<table>
<tr><th>Command</th><th>Function</th><th>Notes</th></tr>
<tr><td colspan="3">Section 1</td></tr>
<tr><td>ls</td><td>ls (stands for "list storage") : display files and folders</td><td>Use ls -la to see all files, including hidden files (files that start with a '.' , often called 'dot-files')</td></tr>
<tr><td>touch</td><td>- Creates a new empty, zero-byte file
OR
- Updates the date/time stamp of an existing file</td><td>Very handy to create files on the fly and then edit them with an editor such as vim, nano, or VSCode.
The touch command creates a new zero-byte (empty) file with the name we specify. If we touch an existing file, it only modifies the date/time stamp and does NOT modify the contents.</td></tr>
<tr><td>cat</td><td>Comes from the word 'con<u>cat</u>enate' which means to chain separate files together into one file.

Most often, however, this command is used to print a file on the screen if it's a smaller file.

If the file has too many lines, cat scrolls the lines too fast and you'll only see the last screen-full of lines.

You can also concatenate or join append multiple files together into a single file.</td><td>cat <fileName> : prints the contents of a file on the screen

cat -nb <fileName> : shows line numbers for every line, including blank lines.

To join multiple files into one file:
cat file-1 file-2 file-3 > final-fileName

Each file is appended in the order it is listed and the '>' directs the output to new file named final-fileName</td></tr>
<tr><td>more</td><td>Instead of cat, the more command also prints a file to the screen, BUT only one screen-full at a time if it's a long file. When you're ready to see the next screen press the down-arrow key.

The problem is that it can only go in one direction; you can't scroll back up.

[CTRL]-c to break out of the more command</td><td>Use the more command to read long files by letting you scroll down at your own pace.
However, more only scrolls in one direction; you cannot scroll back up.

<u>Use the less command to scroll backwards and forwards.</u></td></tr>
<tr><td>less</td><td>This is the BEST command to use if you want to view a long file. Like the more command, less scrolls one screen at a time. However you can scroll forwards or backwards.

[CTRL] c to break out of the less command</td><td>less -N <fileName> : shows line numbers</td></tr>
<tr><td colspan="3">Section 2</td></tr>
<tr><td>cp</td><td>copy file to same location with a different name OR copy to a different location with the same or different name</td><td>In the same directory cp creates an exact copy of the file, but target fileName must be different from the source fileName.
If copying the file to a different location, you can leave the fileName the same OR also specify a different fileName

cp will always create one or more target files.</td></tr>
<tr><td>mv</td><td>Moves a file from location to another OR
Renames a file</td><td>In the same directory, mv renames a source file to a different name.
If moving the source file to another location, you can preserve the fileName as is OR you can move and rename the fileName to a different name.

mv does NOT create new files
You can move multiple files to a different location with one move command
You CANNOT rename multiple files with one command</td></tr>
<tr><td>rm</td><td>Removes (deletes a file)
*** BE VERY CAREFUL WITH THIS COMMAND ***

rm removes or delete both files AND directories

Some files are protected with specific permissions and may require the use of 'sudo' (super-user do) prefix before the rm command to execute the command successfully.

Directories must be empty before removing them, meaning any files OR SUBDIRECTORIES must be removed first.

However, using the -R option will recursively remove any subdirectories and sub-subdirectories.</td><td>Example: Remove (delete) the hello.txt file
rm hello.txt

It is very possible to irrevocably take down a whole server with the rm command if you use the rm command:
- in the wrong directory location (e.g., root or top of the file structure)
- use wild-card characters such as '*' (meaning anything).

Best Practice:
Step 1: Make a backup of the file(s) you intend to delete. You can create a directory and copy those files into that backup directory before proceeding.

Step 2: Verify what you want to delete, by using the ls command. This will show you what files you are about to delete.

Step 3, recall the ls command you just used, and replace the ls with rm
Example:

ls -l *.txt *.file ## notice that we can list multiple items

-rw-r--r--@ 1 darlayoung staff 70 Jan 5 21:48 combined.file
-rw-r--r--@ 1 darlayoung staff 108 Jan 4 22:11 final.txt
-rw-r--r--@ 1 darlayoung staff 23 Jan 5 21:46 first.file
-rw-r--r--@ 1 darlayoung staff 24 Jan 5 21:47 second.file
-rw-r--r--@ 1 darlayoung staff 23 Jan 5 21:47 third.file

[Press Up-Arrow] key to recall the last command and replace the ls -l with an rm
$ rm *.txt *.file</td></tr>
<tr><td colspan="3">Section 3</td></tr>
<tr><td>pwd</td><td>print working directory

Displays the current location in the file structure.</td><td>Example:
$ pwd
/home/darla/Linux4Adies</td></tr>
<tr><td>tree</td><td>shows a graphic representation of the directory structure.

- Using tree by itself can generate a lot of output if there are many levels of subdirectories, so use the -L <number> to limit how many levels to display.
- The tree command will also display files; this can be very useful while learning about how to move/copy files around the Linux file structure.</td><td>Example: To see the directory structure one level below the /usr directory use the -L 1 to specify a Level of 1

$ tree -L1 /usr
/usr
├── bin
├── include
├── lib
├── libexec
├── local
├── sbin
├── share
└── x86_64-alpine-linux-musl</td></tr>
<tr><td>cd</td><td>change directory to a new location</td><td>Example:

cd /etc/local/bin

changes the current location to the /etc/local/bin folder</td></tr>
<tr><td>cd ..</td><td>The ".." is a short-hand representation meaning "previous directory". Using cd .. is a quick way to go up a level from where you are.</td><td>Suppose your current location is /etc/local/bin directory

Example:

cd ..

Will change your working directory location to /etc/local

cd ../..

Will change your working directory from /etc/local/bin to /etc since the '../..' moved the location up 2 levels.</td></tr>
<tr><td>mkdir</td><td>make directory : used to create a new directory or subdirectory.</td><td>Suppose your current location is /home/ada-dev
Example:

$ mkdir my-files

Creates a subdirectory named myfiles. The complete file path to that directory is
/home/ada-dev/myfiles</td></tr>
<tr><td>rmdir</td><td>remove directory : used to delete a directory and its subdirectories

The rm command can also remove directories, and many people prefer using the rm command to rmdir</td><td>Directories must be empty before this command will work.</td></tr>
<tr><td colspan="3">Section 4</td></tr>
<tr><td>history</td><td>Display history stack of all commands entered</td><td>Example: To display the history stack (aka "command buffer")

$ history
9 mkdir -p /home/darla/test-files /home/darla/pics-of-kittens
10 cd /home
11 tree
12 touch darla/pics-of-kittens/kittens01.png
13 touch darla/pics-of-kittens/kittens02.png darla/pics-of-kittens/kittens03.png darla/pics-of-kittens/kittens04.png
14 tree
15 touch darla/test-files/testfile01.txt
16 touch darla/test-files/testfile02.txt darla/test-files/testfile03.txt
17 tree
18 mkdir -p /mnt/darla/pics /mnt/darla/tests
19 pwd
20 tree /mnt
21 ls -l /home/darla/pics-of-kittens/* /mnt/darla/pics/
22 cp /home/darla/pics-of-kittens/* /mnt/darla/pics/
23 cp /home/darla/test-files/testfile03.txt /mnt/darla/tests/testfile03.renamed
24 tree /mnt</td></tr>
<tr><td>!</td><td>Each command in the history buffer has a unique line number.

If you want to execute a command from the history stack, type "!" immediately (no space) followed by the commands line number. No need to retype the whole command or copy/paste it.</td><td>Example:

Suppose you execute the history command to see previous commands. You can then execute a specific command by typing the '!' immediately followed by the line number of the command you want to execute again.

$ history
9 mkdir -p /home/darla/test-files /home/darla/pics-of-kittens
10 cd /home
11 tree
12 touch darla/pics-of-kittens/kittens01.png
13 touch darla/pics-of-kittens/kittens02.png darla/pics-of-kittens/kittens03.png darla/pics-of-kittens/kittens04.png
14 tree
15 touch darla/test-files/testfile01.txt
16 touch darla/test-files/testfile02.txt darla/test-files/testfile03.txt
17 tree
18 mkdir -p /mnt/darla/pics /mnt/darla/tests
19 pwd
20 tree /mnt
21 ls -l /home/darla/pics-of-kittens/* /mnt/darla/pics/
22 cp /home/darla/pics-of-kittens/* /mnt/darla/pics/
23 cp /home/darla/test-files/testfile03.txt /mnt/darla/tests/testfile03.renamed
24 tree /mnt

$!23 cp /home/darla/test-files/testfile03.txt /mnt/darla/tests/testfile03.renamed</td></tr>
<tr><td>grep</td><td>search using regular expression (grep="global regular expression print)</td><td>Example: Use grep to filter output from the history command to find only occurrences of the 'rm' command:

history | grep "rm"</td></tr>
<tr><td>|</td><td>Pipe symbol : pass output from one command into another for additional execution.</td><td>Examples:
history | grep "alias"
Searches the history buffer for any occurrence of the word 'alias'

ls -a | wc -l : takes output from the ls -a command and counts the number of lines returned</td></tr>
<tr><td>man</td><td>User manual page for a command</td><td>Example: Use the man command to see how to use the mkdir command:

man mkdir

Note: man can be very verbose and you may need to search through it for a particular answer.</td></tr>
<tr><td>--help</td><td>quick help (may not exist for every command, so use the man command instead)</td><td>Example: python3 –help (2 hyphens before 'help') generates the following information:

usage: /Library/Frameworks/Python.framework/Versions/3.13/Resources/Python.app/Contents/MacOS/Python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options (and corresponding environment variables):
-b : issue warnings about converting bytes/bytearray to str and comparing bytes/bytearray with str or bytes with int. (-bb: issue errors)
-B : don't write .pyc files on import; also PYTHONDONTWRITEBYTECODE=x
-c cmd : program passed in as string (terminates option list)
-d : turn on parser debugging output (for experts only, only works on debug builds); also PYTHONDEBUG=x
-E : ignore PYTHON* environment variables (such as PYTHONPATH)
-h : print this help message and exit (also -? or –help)</td></tr>
<tr><td>alias</td><td>This command allows you to equate a word or abbreviation to represent a command.

Typing 'alias' by itself will display all aliases already available</td><td>Example: Instead of always typing the command "ls -latr", you can instead use just 'll' to represent it (a lot less typing):

$ alias ll="ls -latr"

From this point onwards, typing "ll" will produce the same results as "ls -latr".
By the way, using 'll' is such a common alias that many Linux version have it built in already.

Here is another one that will come in handy if you use the history command a lot:

Example: Suppose you want to find a specific word or command that you've used before … let's say the 'touch' command for instance. Ordinarily, you would use both history and grep commands together to find all occurrences of 'touch'.

$ history | grep 'touch'

Create an alias instead:

$ alias hist="history | grep "

Make sure to leave a space after the word 'grep'. To use this alias, you can type ...and every command with the word 'touch' in it will be returned.

$ hist touch
5 touch hello-world.txt
6 touch myfirstFile.txt
7 touch clear
9 hist touch</td></tr>
<tr><td colspan="3">Section 5</td></tr>
<tr><td>></td><td>Redirect : create / over-write</td><td>A single '>' redirects output of a command from the screen to a file that you specify. If that file does not exist, then it will create the file; if the file does exist, any existing content is overwritten with the new output every time the command is executed.</td></tr>
<tr><td>>></td><td>Redirect : create / append</td><td>Double '>>' redirects output of a command from the screen to a file that you specify. If that file does not exist, then it will create the file; if the file does exist, the new output is appended to the file, and any existing content is preserved.
Some log files are set up this way so you can see successive transactions over a period of time.</td></tr>
<tr><td>echo "text"</td><td>displays on the screen whatever was typed after the word "echo": Use in combination with redirections symbols above to create / append to a file instead of the screen</td><td>Example:

echo "Hello World" > hello.txt</td></tr>
<tr><td colspan="3">Section 6</td></tr>
<tr><td>~</td><td>This is the tilde characterand in Linux is a shorthand that represents the home directory.</td><td>$ cd /home/<yourUserName> will take you to your home directory. We know that just typing 'cd' and pressing [Enter] will also take us home.

But what if you're creating a script (e.g., an alias, for example)?

We can use the tilde character to replace the /home/<username>

Example:

/home/darla/splunk.conf is the same as ~/splunk.conf

The real value in using the '~' character is that you don't need to hard-code a home directory name if you're writing a script that might be used by many different users.

The '~' character behaves like a system variable. In fact, you can even type 'echo ~' and it will return the name of the home directory ite</td></tr>
<tr><td>file paths and http</td><td>File path is essentially the roadmap to accessing a file. In some cases, files are easily accessible because they are located near the top of a root directory. In other case, however, it may be more complicated because the file may be nested several levels deep in subdirectories.

Simple Rule: Subdirectory levels are always separated by a forward slash "/". Whenever you see a forward slash, that indicates another level further away from the root directory.

An analogy for file path is like the turn-by-turn instructions a GPS gives you when going to a location:

"Go straight past these lights, and turn left at the next intersection onto Vine street. Then turn right on Route 53."

File paths help you and your applications locate files in a Linux server. And actually you are already very Familiar with file paths!

For example:
This is the URL for a great resource for learning Linux:
https://youtu.be/MnY0K-3_Fjk?si=m5qhzQI0Bv4_jZJ</td><td>Example:

https://drive.google.com/drive/folders/1QH9TVKX3gpqHrKJWnUhsfwi

URLs are actually file paths of locations on a server.

https:// → means web traffic and what follows the "https://" is a server

drive.google.com → this is the DNS (domain name service) entry for the servers in Google's network

/drive → refers to a directory on that server name "drive"

/folders → this is another subdirectory under the drive folder above

….and the remainder is a reference to a specific resource within the server that contains my docs for this course.</td></tr>
<tr><td>nano</td><td>nano is a simple full-screen editor that comes installed on almost all Linux distributions</td><td>Example: nano hello-world.txt

NOTE: This version of Ubuntu in the Play-With-Docker site does not have nano installed.</td></tr>
<tr><td>vi (vim)</td><td>vi or vim is a built-in line editor installed on all Linux distributions.

Line editors are bit more difficult to use because you edit a file, line-by-line, and know the commands to do it.

Most people prefer to use full-screen editors and this is part of the reason why many people are afraid of Linux.

However, recently there's been a renewed interest in vi, vim, and nvim.
Vi : first and oldest version of visual editor

vim : stands for vi-improved

nvim : refers to NeoVim, which is the most popular line editor among "power programmers" with several extensions that allow you to customize the environment to make it more user-friendly and efficient.</td><td>Example: vi hello-world.txt</td></tr>
</table>