SANDBOX

1. Turn it on by going to **current project – Particular project – Get a sandbox**
2. **Use Ubuntu 28 or anyone, then close view**
3. Go back to Sandbox on the Dashboard.
4. Use **Web Terminal** (best) or **SSH** or **SFTP**
5. c. ctrl + c...ends a program
6. use webterm on web or
7. SSH , copy it and use CMD, paste it and enter and press yes
8. Do not destroy your sandbox...not to loose all files or folders.

Commands in Sandbox

1. **touch readme.txt** to create readme.txt file
2. **exit** to exit
3. **clear** like cls

Intranet Colors

1. Blue – a project
2. Light blue – 2nd deadline
3. Green – evaluation
4. Yellow – Evaluation
5. Purple – Peer learning day

SHELL Navigation

1. **pwd**command (print working directory)…shows the current directory
2. cd Desktop….changes the directory to desktop
3. cd .. takes you to parent
4. cd ./Desktop/mine …for absolute path

or use cd Desktop/mine

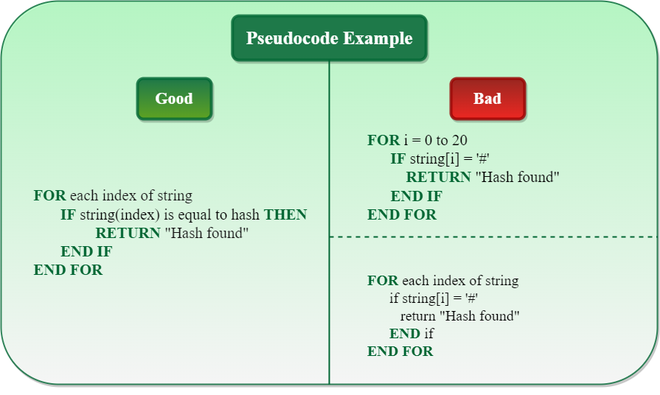
1. clear….to do cls as in cmd
2. ls…list all files and folders in a directory
3. whoami…tells you who the user is
4. less index.html ….. will open this file
5. q …..to quit maybe a file is opening
6. touch index.html ………………creates a file
7. touch index.html style.css script.js ……….. to crate multiple files
8. ls -l … tells you more about all folders and files
9. ls -a to see all dot files. File names that begin with a period character are hidden. This only means that ls will not list them unless we say ls -a.
10. cp index.html about.html …..will copy index.html into about.html
11. File index.html …. Will tell you the type of the file
12. mv index.html my\_index.html ….. will rename index to my\_index
13. mv index.html Desktop ….. will move the file into a new folder called Desktop
14. rm about.html … to delete file
15. mkdir Folder1 … will create folder1
16. rmdir folder1 … to remove a folder
17. exit …. To close
18. cd ….. takes you to home
19. cd ~user\_name. in this case, cd will change the working directory to the home directory of the specified user.

Convert decimal to binary, octal and hexadecimal

|  |  |  |  |
| --- | --- | --- | --- |
| **Decimal** | **Octal** | **Hexadecimal** | **Binary** |
| 0 | /000 | 0x00 | 0 0 0 0   0 0 0 0 |
| 1 | /001 | 0x01 | 0 0 0 0   0 0 0 1 |
| 2 | /002 | 0x02 | 0 0 0 0   0 0 1 0 |

'0x' means that the number that follows is in hexadecimal. It's a way of unambiguously stating that a number is in hex, and is a notation recognized by C compilers and some assemblers.

**NB:** All works must be done in a **Sandbox Ubuntu 20.04**

****

# The Feynman Technique of Learning

## Step 1 – Study

## Step 2 – Teach

.

## Step 3 – Fill the Gaps

## Step 4 – Simplify

### “If you can’t explain it to a six-year-old, you don’t understand it yourself.”

### – Albert Einstein

# Note: Ensure you wake the Sandbox before a week

 Unix-like operating systems such as Linux and Ubuntu

* [**pwd**](http://linuxcommand.org/lc3_man_pages/pwdh.html) (print working directory),
* [**cd**](http://linuxcommand.org/lc3_man_pages/cdh.html) (change directory), and
* [**ls**](http://linuxcommand.org/lc3_man_pages/ls1.html) (list files and directories),
* [**less**](http://linuxcommand.org/lc3_man_pages/less1.html) (view text files)
* [**file**](http://linuxcommand.org/lc3_man_pages/file1.html) (classify a file's contents)

# cd /usr/bin absolute directory

Where an absolute pathname starts from the root directory and leads to its destination, a relative pathname starts from the working directory. To do this, it uses a couple of special notations to represent relative positions in the file system tree. These special notations are "." (dot) and ".." (dot dot).

The "." notation refers to the working directory itself and the ".." notation refers to the working directory's parent directory. Here is how it works. Let's change the working directory to /usr/bin again:

# Nb cd ./mine is same as cd mine

# Though Linux supports long file names which may contain embedded spaces and punctuation characters, limit the punctuation characters to period, dash, and underscore. Most importantly, do not embed spaces in file names. If you want to represent spaces between words in a file name, use underscore characters.

## **ls**

The **ls** command is used to list the contents of a directory. It is probably the most commonly used Linux command. It can be used in a number of different ways. Here are some examples:

|  |  |
| --- | --- |
| Examples of the ls command | |
| **Command** | **Result** |
| **ls** | List the files in the working directory |
| **ls /bin** | List the files in the /bin directory (or any other directory we care to specify) |
| **ls -l** | List the files in the working directory in long format |
| **ls -l /etc /bin** | List the files in the /bin directory and the /etc directory in long format |
| **ls -la ..** | List all files (even ones with names beginning with a period character, which are normally hidden) in the parent of the working directory in long format |

[Previous](http://linuxcommand.org/lc3_lts0020.php) | [Contents](http://linuxcommand.org/lc3_learning_the_shell.php#contents) | [Next](http://linuxcommand.org/lc3_lts0040.php)

# Looking Around

Now that we know how to move from working directory to working directory, we're going to take a tour of our Linux system and, along the way, learn some things about what makes it tick. But before we begin, we have to learn about some tools that will come in handy during our journey. These are:

* [**ls**](http://linuxcommand.org/lc3_man_pages/ls1.html) (list files and directories)
* [**less**](http://linuxcommand.org/lc3_man_pages/less1.html) (view text files)
* [**file**](http://linuxcommand.org/lc3_man_pages/file1.html) (classify a file's contents)

# Manipulating Files

This lesson will introduce the following commands:

* [**cp**](http://linuxcommand.org/lc3_man_pages/cp1.html) - copy files and directories
* [**mv**](http://linuxcommand.org/lc3_man_pages/mv1.html) - move or rename files and directories
* [**rm**](http://linuxcommand.org/lc3_man_pages/rm1.html) - remove files and directories
* [**mkdir**](http://linuxcommand.org/lc3_man_pages/mkdir1.html) - create directories
* Uname ----tells you the operating system

## **ls**

The **ls** command is used to list the contents of a directory. It is probably the most commonly used Linux command. It can be used in a number of different ways. Here are some examples:

|  |  |
| --- | --- |
| Examples of the ls command | |
| **Command** | **Result** |
| **ls** | List the files in the working directory |
| **ls /bin** | List the files in the /bin directory (or any other directory we care to specify) |
| **ls -l** | List the files in the working directory in long format |
| **ls -l /etc /bin** | List the files in the /bin directory and the /etc directory in long format |
| **ls -la ..** | List all files (even ones with names beginning with a period character, which are normally hidden) in the parent of the working directory in long format |

These examples also point out an important concept about commands. Most commands operate like this:

*command -options arguments*

where *command* is the name of the command, *-options* is one or more adjustments to the command's behavior, and *arguments* is one or more "things" upon which the command operates.

In the case of **ls**, we see that **ls** is the name of the command, and that it can have one or more options, such as **-a** and **-l**, and it can operate on one or more files or directories.

### A Closer Look at Long Format

If we use the **-l** option with **ls**, you will get a file listing that contains a wealth of information about the files being listed. Here's an example:

**-rw------- 1 me me 576 Apr 17 2019 weather.txt**

**drwxr-xr-x 6 me me 1024 Oct 9 2019 web\_page**

**-rw-rw-r-- 1 me me 276480 Feb 11 20:41 web\_site.tar**

**-rw------- 1 me me 5743 Dec 16 2018 xmas\_file.txt**

**---------- ------- ------- -------- ------------ -------------**

**| | | | | |**

**| | | | | File Name**

**| | | | |**

**| | | | +--- Modification Time**

**| | | |**

**| | | +------------- Size (in bytes)**

**| | |**

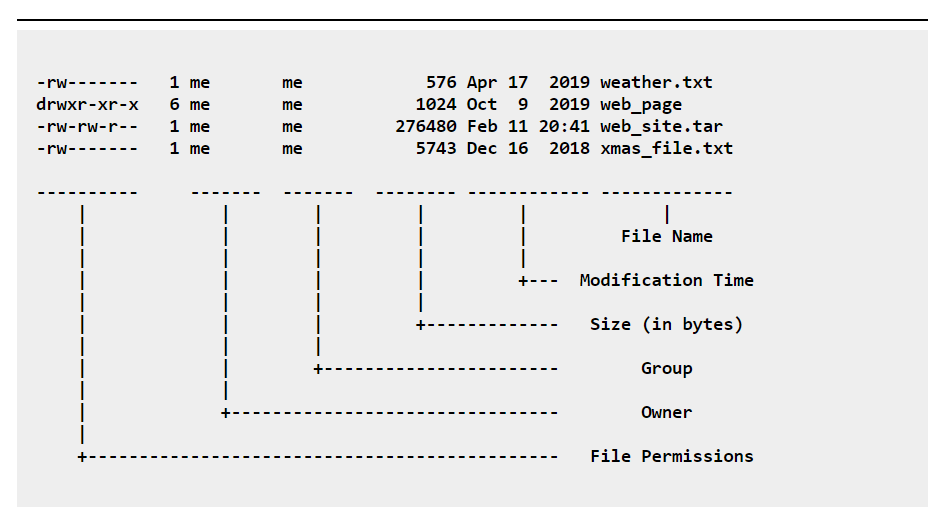
**| | +----------------------- Group**

**| |**

**| +-------------------------------- Owner**

**|**

**+---------------------------------------------- File Permissions**

**File Name**

The name of the file or directory.

**Modification Time**

The last time the file was modified. If the last modification occurred more than six months in the past, the date and year are displayed. Otherwise, the time of day is shown.

**Size**

The size of the file in bytes.

**Group**

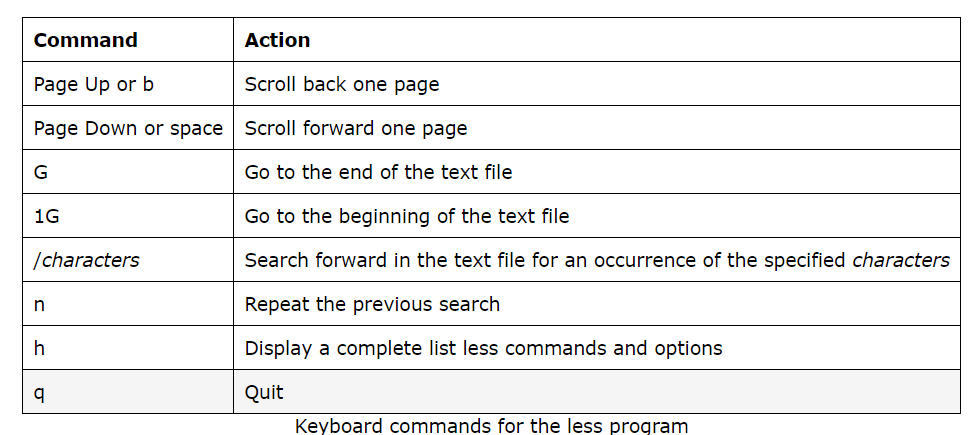
The name of the group that has file permissions in addition to the file's owner.

**Owner**

The name of the user who owns the file.

**File Permissions**

A representation of the file's access permissions. The first character is the type of file. A "-" indicates a regular (ordinary) file. A "d" indicates a directory. The second set of three characters represent the **read, write,** and **execution** rights of the file's owner. The next three represent the rights of the file's group, and the final three represent the rights granted to everybody else. We'll discuss this in more detail in a later lesson.

**LESS**

The **file** program can recognize most types of files, such as:

|  |  |  |
| --- | --- | --- |
| Various kinds of files | | |
| **File Type** | **Description** | **Viewable as text?** |
| ASCII text | The name says it all | yes |
| Bourne-Again shell script text | A **bash** script | yes |
| ELF 64-bit LSB executable | An executable binary program | no |
| ELF 64-bit LSB shared object | A shared library | no |
| GNU tar archive | A tape archive file. A common way of storing groups of files. | no, use **tar tvf** to view listing. |
| gzip compressed data | An archive compressed with **gzip** | no |
| HTML document text | A web page | yes |
| JPEG image data | A compressed JPEG image | no |
| PostScript document text | A PostScript file | yes |
| Zip archive data | An archive compressed with **zip** | no |

cp -u \*.html mine will copy all html files to the folder **mine** assuming user@DESKTOP-AGFEK4G MINGW64 ~/Desktop/mine/subfolder (master) is the path and user@DESKTOP-AGFEK4G MINGW64 ~/Desktop/mine/subfolder/newone

## **Wildcards**

Before we begin with our commands, we'll first look at a shell feature that makes these commands so powerful. Since the shell uses filenames so much, it provides special characters to help you rapidly specify groups of filenames. These special characters are called *wildcards*. Wildcards allow you to select filenames based on patterns of characters. The table below lists the wildcards and what they select:

|  |  |
| --- | --- |
| Summary of wildcards and their meanings | |
| **Wildcard** | **Meaning** |
| **\*** | Matches any characters |
| **?** | Matches any single character |
| **[*characters*]** | Matches any character that is a member of the set *characters*. The set of characters may also be expressed as a *POSIX character class* such as one of the following:   |  |  | | --- | --- | | POSIX Character Classes | | | **[:alnum:]** | Alphanumeric characters | | **[:alpha:]** | Alphabetic characters | | **[:digit:]** | Numerals | | **[:upper:]** | Uppercase alphabetic characters | | **[:lower:]** | Lowercase alphabetic characters | |
| **[!*characters*]** | Matches any character that is not a member of the set *characters* |

Using wildcards, it is possible to construct very sophisticated selection criteria for filenames. Here are some examples of patterns and what they match:

|  |  |
| --- | --- |
| Examples of wildcard matching | |
| **Pattern** | **Matches** |
| \* | All filenames |
| g\* | All filenames that begin with the character "g" |
| b\*.txt | All filenames that begin with the character "b" and end with the characters ".txt" |
| Data??? | Any filename that begins with the characters "Data" followed by exactly 3 more characters |
| [abc]\* | Any filename that begins with "a" or "b" or "c" followed by any other characters |
| [[:upper:]]\* | Any filename that begins with an uppercase letter. This is an example of a character class. |
| BACKUP.[[:digit:]][[:digit:]] | Another example of character classes. This pattern matches any filename that begins with the characters "BACKUP." followed by exactly two numerals. |
| \*[![:lower:]] | Any filename that does not end with a lowercase letter. |

You can copy multiple files simultaneously into another directory. In this example, copy the files named main.c, demo.h and lib.c into a directory named backup in the current directory:  
$ cp main.c demo.h lib.c backup  
If backup is located in /home/project/ directory or folder, then use full path as follows:  
$ cp main.c demo.h lib.c /home/project/backup/

For mine : **cp about.html see.html newone** will copy the files listed into the folder **mine**

### **Copy a file to another directory**

To copy a file from your current directory into another directory called /tmp/, enter:  
$ cp filename /tmp  
$ ls /tmp/filename  
$ cd /tmp  
$ ls  
$ rm filename

### **Verbose option**

To see files as they are copied pass the -v option as follows to the cp command:  
$ cp -v filename.txt filename.bak  
$ cp -v foo.txt /tmp  
Here is what I see:

foo.txt -> /tmp/foo.txt

### **Preserve file attributes**

To copy a file to a new file and preserve the modification date, time, and access control list associated with the source file, enter:  
$ cp -p file.txt /dir1/dir2/  
$ cp -p filename /path/to/new/location/myfile  
This option (-p) forces cp to preserve the following attributes of each source file in the copy as allowed by permissions:

1. Modification time/date
2. Access time
3. File flags
4. File mode
5. User ID (UID)
6. Group ID (GID)
7. Access Control Lists (ACLs)
8. Extended Attributes (EAs)

### **Copying all files**

The star wildcard represents anything i.e. all files. To copy all the files in a directory to a new directory, enter:

$ cp \* /home/tom/backup

The star wildcard represents anything whose name ends with the .doc extension. So, to copy all the document files (\*.doc) in a directory to a new directory, enter:

$ cp \*.doc /home/tom/backup

### **Recursive copy**

To copy a directory, including all its files and subdirectories, to another directory, enter (copy directories recursively):  
$ cp -R \* /home/tom/backup

## **cp**

The **cp** program copies files and directories. In its simplest form, it copies a single file:

[me@linuxbox me]$ **cp *file1 file2***

It can also be used to copy multiple files (and/or directories) to a different directory:

[me@linuxbox me]$ **cp *file... directory***

**A note on notation:** ... signifies that an item can be repeated one or more times.

Other useful examples of **cp** and its options include:

|  |  |
| --- | --- |
| Examples of the cp command | |
| **Command** | **Results** |
| cp *file1 file2* | Copies the contents of *file1* into *file2*. If *file2* does not exist, it is created; **otherwise, *file2* is silently overwritten with the contents of *file1*.** |
| cp -i *file1 file2* | Like above however, since the "-i" (interactive) option is specified, if *file2* exists, the user is prompted before it is overwritten with the contents of *file1*. |
| cp *file1 dir1* | Copy the contents of *file1* (into a file named *file1*) inside of directory *dir1*. |
| cp -R *dir1 dir2* | Copy the contents of the directory *dir1*. If directory *dir2* does not exist, it is created. Otherwise, it creates a directory named *dir1* within directory *dir2*. |

## **mv**

The **mv** command moves or renames files and directories depending on how it is used. It will either move one or more files to a different directory, or it will rename a file or directory. To rename a file, it is used like this:

[me@linuxbox me]$ **mv *filename1 filename2***

To move files (and/or directories) to a different directory:

[me@linuxbox me]$ **mv *file... directory***

Examples of **mv** and its options include:

|  |  |
| --- | --- |
| Examples of the mv command | |
| **Command** | **Results** |
| mv *file1 file2* | If *file2* does not exist, then *file1* is renamed *file2*. **If *file2* exists, its contents are silently replaced with the contents of *file1*.** |
| mv -i *file1 file2* | Like above however, since the "-i" (interactive) option is specified, if *file2* exists, the user is prompted before it is overwritten with the contents of *file1*. |
| mv *file1 file2 dir1* | The files *file1* and *file2*are moved to directory *dir1*. If *dir1* does not exist, **mv** will exit with an error. |
| mv *dir1 dir2* | If *dir2* does not exist, then *dir1* is renamed *dir2*. If *dir2* exists, the directory *dir1* is moved within directory *dir2*. |

## **rm**

The **rm** command removes (deletes) files and directories.

[me@linuxbox me]$ **rm *file...***

Using the recursive option (-r), **rm** can also be used to delete directories:

[me@linuxbox me]$ **rm -r *directory...***

Examples of **rm** and its options include:

|  |  |
| --- | --- |
| Examples of the rm command | |
| **Command** | **Results** |
| rm *file1 file2* | Delete *file1* and *file2*. |
| rm -i *file1 file2* | Like above however, since the "-i" (interactive) option is specified, the user is prompted before each file is deleted. |
| rm -r *dir1 dir2* | Directories *dir1* and *dir2* are deleted along with all of their contents. |

## **Be careful with rm!**

Linux does not have an undelete command. Once you delete something with **rm**, it's gone. You can inflict terrific damage on your system with **rm** if you are not careful, particularly with wildcards.

***Before you use*rm*with wildcards, try this helpful trick:*** construct your command using **ls** instead. By doing this, you can see the effect of your wildcards before you delete files. After you have tested your command with **ls**, recall the command with the up-arrow key and then substitute **rm** for **ls** in the command.

## **mkdir**

The **mkdir** command is used to create directories. To use it, you simply type:

[me@linuxbox me]$ **mkdir *directory...***

## **Using Commands with Wildcards**

Since the commands we have covered here accept multiple file and directories names as arguments, you can use wildcards to specify them. Here are a few examples:

|  |  |
| --- | --- |
| Command examples using wildcards | |
| **Command** | **Results** |
| cp \*.txt text\_files | Copy all files in the current working directory with names ending with the characters ".txt" to an existing directory named *text\_files*. |
| mv dir1 ../\*.bak dir2 | Move the subdirectory *dir1* and all the files ending in ".bak" in the current working directory's parent directory to an existing directory named *dir2*. |
| rm \*~ | Delete all files in the current working directory that end with the character "~". Some applications create backup files using this naming scheme. Using this command will clean them |

**Appendix A. UNIX System V Signals**

|  |
| --- |
| *Symbol*  *Number Action Meaning*  SIGHUP 1 exit Hangs up.  SIGINT 2 exit Interrupts.  SIGQUIT 3 core dump Quits.  SIGILL 4 core dump Illegal instruction.  SIGTRAP 5 core dump Trace trap.  SIGIOT 6 core dump IOT instruction.  SIGEMT 7 core dump MT instruction.  SIGFPE 8 core dump Floating point exception.  SIGKILL 9 exit Kills (cannot be caught or ignored).  SIGBUS 10 core dump Bus error.  SIGSEGV 11 core dump Segmentation violation.  SIGSYS 12 core dump Bad argument to system call.  SIGPIPE 13 exit Writes on a pipe with no one to read it.  SIGALRM 14 exit Alarm clock.  SIGTERM 15 exit Software termination signal. |

# SANDBOX

# This is your playing environment. It has all dependencies. So you use it without depending on your/ that is in isolation Operating System.

# Exit your sandbox but don’t destroy.

# echo will write into a file. Whatever you give to echo is given back to you

# echo “Jedy is good” > readme.txt to write into readme.txt (the > will overwrite the former but to keep original use >>)

# echo “Jedy is good” >> readme.txt

# cat filename will give you the content of the file

# Folders are blue, files white

# ls -l ……..note the – is called a flask.

# ls -r list the directory in reverse form

# HOW TO SEE THE CONTENTS OF A FILE

## **The**cat**Command**

The cat command is the simplest way to view the contents of a file. **It displays the contents of the file(s) specified on to the output terminal**.

Let’s look at an example:

* cat boy.txt

This will print the contents of the file boy.txt

* Sometimes, we might want to number the lines in the output.

We can do this by using the *-n* option:

cat -n boy.txt

* Note that the line numbering starts from 1.Let’s look at a few other significant options:
* *-e* displays control and non-printing characters followed by a $ symbol at the end of each line
* *-t* each tab will display as ^I and each form feed will display as ^L
* *-v* displays control and non-printing characters

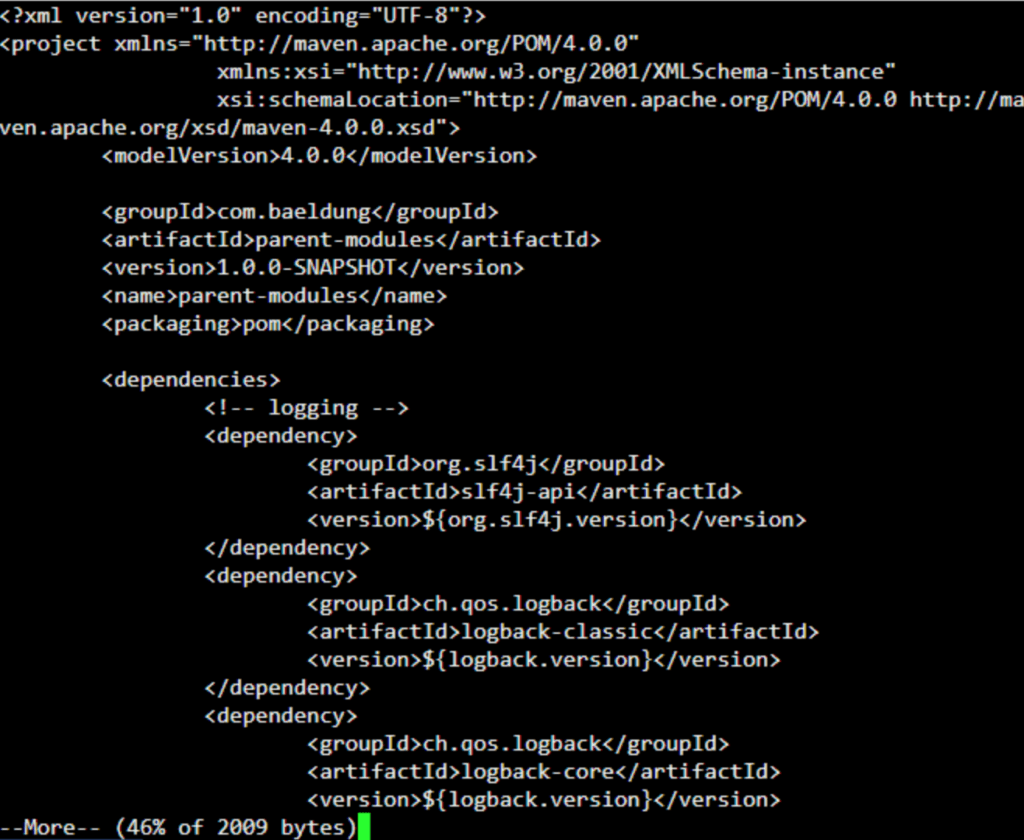
## **The**more**Command**

The **cat** command is all well and good for small files. But, if the file is large, the contents will zoom past and we’ll only see the last screen worth of content.

One way to overcome this is by using the **more** command.

**The more command displays the contents of the file one screen at a time for large files**. If the contents of the file fit a single screen, the output will be the same as the cat command.

Let’s use this command on a pom.xml file:

[](https://www.baeldung.com/wp-content/uploads/2019/11/Screenshot-2019-11-20-at-10.38.56.png)

Note the text “–More–(46%)” at the end of the output. Here, the text “(46%)” tells us that the file is big and we’re currently seeing only 46% of the content. This percentage will increase as we traverse through the file.

The cursor will stay at the end of this text. Then, we can **scroll through the contents of the file using the Enter key, one line at a time**.

We can also **scroll through the file page by page by using the Space bar**. And to scroll back to the previous page, we can use the b key.**We’ll use the q key to go back to the command prompt**.

**The more command can also be used to view multiple files**. We just have to list each of them one after another:

more pom.xml a.txt b.txt

### **Commands**

As well as the keys used above, we can use a few other commands while viewing the file.

Let’s look at a few important ones:

* [k]/<text> searches for the kth match of the regular expression <text>
* !<cmd> executes <cmd> in a subshell
* [k]:n goes to the kth next file
* [k]:p goes to the kth previous file
* [k]z displays the next k lines of text. If we don’t specify k, it defaults to the current page size
* :f displays the current file name and line number
* = displays the current line number

Also, **we can use h or ? at any point to list all the commands that can be used with more**.

The more command also allows us to specify various options on the command line to customize the output. Let’s look at a few of these.

### **3.2. Alter Page Size**

Let’s suppose we want to view only a certain number of lines at a time. We can do this by specifying the number of lines as an option:

more -5 pom.xml

This will display the first 5 lines of the file instead of a screen worth of content.

Subsequently, when we use the Space bar, the next 5 lines will be shown. Similarly, the b key will show the previous 5 lines.

### **3.3. Specify Start of Content**

We can also specify the line number in the file from where we want to start viewing the content:

more +10 pom.xml

This will cause the output to start from the 10th line.

### **3.4. Start From the First Occurrence of a Text**

It’s possible to search for a particular text in the file and start viewing the file from that point:

more +/slf4j pom.xml

The above command will output the file contents from the first occurrence of the text “slf4j” in the file.

Apart from the above, the *more* command provides a few other options. Let’s briefly look at these:

* *-d*this option is used to help the user navigate; it’ll prompt the user with the message “[Press space to continue, ‘q’ to quit.]”; it’ll also display “[Press ‘h’ for instructions.]” when an illegal key is pressed
* *-l*the *more* command usually treats *^L* (form feed) as a special character and will pause after any line that contains a form feed; the *-l* option will prevent this behavior
* *-f*this option stops the wrapping of long lines
* *-p*clears the screen and then displays the text
* *-c*displays the pages on the same area by overlapping the previously displayed text
* *-u*suppresses underlining
* *-s* squeezes multiple blank lines into one

## **The**less**Command**

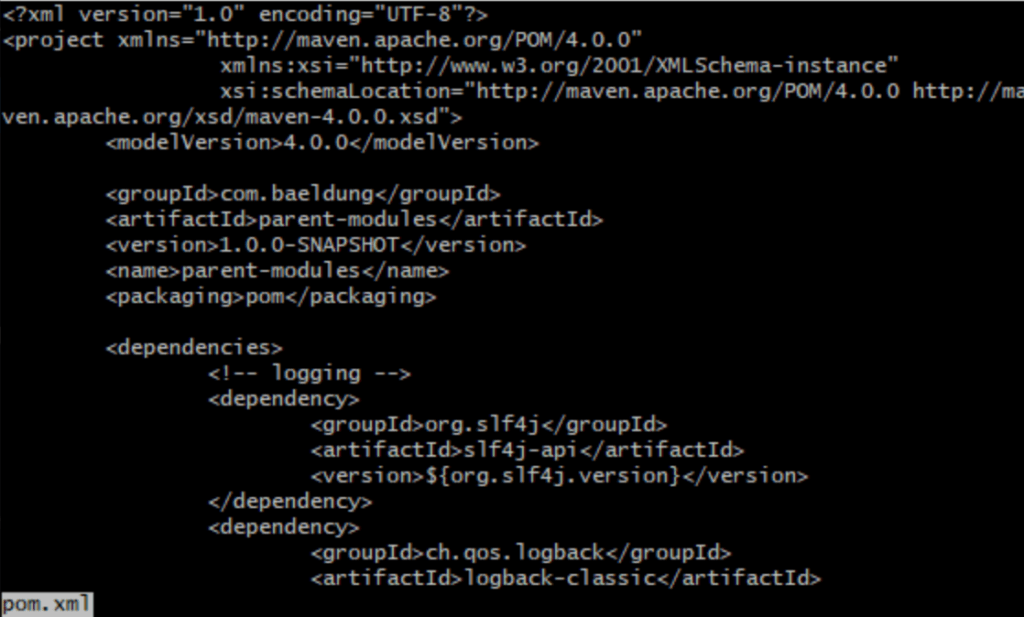
Now, let’s move to the less command. **The less command is similar to the more command but provides extensive features**. One important one is that **it allows backward as well as forward movement in the file, even with pipes**.

Also, since it does not read the entire file before starting, **it starts up faster compared to text editors** — especially when we’re viewing large files.

To view our pom.xml file, we’ll simply replace more with less:

less pom.xml

This should show us the first page of the file with a prompt at the end:

[](https://www.baeldung.com/wp-content/uploads/2019/11/Screenshot-2019-11-20-at-10.40.00.png)

Note how the file name is displayed at the prompt.

Unlike more, **if the file content fits the screen, less will still display the prompt**. To override this we have to specify the -F option.

Like cat, it’s possible to number the lines in the file. For this, we’ve to specify the -N or –LINE-NUMBERS option.

# mv file1 file2 will rename file1 to file2

# You can only run maximum of 3 sandboxes at a time.

# Whoami will tell you the user while uname will tell you the operating system

# mv \*.txt ../Folder1 will move .txt files from the current folder to Folder1, assuming Folder1 and the current folder are inside another folder.

# mv \*.txt .. will copy the files to the parent folder of the current folder.

# mv index.html main.py Folder3 will move these files from the parent directory to the folder called Folder3.

# I can be in Desktop and use ls Folder4 to see the contents of folder4 if folder4 is inside Desktop.

# mv blog\_\* folder5 This means move any file that has blog and anything in front of it to folder5

# cp main.py main2.py will just make a copy of main.py as a duplicate.

# rm -r Folder6 will remove a folder but rm Folder6 wont do it because of fear it has some files.

# man man gives you the manual on manual. Press h for help or q to quit.

# man ls gives you manual for ls

# cd / takes you to the root of your directories.

# cd – takes you to the last directory where you were.

# Tilde (~) shows the user’s home folder of the current directory. The user can type commands like cd/ at the command prompt. This command changes the directory to the root folder.

What is the difference between **cd** and **cd ~**

**~ stands for /home/username location, so you save some time when typing**. For example cd /home/username/Downloads is the same as cd ~/Downloads, but less typing. Both commands do the same, change the working directory to that location.

1. **vi readme.txt** will allow you to edit a file
2. **rmdir** removes empty directory.
3. Use **tab** key to do autocomplete

# RESOURCES

# <https://www.online-convert.com/> doc to png

# <https://app.zenflowchart.com/flowchart/> flowchart maker or <https://app.diagrams.net/>