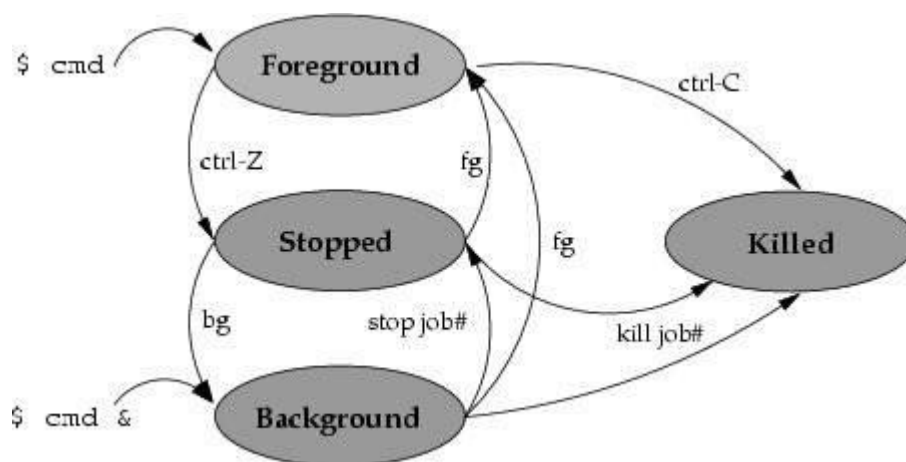


Figure 6 Foreground vs background (<https://www.baeldung.com/linux/foreground-background-process#:~:text=A%20process%20that%20connects%20to,is%20called%20a%20background%20job.>)

Run a command, if you need to free up the terminal, you can stop the process. Ctrl-z stops a process

```
sleep 10000
```

```
^Z
```

Get a list of jobs in the background: `jobs`

```
jobs -l
```

Bring a job back to the foreground: `fg`
 multiple stopped jobs, use % and specify the job ID

`fg %1`

Bring a job to the background: `bg`
 multiple stopped jobs, use % and specify the job ID

`bg %1`

Check with jobs: the process is in the background, but running instead of being stopped.

Process Management

A process is a running program. Each process on the system has a numeric process ID (PID)

Command	Options	Common usage	What does it?
ps			Process status, showing a snapshot of processes, By default, it will only show you the processes running in the local shell
		<code>ps -elf</code>	Display detailed information about running processes e = display all processes including system processes l = long detailed listing
		<code>ps -auxf</code>	a = show processes for all users u = display the process's user/owner x = also show processes not attached to a terminal f = forest view
pstree			Information presented in a forest view, less informative than <code>ps -auxf</code>
		<code>pstree -pn</code>	Show extra information
top			Show real time processes
Htop			More elaborated than top
kill		<code>kill pid</code>	Kill process with id <i>pid</i>
pkill		<code>pkill name</code>	Kill process with name <i>name</i>
killall		<code>killall name</code>	Kill all processes with names beginning <i>name</i>

Table 43 process management commands

```

frankvp@CRD-L-08004: /mnt/c/Users/u0015831$ ps
  PID TTY          TIME CMD
  131 pts/0    00:00:00 bash
  144 pts/0    00:00:00 ps
frankvp@CRD-L-08004: /mnt/c/Users/u0015831$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0 08:08 ?        00:00:00 /init
root          94         1  0 08:09 ?        00:00:00 /init
root         95         1  0 08:09 ?        00:00:00 /init
frankvp       96        95  0 08:09 pts/1    00:00:00 -bash
root        129         1  0 16:49 ?        00:00:00 /init
root        130        129  0 16:49 ?        00:00:00 /init
frankvp      131        130  0 16:49 pts/0    00:00:00 -bash
frankvp     145        131  0 16:49 pts/0    00:00:00 ps -ef
frankvp@CRD-L-08004: /mnt/c/Users/u0015831$ pstree
init--init--init--bash
  |
  |--init--init--bash--pstree
  |
  |--{init}
frankvp@CRD-L-08004: /mnt/c/Users/u0015831$ pstree -pn
init(1)---{init}(9)
  |--init(94)---init(95)---bash(96)
  |--init(129)---init(130)---bash(131)---pstree(147)
frankvp@CRD-L-08004: /mnt/c/Users/u0015831$

```

Processes

Source: <https://Linuxjourney.com/lesson/process-states>

```

frankvp@CRD-L-08004:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.0  0.0      896   560 ?        S1   Sep16   0:00 /init
root        10   0.0  0.0      904    84 ?        Ss   Sep16   0:00 /init
root        11   0.0  0.0      904    88 ?        R    Sep16   0:00 /init
frankvp     12   0.0  0.0   10696  5952 pts/0    Ss   Sep16   0:01 -bash
root       2205   0.0  0.0      904    84 ?        Ss   14:06   0:00 /init
root       2206   0.0  0.0      904    88 ?        S    14:06   0:00 /init
frankvp    2207   0.0  0.0   10056  5124 pts/1    Ss+  14:06   0:00 -bash
root       2284   0.0  0.0      904    84 ?        Ss   14:06   0:00 /init
root       2285   0.0  0.0      904    88 ?        S    14:06   0:00 /init
frankvp    2286   0.0  0.0   10056  5144 pts/2    Ss   14:06   0:00 -bash
frankvp    2299   0.0  0.0      8492  3512 pts/2    S+   14:06   0:00 nano
frankvp    2309   0.0  0.0     10620  3372 pts/0    R+   14:20   0:00 ps aux
frankvp@CRD-L-08004:~$

```

in the STAT column, you'll see lots of values. A Linux process can be in a number of different states. The most common state codes you'll see are described below:

- R: running or runnable (waiting for the CPU to process it)
- S: Interruptible sleep, waiting for an event to complete, such as input from the terminal
- D: Uninterruptible sleep, processes that cannot be killed or interrupted with a signal, usually to make them go away you have to reboot or fix the issue
- Z: Zombie, are terminated processes that are waiting to have their statuses collected
- T: Stopped, a process that has been suspended/stopped

Useful links

- William Shotts: The Linux Command Line - <https://linuxcommand.org/tlcl.php>
- The Linux Documentation Project - <https://tldp.org/>