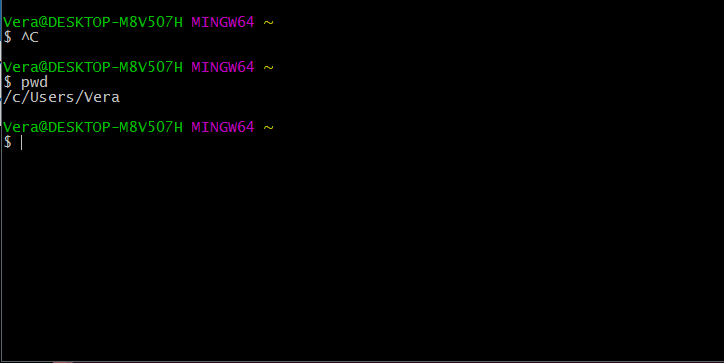
Linux Terminal Tutorial Episode 1: Back to Basics

/ shows your present working directory  
pwd print working directory. shows which directory you’re currently in  


ls means list. It will show you all the visible files and directories within route.  
ls –a would list out the hidden files as well  
cd change directory  
~ represents your home directory  
mkdir make directory  
cd .. would take you back to the last directory  
cd ~ takes you back home directory  
cd / takes you back to root  
rmdir nameofcdirectory removes directory  
filename to open up a file(?)  
rm filename remove file  
rm –R removes everything in directory with the directory itself

cmd line is case sensitive, so cd documents isn’t the same as cd Documents.  
cd only works backwards when you’re returning to root or home.  
You cannot cd into folders that are not child folders of the current working directory. You’ll have to type cd /Users/Vera/Documents for example.  
You can only remove directory with rmdir, if the directory is empty. If not, use rm –R.

CODECADEMY

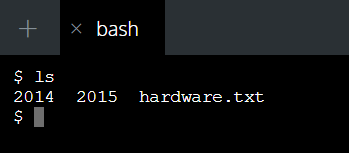
The command line is a text interface for your computer. It's a program that takes in commands, which it passes on to the computer's operating system to run.

From the command line, you can navigate through files and folders on your computer, just as you would with Finder on Mac OS or Windows Explorer on Windows. The difference is that the command line is fully text-based.

The advantage of using the command line is its power. You can run programs, write scripts to automate common tasks, and combine simple commands to handle difficult tasks - making it an important programming tool.

This course is for unix-based systems such as Linux and Mac OS X. An appendix of all commands taught in this course is available here.

To access the command line, we use a terminal emulator, often just called the terminal.  
In the terminal, after the $ type:  
ls  
and press Enter. Be sure to type the letter l as in "lemon" and not the number 1.  
You should see three items print out below the command.



1. In the terminal, first you see $. This is called a shell prompt. It appears when the terminal is ready to accept a command.
2. When you type ls, the command line looks at the folder you are in, and then "lists" the files and folders inside it. The directories 2014, 2015, and the file hardware.txt are the contents of the current directory.

ls is an example of a command, a directive to the computer to perform a specific task.

Instructions

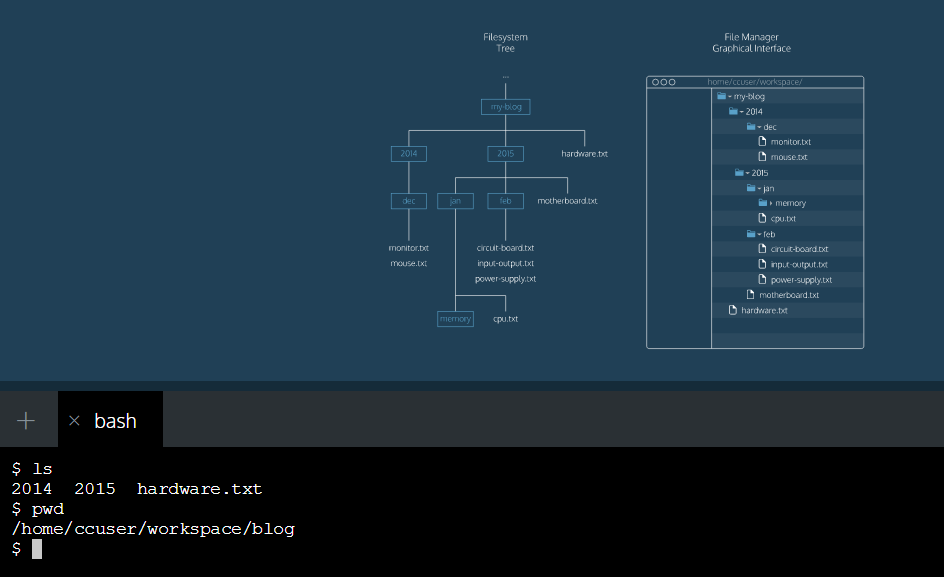
When using the command line, we refer to folders as directories. Files and directories on your computer are organized into a filesystem.

A filesystem organizes a computer's files and directories into a tree structure:

1. The first directory in the filesystem is the root directory. It is the parent of all other directories and files in the filesystem.
2. Each parent directory can contain more child directories and files. Here blog/ is the parent of 2014/, 2015/, and hardware.txt.
3. Each directory can contain more files and child directories. The parent-child relationship continues as long as directories and files are nested.

You're probably already familiar with this tree structure - Mac Finder and Windows Explorer represent the filesystem as trees as well.

1. Let's see how to navigate the filesystem from the command line. In the terminal, after the shell prompt, type  
   pwd  
   and press Enter. We'll explain this in the next exercise.



pwd stands for "print working directory". It outputs the name of the directory you are currently in, called the working directory.

Here the working directory is blog/. In Codecademy courses, your working directory is usually inside the home/ccuser/workspace/ directory.

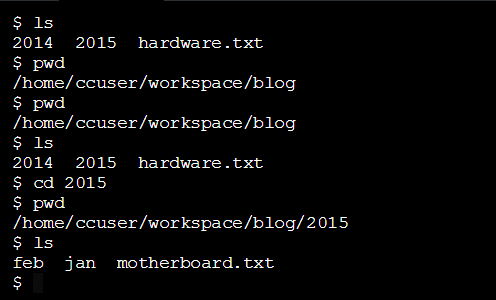
Together with ls, the pwd command is useful to show where you are in the filesystem.

1. Let's continue with more commands. In the terminal, print the working directory.
2. List all files and directories in the working directory.
3. Then type

cd 2015

Again, print the new current working directory.

List all files and directories in the working directory.



1. cd stands for "change directory". Just as you would click on a folder in Windows Explorer or Finder, cd switches you into the directory you specify. In other words, cd changes the working directory.
2. The directory we change into is 2015. When a file, directory or program is passed into a command, it is called an argument. Here the 2015 directory is an argument for the cd command.

The cd command takes a directory name as an argument, and switches into that directory.

Instructions:

1. Then type

cd jan/memory/  
Print the working directory to see the new location.

1. Then type

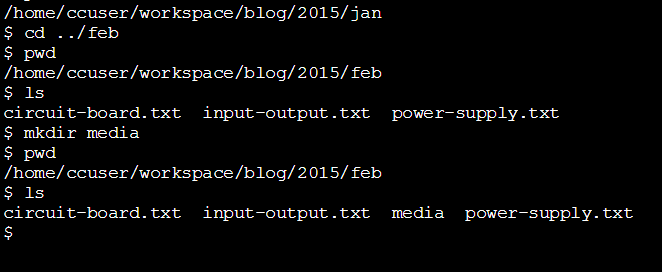
cd ..

Print the working directory again to see the new location.

To navigate directly to a directory, use cd with the directory's path as an argument. Here, cd jan/memory/ command navigates directly to the jan/memory directory.  
$ cd ..  
To move up one directory, use cd ... Here, cd .. navigates up from jan/memory/ to jan/.

Instructions:

1. Change the directory to the 2015/feb/ directory  
   cd ../feb  
   List all files and directories in the working directory.
2. Type  
   mkdir media  
   Again, list all files and directories in the working directory. You'll see that there is now a new directory named media/.



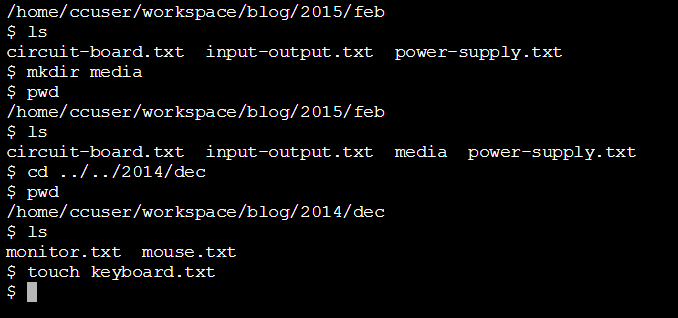
The mkdir command stands for "make directory". It takes in a directory name as an argument, and then creates a new directory in the current working directory.

Here we used mkdir to create a new directory named media/ inside the feb/ directory.

Instructions:

1. Navigate to the 2014/dec/ directory.  
cd ../../2014/dec  
List all files and directories in the working directory.

2. Then type  
touch keyboard.txt  
Again, list all files and directories in the working directory. You'll see that there is now a new file named keyboard.txt.



touch keyboard.txt

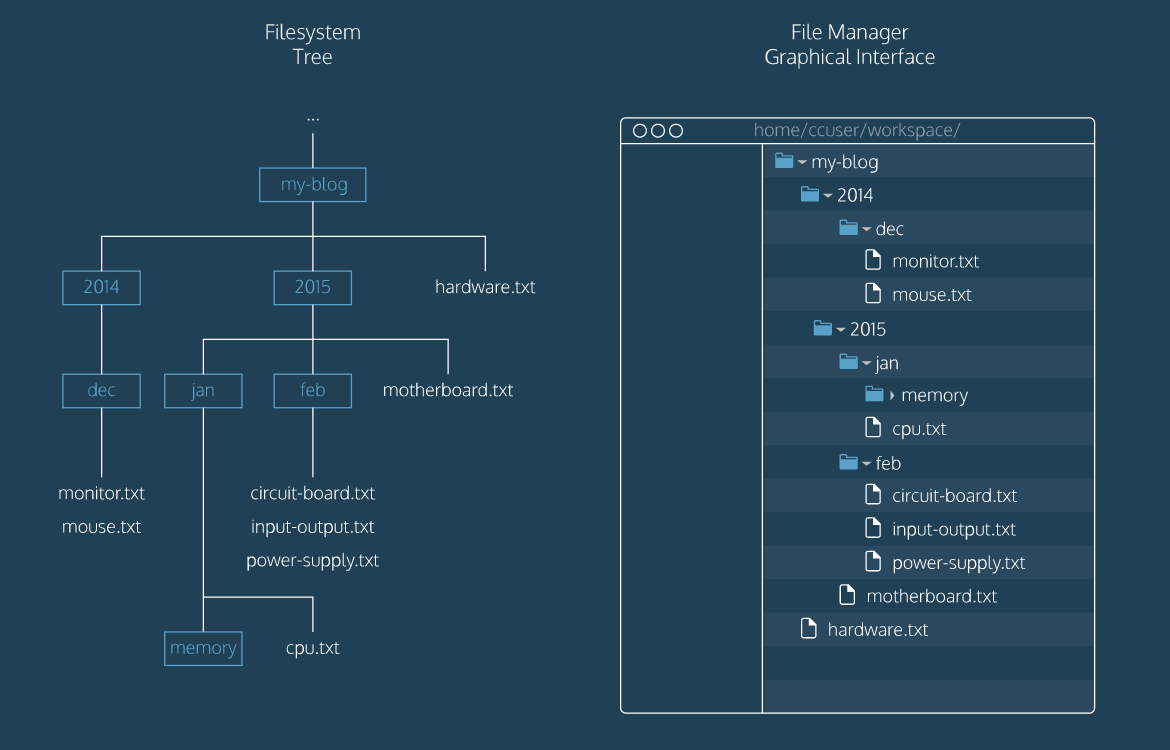
The touch command creates a new file inside the working directory. It takes in a filename as an argument, and then creates an empty file in the current working directory.  
Here we used touch to create a new file named keyboard.txt inside the 2014/dec/ directory.

Instructions:  
The commands we've covered so far are commonly used to navigate the filesystem. There are more commands you can use to master the command line, and we'll cover them in the next lessons.  
Let's summarize what we've done so far.

Summary:

Congratulations! You've learned five commands commonly used to navigate the filesystem from the command line. What can we generalize so far?

* The command line is a text interface for the computer's operating system. To access the command line, we use the terminal.
* A filesystem organizes a computer's files and directories into a tree structure. It starts with the root directory. Each parent directory can contain more child directories and files.
* From the command line, you can navigate through files and folders on your computer:
  + pwd outputs the name of the current working directory.
  + ls lists all files and directories in the working directory.
  + cd switches you into the directory you specify.
  + mkdir creates a new directory in the working directory.



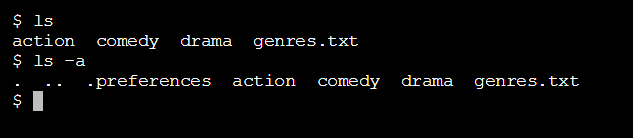
MANIPULATION:

So far we've used the command line to navigate the filesystem.

We can do more with the command line to view directories and files. We can also use the command line to copy, move, and remove files and directories. Let's see how to do this.

Instructions:

1. In the terminal after the shell prompt, type  
   ls
2. Then type  
   ls -a

Do you see the differences between the outputs of both commands? Click Next to learn how this works.  


1. The ls command lists all files and directories in the working directory.
2. The -a modifies the behavior of the ls command to also list the files and directories starting with a dot (.). Files started with a dot are hidden, and don't appear when using ls alone.

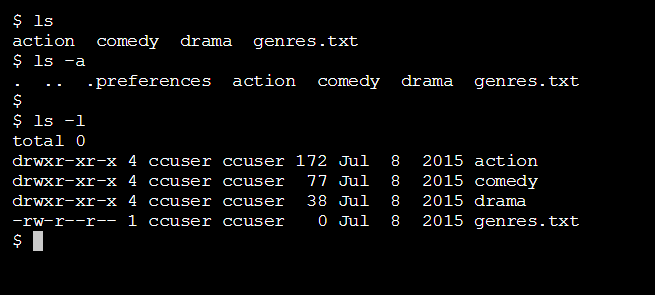
The -a is called an option. Options modify the behavior of commands. Here we used ls -a to display the contents of the working directory in more detail.

In addition to -a, the ls command has several more options. Here are three common options:

* -a - lists all contents, including hidden files and directories
* -l - lists all contents of a directory in long format
* -t - order files and directories by the time they were last modified.

Instructions:

1. In the terminal, type  
   ls –l  
   Click Next to find out what these columns mean.



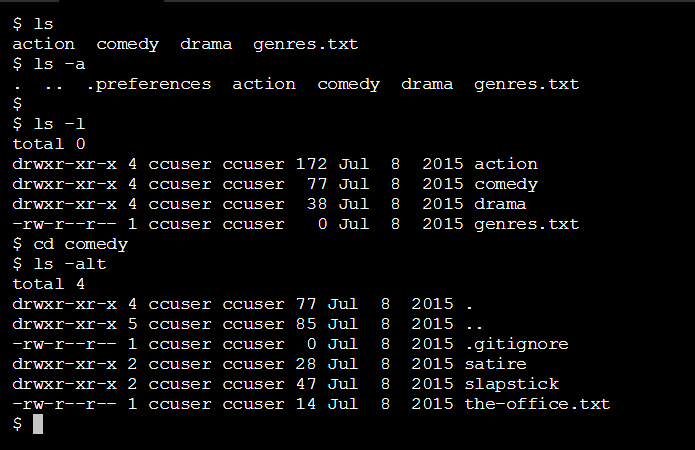
The -l option lists files and directories as a table. Here there are four rows, with seven columns separated by spaces. Here's what each column means:

1. Access rights. These are actions that are permitted on a file or directory.
2. Number of hard links. This number counts the number of child directories and files. This number includes the parent directory link (..) and current directory link (.).
3. The username of the file's owner. Here the username is cc.
4. The name of the group that owns the file. Here the group name is eng.
5. The size of the file in bytes.
6. The date & time that the file was last modified.
7. The name of the file or directory.

Instructions:

1. Let's try out another option for the ls command.  
   Navigate to the comedy/ directory.
2. Then type

ls –alt

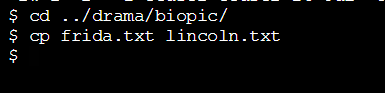


The -t option orders files and directories by the time they were last modified.  
In addition to using each option separately, like ls -a or ls -l, multiple options can be used together, like ls -alt.  
Here, ls -alt lists all contents, including hidden files and directories, in long format, ordered by the date and time they were last modified.

Instructions:

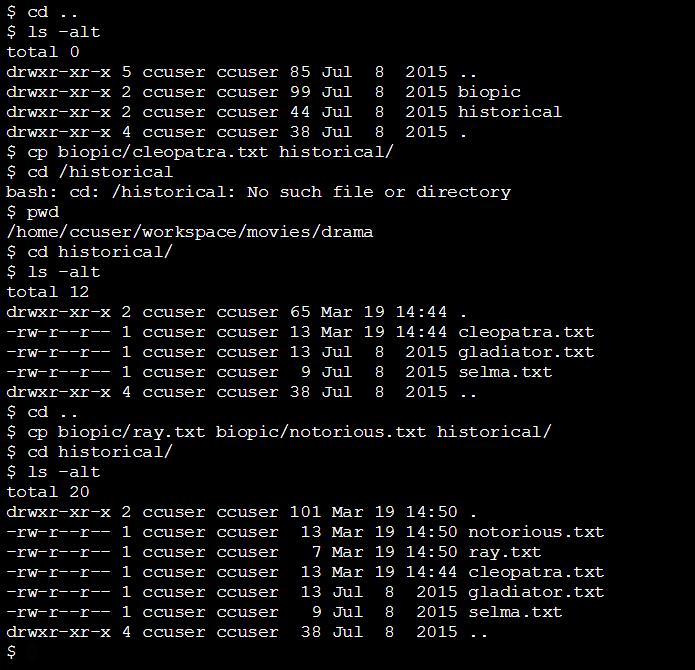
1. Let's move on to copying, moving, and removing files and directories from the command line.  
   Navigate to the drama/biopic/ directory.  
   cd ../drama/biopic/  
   List all files and directories in the working directory.
2. Then type  
   cp frida.txt lincoln.txt

Click Next to learn about this command.



cp frida.txt lincoln.txt  
The cp command copies files or directories. Here, we copy the contents of frida.txt into lincoln.txt.

1. Navigate to the drama/ directory.  
   cd ..  
   List all files and directories in the working directory.
2. Type  
   cp biopic/cleopatra.txt historical/
3. Navigate to the historical/ directory.  
   List all files and directories in the working directory. You should see a new copy of cleopatra.txt in this directory.
4. Here's one more way to use cp.  
   Navigate up one directory from drama/historical/ to drama/. (Here's a hint on how to do this.)
5. Then type  
   cp biopic/ray.txt biopic/notorious.txt historical/
6. Change directories into historical/.  
   List all files and directories in the working directory. You should see a new copy of ray.txt and notorious.txt in this directory.



To copy a file into a directory, use cp with the source file as the first argument and the destination directory as the second argument. Here, we copy the file biopic/cleopatra.txt and place it in the historical/ directory.

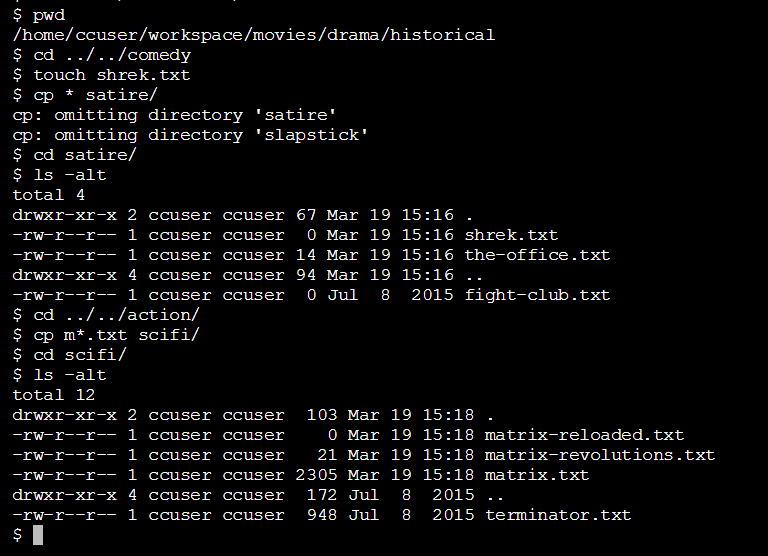
cp biopic/ray.txt biopic/notorious.txt historical/

To copy multiple files into a directory, use cp with a list of source files as the first arguments, and the destination directory as the last argument. Here, we copy the files biopic/ray.txt and biopic/notorious.txt into the historical/ directory.

Instructions:

1. Let's look at two more ways to use cp.  
   Navigate to the comedy/ directory.  
   cd ../../comedy
2. In this directory, create a new file named shrek.txt. (Here's a hint on how to do this.)
3. Then type  
   cp \* satire/
4. Navigate to the satire/ directory.  
   List all files and directories in the working directory.  
   You should see a copy of the files the-office.txt and shrek.txt in this directory. We'll explain how this works in the next exercise.
5. Here's another way to use cp.  
   Navigate to the action/ directory. Type  
   cd ../../action/  
   Here we navigate up two directories, and then into the action/ directory.
6. Type  
   cp m\*.txt scifi/
7. Change directories into scifi/.

List all files and directories in the working directory.  
You should see a copy of all text files starting with "m": matrix.txt, matrix-reloaded.txt, and matrix-revolutions.txt.

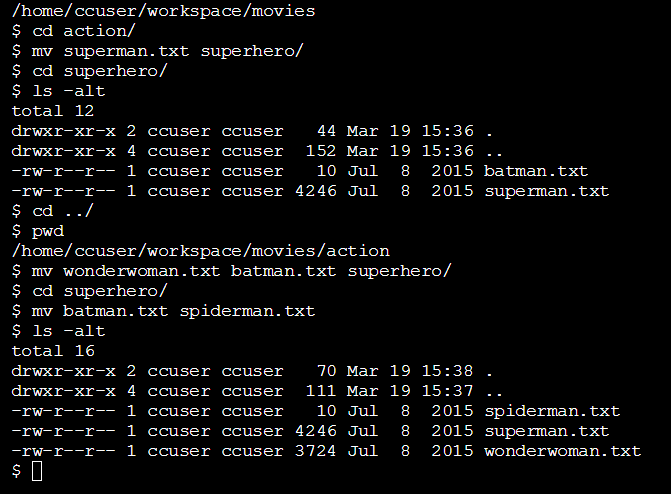


cp \* satire/  
In addition to using filenames as arguments, we can use special characters like \* to select groups of files. These special characters are called wildcards. The \* selects all files in the working directory, so here we use cp to copy all files into the satire/ directory.

cp m\*.txt scifi/  
Here, m\*.txt selects all files in the working directory starting with "m" and ending with ".txt", and copies them to scifi/.

Instructions:

1. In addition to copying files, we can move files from the command line.  
   Change directories into the action/ directory.  
   cd ../
2. Type  
   mv superman.txt superhero/
3. Navigate to the superhero/ directory.  
   List all files and directories in the working directory. You should see superman.txt in it.
4. Here's another way to use mv.  
   Navigate up one directory from action/superhero/ to action/.
5. Then type  
   mv wonderwoman.txt batman.txt superhero/
6. Navigate to superhero/ again.  
   List all files and directories in the working directory. You should see wonderwoman.txt and batman.txt in it.
7. Here's one more way to use mv.  
   Type  
   mv batman.txt spiderman.txt
8. List all files and directories in the working directory. You should see the file batman.txt has been renamed as spiderman.txt.



The mv command moves files. It's similar to cp in its usage.

mv superman.txt superhero/

To move a file into a directory, use mv with the source file as the first argument and the destination directory as the second argument. Here we move superman.txt into superhero/.

mv wonderwoman.txt batman.txt superhero/

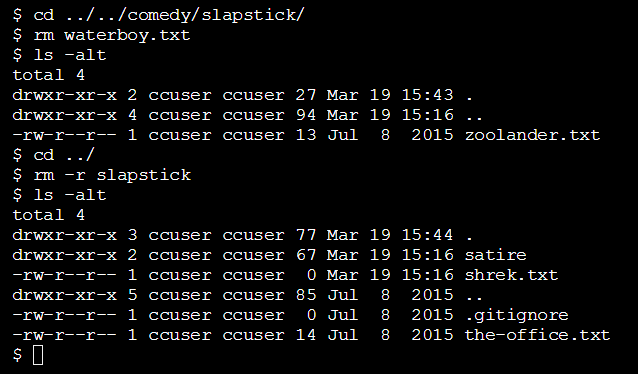
To move multiple files into a directory, use mv with a list of source files as the first arguments, and the destination directory as the last argument. Here, we move wonderwoman.txt and batman.txt into superhero/.

mv batman.txt spiderman.txt

To rename a file, use mv with the old file as the first argument and the new file as the second argument. By moving batman.txt into spiderman.txt, we rename the file as spiderman.txt.

Instructions:

1. Change directory to comedy/slapstick.  
   List all files and directories in the working directory.  
   cd ../../comedy/slapstick/
2. Type  
   rm waterboy.txt
3. List all files and directories in the working directory. You should see that waterboy.txt has been removed.
4. Navigate up one directory from comedy/slapstick/ to comedy/.
5. Type  
   rm -r slapstick
6. List all files and directories in the working directory. You should see that the slapstick/ directory has been removed.



rm waterboy.txt

The rm command deletes files and directories. Here we remove the file waterboy.txt from the filesystem.

rm -r comedy

The -r is an option that modifies the behavior of the rm command. The -r stands for "recursive," and it's used to delete a directory and all of its child directories.  
Be careful when you use rm! It deletes files and directories permanently. There isn't an undelete command, so once you delete a file or directory with rm, it's gone.  
The commands we've covered so far are commonly used to view and change the filesystem.  
Let's summarize what we've done so far.

Congratulations! You learned how to use the command line to view and manipulate the filesystem. What can we generalize so far?

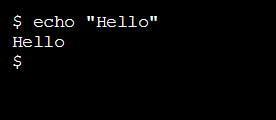
* Options modify the behavior of commands:
* ls -a lists all contents of a directory, including hidden files and directories
* ls -l lists all contents in long format
* ls -t orders files and directories by the time they were last modified
* Multiple options can be used together, like ls -alt
* From the command line, you can also copy, move, and remove files and directories:
* cp copies files
* mv moves and renames files
* rm removes files
* rm -r removes directories
* Wildcards are useful for selecting groups of files and directories

REDIRECTIONS:

Up until now, we have run commands in the command line and received a stream of output in the terminal. In this lesson, we'll focus on input and output (I/O) redirection.  
Through redirection you can direct the input and output of a command to and from other files and programs, and chain commands together in a pipeline. Let's try it out.

Instructions:

1. Let's begin by taking a closer look at input and output.  
   In the terminal, after the shell prompt, type  
   echo "Hello"



What happens when you type this command?

$ echo "Hello"  
Hello

The echo command accepts the string "Hello" as standard input, and echoes the string "Hello" back to the terminal as standard output.

Let's learn more about standard input, standard output, and standard error:

1. standard input, abbreviated as stdin, is information inputted into the terminal through the keyboard or input device.
2. standard output, abbreviated as stdout, is the information outputted after a process is run.
3. standard error, abbreviated as stderr, is an error message outputted by a failed process.

Redirection reroutes standard input, standard output, and standard error to or from a different location.

Instructions:

1. Now that you are familiar with standard input, standard output, and standard error, let's try our first redirect.  
   In the terminal, type  
   echo "Hello" > hello.txt
2. Then type  
   cat hello.txt

How does redirection work?  
$ echo "Hello" > hello.txt

The > command redirects the standard output to a file. Here, "Hello" is entered as the standard input. The standard output "Hello" is redirected by > to the file hello.txt.

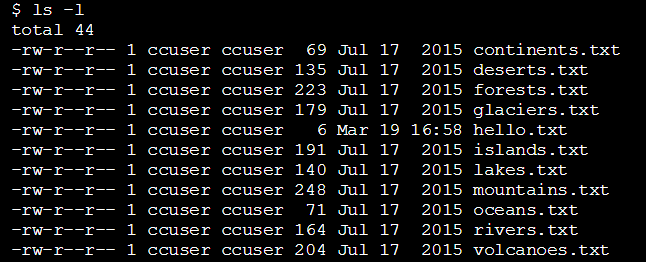
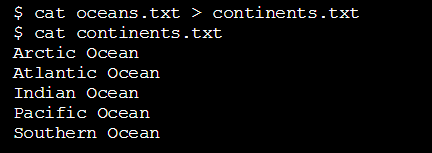
$ cat hello.txt

The cat command outputs the contents of a file to the terminal. When you type cat hello.txt, the contents of hello.txt are displayed.

Instructions:

1. Let's practice redirection some more. In the terminal, type  
   ls –l  
   This is the filesystem we'll work with.
2. Then type  
   cat oceans.txt > continents.txt
3. Use cat to view the contents of continents.txt.

Notice that we only see oceans as output.

$ cat oceans.txt > continents.txt

> takes the standard output of the command on the left, and redirects it to the file on the right. Here the standard output of cat oceans.txt is redirected to continents.txt.

Note that > overwrites all original content in continents.txt. When you view the output data by typing cat on continents.txt, you will see only the contents of oceans.txt.

Instructions:

1. Type  
   cat glaciers.txt >> rivers.txt
2. Use cat to view the contents of rivers.txt.

Notice that we see both rivers and glaciers as output.

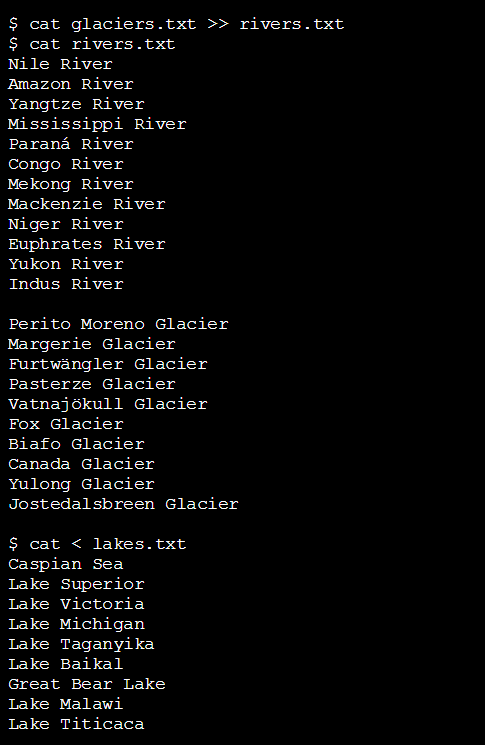
$ cat glaciers.txt >> rivers.txt

>> takes the standard output of the command on the left and appends (adds) it to the file on the right. You can view the output data of the file with cat and the filename.

Here, the the output data of rivers.txt will contain the original contents of rivers.txt with the content of glaciers.txt appended to it.

Instructions:

1. In the terminal type  
   cat < lakes.txt

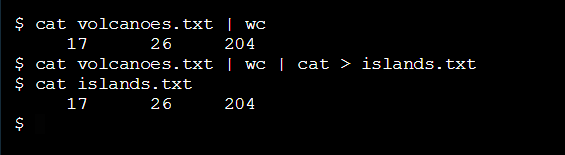


$ cat < lakes.txt

< takes the standard input from the file on the right and inputs it into the program on the left. Here, lakes.txt is the standard input for the cat command. The standard output appears in the terminal.

Instructions:

1. Let's try some more redirection commands. In the terminal, type  
   cat volcanoes.txt | wc
2. Type  
   cat volcanoes.txt | wc | cat > islands.txt
3. Use cat to output the contents of islands.txt. Notice that three numbers appear as output.



$ cat volcanoes.txt | wc

| is a "pipe". The | takes the standard output of the command on the left, and pipes it as standard input to the command on the right. You can think of this as "command to command" redirection.

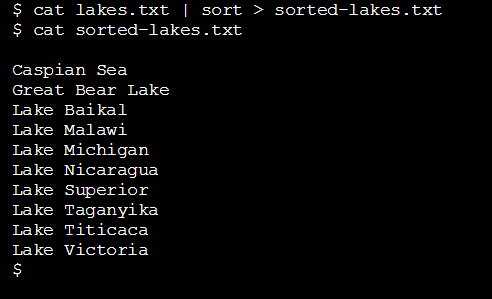
Here the output of cat volcanoes.txt is the standard input of wc. in turn, the wc command outputs the number of lines, words, and characters in volcanoes.txt, respectively.

$ cat volcanoes.txt | wc | cat > islands.txt

Multiple |s can be chained together. Here the standard output of cat volcanoes.txt is "piped" to the wc command. The standard output of wc is then "piped" to cat. Finally, the standard output of cat is redirected to islands.txt.

Instructions:

1. A few commands are particularly powerful when combined with redirection. Let's try them out.  
   First, use cat to output the contents of lakes.txt.
2. We'll begin with sort. In the terminal type  
   sort lakes.txt
3. Then type  
   cat lakes.txt | sort > sorted-lakes.txt
4. Use cat to output the contents of sorted-lakes.txt.

$ sort lakes.txt

sort takes the standard input and orders it alphabetically for the standard output. Here, the lakes in sort lakes.txt are listed in alphabetical order.

$ cat lakes.txt | sort > sorted-lakes.txt

Here, the command takes the standard output from cat lakes.txt and "pipes" it to sort. The standard output of sort is redirected to sorted-lakes.txt.  
You can view the output data by typing cat on the file sorted-lakes.txt.

Instructions:

1. Use cat to output the contents of deserts.txt.
2. Type  
   uniq deserts.txt
3. In the terminal type  
   sort deserts.txt | uniq
4. Then type  
   sort deserts.txt | uniq > uniq-deserts.txt
5. Use cat to output the contents of uniq-deserts.txt.


$ uniq deserts.txt

uniq stands for "unique" and filters out adjacent, duplicate lines in a file. Here uniq deserts.txt filters out duplicates of "Sahara Desert", because the duplicate of 'Sahara Desert' directly follows the previous instance. The "Kalahari Desert" duplicates are not adjacent, and thus remain.

$ sort deserts.txt | uniq

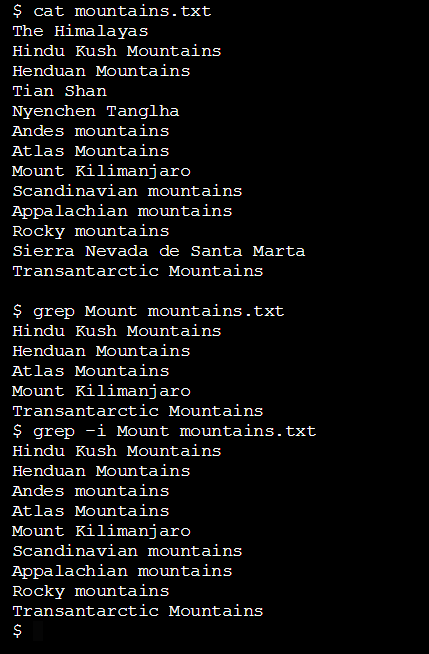
A more effective way to call uniq is to call sort to alphabetize a file, and "pipe" the standard output to uniq. Here by piping sort deserts.txt to uniq, all duplicate lines are alphabetized (and thereby made adjacent) and filtered out.

sort deserts.txt | uniq > uniq-deserts.txt

Here we simply send filtered contents to uniq-deserts.txt, which you can view with the cat command.

Instructions:

1. Use cat to output the contents of mountains.txt.
2. In the terminal type  
   grep Mount mountains.txt
3. Then type  
   grep -i Mount mountains.txt



$ grep Mount mountains.txt

grep stands for "global regular expression print". It searches files for lines that match a pattern and returns the results. It is also case sensitive. Here, grep searches for "Mount" in mountains.txt.

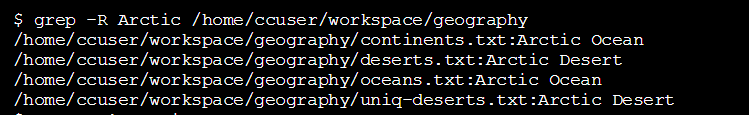
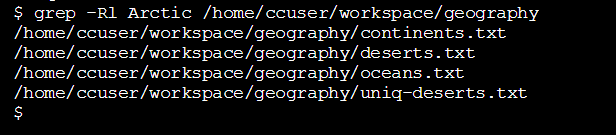
$ grep -i Mount mountains.txt

grep -i enables the command to be case insensitive. Here, grep searches for capital or lowercase strings that match Mount in mountains.txt.

The above commands are a great way to get started with grep. If you are familiar with regular expressions, you can use regular expressions to search for patterns in files.

Instructions:

1. grep can also be used to search within a directory. In the terminal, type  
   grep -R Arctic /home/ccuser/workspace/geography
2. Then type  
   grep -Rl Arctic /home/ccuser/workspace/geography

$ grep -R Arctic /home/ccuser/workspace/geography

grep -R searches all files in a directory and outputs filenames and lines containing matched results. -R stands for "recursive". Here grep -R searches the /home/ccuser/workspace/geography directory for the string "Arctic" and outputs filenames and lines with matched results.

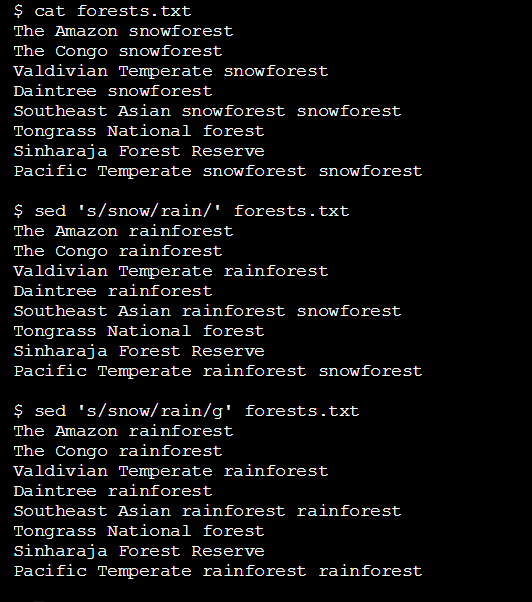
$ grep -Rl Arctic /home/ccuser/workspace/geography

grep -Rl searches all files in a directory and outputs only filenames with matched results. -R stands for "recursive" and l stands for "files with matches". Here grep -Rl searches the /home/ccuser/workspace/geography directory for the string "Arctic" and outputs filenames with matched results.

Instructions:

1. Use cat to display the contents of forests.txt

2. In the terminal, type  
sed 's/snow/rain/' forests.txt

3. Now type  
sed 's/snow/rain/g' forests.txt  


$ sed 's/snow/rain/' forests.txt

sed stands for "stream editor". It accepts standard input and modifies it based on an expression, before displaying it as output data. It is similar to "find and replace".

Let's look at the expression 's/snow/rain/':

* s: stands for "substitution". it is always used when using sed for substitution.
* snow: the search string, the text to find.
* rain: the replacement string, the text to add in place.

In this case, sed searches forests.txt for the word "snow" and replaces it with "rain." Importantly, the above command will only replace the first instance of "snow" on a line.

$ sed 's/snow/rain/g' forests.txt

The above command uses the g expression, meaning "global". Here sed searches forests.txt for the word "snow" and replaces it with "rain", globally. All instances of "snow" on a line will be turned to "rain".

Summary:

Congratulations! You learned how to use the command line to redirect standard input and standard output. What can we generalize so far?

* Redirection reroutes standard input, standard output, and standard error.
* The common redirection commands are:
* redirects standard output of a command to a file, overwriting previous content.
* >> redirects standard output of a command to a file, appending new content to old content.
* < redirects standard input to a command.
* | redirects standard output of a command to another command.
* A number of other commands are powerful when combined with redirection commands:
* sort: sorts lines of text alphabetically.
* uniq: filters duplicate, adjacent lines of text.
* grep: searches for a text pattern and outputs it.
* sed : searches for a text pattern, modifies it, and outputs it.

ENVIRONMENT:

Each time we launch the terminal application, it creates a new session. The session immediately loads settings and preferences that make up the command line environment.

We can configure the environment to support the commands and programs we create. This enables us to customize greetings and command aliases, and create variables to share across commands and programs.

You can reference the filesystem for this lesson here.

Instructions:

1. We'll begin by learning to use a simple, command line text editor called nano.  
   In the terminal, type  
   nano hello.txt  
   This will open the nano text editor.
2. In nano, at the top of the window, type

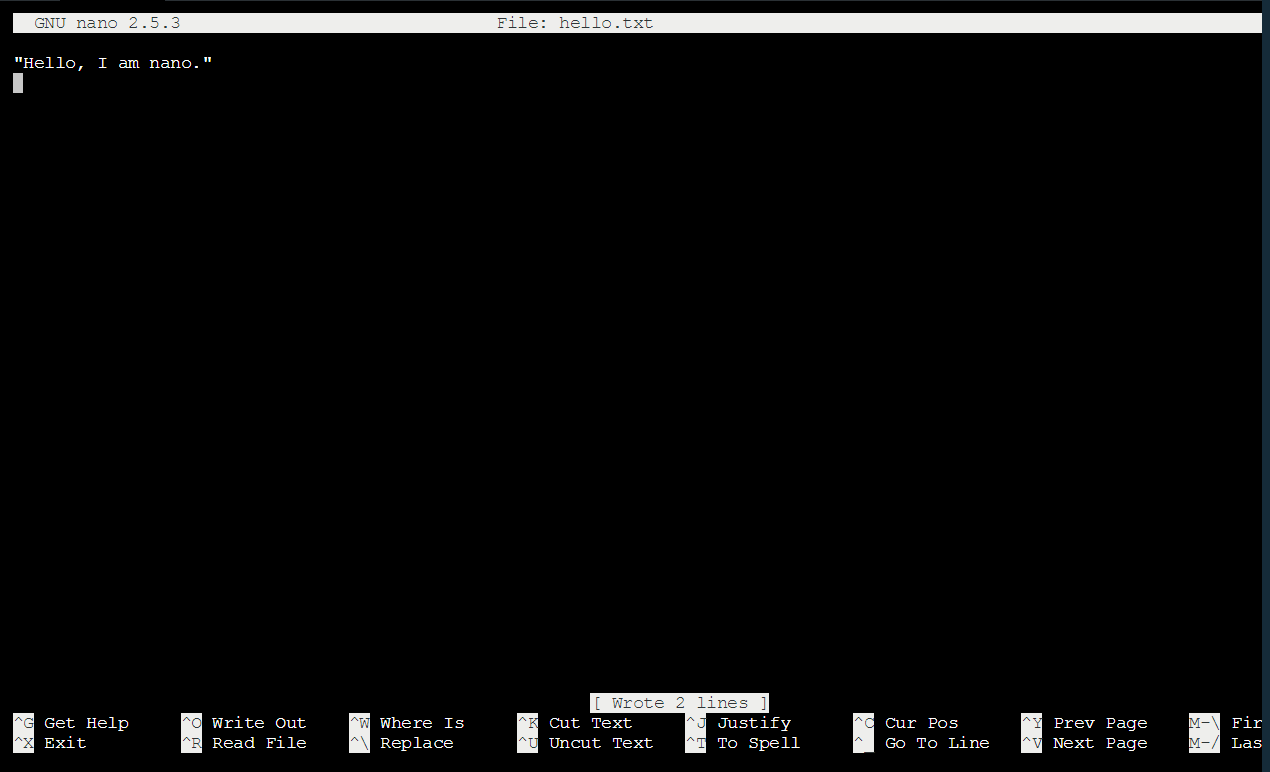
"Hello, I am nano."

Using the menu at the bottom of the terminal for reference, type Ctrl + O (the letter, not the number) to save the file. This is the letter "O", not the number zero.

Press Enter, when prompted about the filename to write.

Then type Ctrl + X to exit nano.

Finally, type clear to clear the terminal window. The command prompt should now be at the top of the window.



Nice. You just edited a file in the nano text editor. How does it work?

$ nano hello.txt

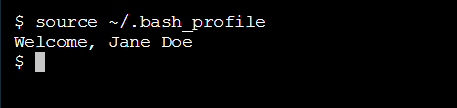
nano is a command line text editor. It works just like a desktop text editor like TextEdit or Notepad, except that it is accessible from the command line and only accepts keyboard input.

1. The command nano hello.txt opens a new text file named hello.txt in the nano text editor.
2. "Hello, I am nano" is a text string entered in nano through the cursor.
3. The menu of keyboard commands at the bottom of the window allow us to save changes to hello.txt and exit nano. The ^ stands for the Ctrl key.
4. Ctrl + O saves a file. 'O' stands for output.
5. Ctrl + X exits the nano program. 'X' stands for exit.
6. Ctrl + G opens a help menu.
7. clear clears the terminal window, moving the command prompt to the top of the screen.

In this lesson, we'll use nano to implement changes to the environment. You can learn more about nano here.

Instructions:

1. Now that you are familiar with editing text in nano, let's create a file to store environment settings.  
   In the terminal, type  
   nano ~/.bash\_profile  
   This opens up a new file in nano.
2. In ~/.bash\_profile, at the top of the file, type  
   echo "Welcome, Jane Doe"  
   You can use your name in place of "Jane Doe".  
   Type Ctrl + O to save the file.  
   Press Enter to write the filename.  
   Type Ctrl + X to exit.  
   Finally, type clear to clear the terminal window.
3. In the terminal, type  
   source ~/.bash\_profile  
   You should see the greeting you entered.



You created a file in nano called ~/.bash\_profile and added a greeting. How does this work?

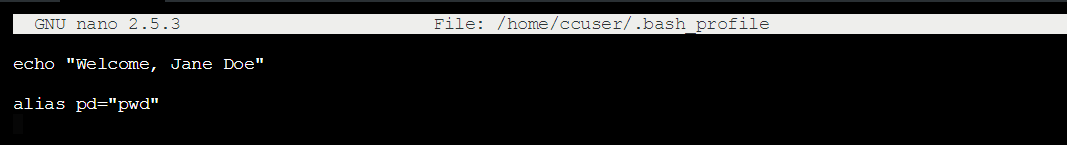
$ nano ~/.bash\_profile

~/.bash\_profile is the name of file used to store environment settings. It is commonly called the "bash profile". When a session starts, it will load the contents of the bash profile before executing commands.

* The ~ represents the user's home directory.
* The . indicates a hidden file.
* The name ~/.bash\_profile is important, since this is how the command line recognizes the bash profile.

1. The command nano ~/.bash\_profile opens up ~/.bash\_profile in nano.
2. The text echo "Welcome, Jane Doe" creates a greeting in the bash profile, which is saved. It tells the command line to echo the string "Welcome, Jane Doe" when a terminal session begins.
3. The command source ~/.bash\_profile activates the changes in ~/.bash\_profile for the current session. Instead of closing the terminal and needing to start a new session, source makes the changes available right away in the session we are in.

Instructions:

1. Now that we know what bash profile is, let's continue configuring the environment by adding command aliases.  
   Open ~/.bash\_profile in nano.
2. In ~/.bash\_profile, beneath the greeting you created, type  
   alias pd="pwd"  
   Save the file.  
   Press Enter to write the filename  
     
   Exit nano.  
   Clear the terminal window.
3. In the command line, use the source command to activate the changes in the current session.  
   source ~/.bash\_profile
4. Let's try out the alias. Type  
   pd  
   You should see the same output as you would by typing the pwd command.

What happens when you store this alias in ~/.bash\_profile?  
alias pd="pwd"

The alias command allows you to create keyboard shortcuts, or aliases, for commonly used commands.

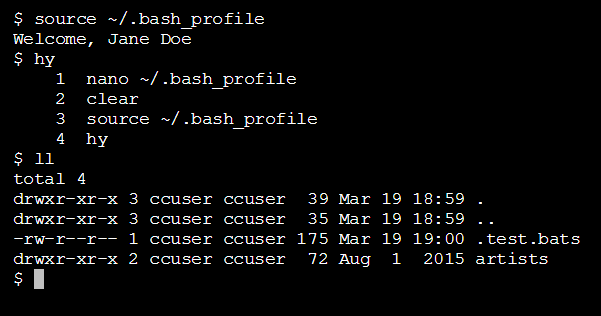
Here alias pd="pwd" creates the alias pd for the pwd command, which is then saved in the bash profile. Each time you enter pd, the output will be the same as the pwd command.

The command source ~/.bash\_profile makes the alias pd available in the current session.

Each time we open up the terminal, we can use the pd alias.

Instructions:

1. Let's practice aliases some more. Open ~/.bash\_profile in nano.
2. In the bash profile, beneath the previous alias, add  
   alias hy="history"  
   Save the file.  
   Press Enter to write the filename.
3. Add another alias  
   alias ll="ls -la"  
   Save the file.  
   Press Enter to write the filename.  
   Exit nano  
   Clear the terminal window.
4. In the command line, use source to activate the changes to the bash profile for the current session.
5. Let's try out the aliases. In the command line, type  
   hy
6. Now type  
   ll



What happens when you store the following aliases in ~/.bash\_profile?

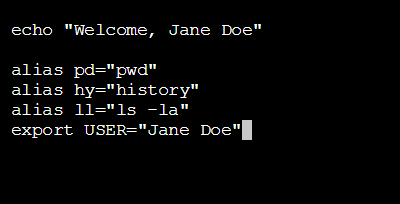
alias hy="history"

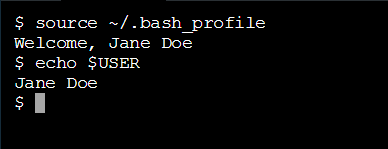
hy is set as alias for the history command in the bash profile. The alias is then made available in the current session through source. By typing hy, the command line outputs a history of commands that were entered in the current session.

alias ll="ls -la"

ll is set as an alias for ls -la and made available in the current session through source. By typing ll, the command line now outputs all contents and directories in long format, including all hidden files.

Instructions:

1. Now that you are familiar with configuring greetings and aliases, let's move on to setting environment variables.  
   Open ~/.bash\_profile in nano.
2. In the bash profile, beneath the aliases, on a new line, type  
   export USER="Jane Doe"  
     
   Feel free to use your own name.  
   Save the file.  
   Press Enter to write the filename.  
   Exit nano.  
   Finally, clear the terminal.
3. In the command line, use source to activate the changes in the bash profile for the current session.
4. Type  
   echo $USER  
   This should return the value of the variable that you set.



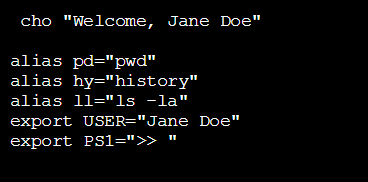
What happens when you store this in ~/.bash\_profile?

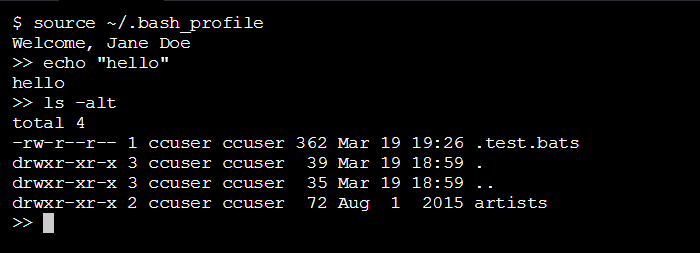
export USER="Jane Doe"

environment variables are variables that can be used across commands and programs and hold information about the environment.

1. The line USER="Jane Doe" sets the environment variable USER to a name "Jane Doe". Usually the USER variable is set to the name of the computer's owner.
2. The line export makes the variable to be available to all child sessions initiated from the session you are in. This is a way to make the variable persist across programs.
3. At the command line, the command echo $USER returns the value of the variable. Note that $ is always used when returning a variable's value. Here, the command echo $USER returns the name set for the variable.

Instructions:

1. Let's learn a few more environment variables, starting with the variable for the command prompt.  
   Open ~/.bash\_profile in nano.
2. On a new line, beneath the last entry, type  
   export PS1=">> "  
   Save the file.  
   Press Enter to write the filename.  
   Exit nano  
   Finally, clear the terminal window.  
   
3. In the command line, use source to activate the changes in the bash profile for the current shell session.
4. Let's try out the new command prompt. In the terminal type  
   echo "hello"
5. Now type  
   ls –alt  
   Did you notice that the prompt has changed?



What happens when this is stored in ~/.bash\_profile?

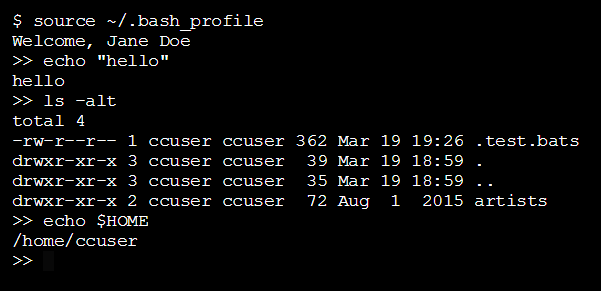
export PS1=">> "

PS1 is a variable that defines the makeup and style of the command prompt.

1. export PS1=">> " sets the command prompt variable and exports the variable. Here we change the default command prompt from $ to >>.
2. After using the source command, the command line displays the new command prompt.

Instructions:

1. Let's learn about two more environment variables.  
   In the command line, type  
   echo $HOME  
   This returns the value of the HOME variable.



What happens when you type this command?

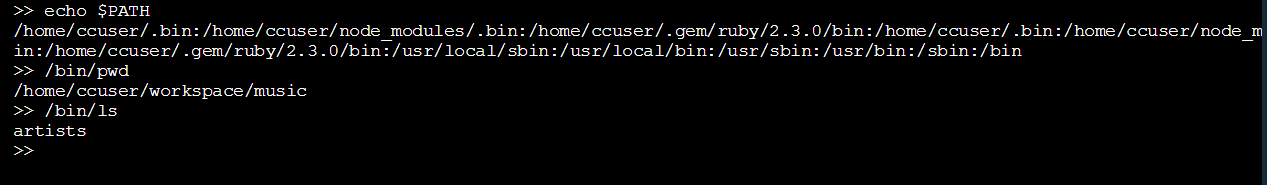
$ echo $HOME

The HOME variable is an environment variable that displays the path of the home directory. Here by typing echo $HOME, the terminal displays the path /home/ccuser as output.

You can customize the HOME variable if needed, but in most cases this is not necessary.

Instructions:

1. In the command line, type  
   echo $PATH
2. Type  
   /bin/pwd
3. Type  
   /bin/ls



What happens when you type this command?

$ echo $PATH

/home/ccuser/.gem/ruby/2.0.0/bin:/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/sbin:/sbin:/bin

PATH is an environment variable that stores a list of directories separated by a colon. Looking carefully, echo $PATH lists the following directories:

1. /home/ccuser/.gem/ruby/2.0.0/bin
2. /usr/local/sbin
3. /usr/local/bin
4. /usr/bin
5. /usr/sbin
6. /sbin
7. /bin

Each directory contains scripts for the command line to execute. The PATH variable simply lists which directories contain scripts.  
For example, many commands we've learned are scripts stored in the /bin directory.

/bin/pwd

This is the script that is executed when you type the pwd command.

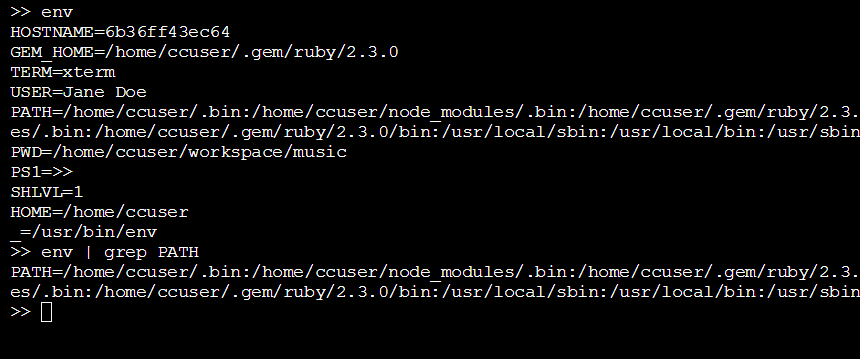
/bin/ls

This is the script that is executed when you type the ls command.

In advanced cases, you can customize the PATH variable when adding scripts of your own.

INSTRUCTIONS:

1. In the command line type  
   env
2. Then type  
   env | grep PATH



What happens when you type this command?

env

The env command stands for "environment", and returns a list of the environment variables for the current user. Here, the env command returns a number of variables, including PATH, PWD, PS1, and HOME.

env | grep PATH

env | grep PATH is a command that displays the value of a single environment variable. Here the standard output of env is "piped" to the grep command. grep searches for the value of the variable PATH and outputs it to the terminal.

SUMMARY:

Congratulations! You learned to use the bash profile to configure the environment. What can we generalize so far?

* The environment refers to the preferences and settings of the current user.
* The nano editor is a command line text editor used to configure the environment.
* ~/.bash\_profile is where environment settings are stored. You can edit this file with nano.
* environment variables are variables that can be used across commands and programs and hold information about the environment.
* export VARIABLE="Value" sets and exports an environment variable.
* USER is the name of the current user.
* PS1 is the command prompt.
* HOME is the home directory. It is usually not customized.
* PATH returns a colon separated list of file paths. It is customized in advanced cases.
* env returns a list of environment variables.

YOUTUBE – GIT TUTORIAL FOR BEGINNERS: COMMAND LINE FUNDAMENTALS

Git is a distributed version control system.  
Central~ is located in one place. youcan check out from the central VCS server, make the changes and check them back in – can be problematic if you can’T get access to the central repository; if the server’s offline, or you don’t have network acces. Also, if the server gets corrupted, you gotta hope there’s a backup somewhere.

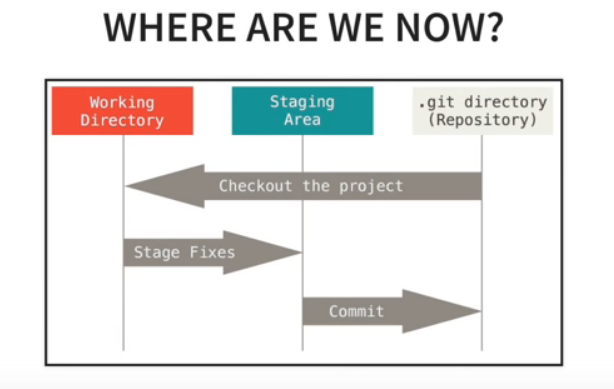
DVCS : everybody has a a local repository. your local repository has all the information that your local repository has, based on the last time that you sync them together.

git help <ver> or git <verb> --help would give you help with a git action.

Example 1:  
Initialize a repository from existing code.  
cd Local-Repo/  
**ls –la**  
**git init** initialize new empty repository  
**ls –la** places a .git directory in the parent repository. If you wanna unfollow the project, you’d just have to remove the .git directory with **rm –rf .git.**

Before our first commit, let’s run git status.  
git status

To ignore a file (to hide personal preferences), you have to create a file:  
touch .gitignore  
A gitignore file is just a simple text file where we can add files that we want git to ignore. Let’s add for example:  
.DS\_Store  
.project  
\*.pyc  
Save it and let’s run git status. You won’t see the affected files now.

Let’s see where we are now.  


Wroking Directory: right now we’re in here. Untracked and modified files are in the working directory and it will list those when we run git status.  
Staging area: where we organize what we want to be commited, so we can pick and chose what we want to commit.

How to add a file to the staging area:  
If we type git add .gitignore  
and git status  
The .gitignore file will be listed in the staging area.  
If we want to add everything to the staging area:  
git add –A  
All of our files are now in the staging are.

If you want to remove files from the staging area:  
git reset file.extension  
To remove everything:  
git reset

How to commit:  
git add –A  
git status  
git comit –m ”Initial commit”  
git status  
nothing to commit, working directory clean

git log  
Will display the commit we’ve just made.

Cloning a reository:  
git clone <url> <where to clone>  
Example: git clone https://github.com/VeraNemedyVarga/remote\_repo.git .  
The dot at the end just means current repository  
cd Cloned-Repo/  
ls –la  
git clone ../remote\_repo.git .  
Would clone every file from remote\_repo.git to current directory.

Viewing information about the remote repository:  
git remote –v  
it lists information to the repository here  
git branch –a  
lists all branches – local and remote in the repository.

How to commit and push files into remote repository.  
when we make a change in our code first we need to commit the changes locally.  
git diff  
git status  
git add –A  
git commit –m „Modified multiply function”

git diff will show me the changes that i have made to the code.  
Now we commited the files locally and we want to push those changes to the remote repository.  
git pull origin master it will pull any changes that have been made since the last time that we have pulled from that repository.  
git push origin master origin is the name of our remote repository and master is the branch that we want to push to.

Common Workflow when working with git:  
So far we’ve been working on our master branch but it isn’t really how you should use git.  
1. Create a Branch for desired Feature:  
git branch branch-name  
git checkout branch-name  
will take you to that branch.  
(git branch lists all the branches)  
  
2. Commit changes  
git status  
git add –A  
git commit –m „Whatever changes made” we have commited the change to our local branch-name branch, but this had no effect on our remote repository.  
  
3. After commit push branch to remote  
git push –u origin branch-name -u tells git that we want to associate our local branch-name branch with the remote one.  
(Then in the future we won’t have to use the –u, we can just do  
git pull  
git push  
and then it will know that those two branches are associated with each other.)  
  
git branch –a  
will show us all our (remote and local) branches, with an \* before the one we’re currently working on.

Merge a Branch with master:  
git checkout master checkout local master branch  
git pull origin master pull all the changes  
git branch –merged shows the branches that we’ve merged so far  
git merge branch-name this merges actually the branch-name branch with master  
git push origin master push the changes to remote master branch

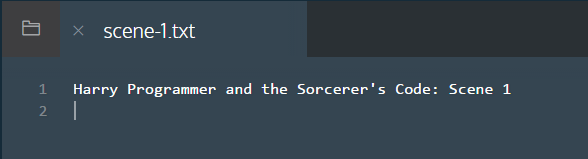
Deleting a Branch:  
get branch –merged check if you managed to merge the branch, so you can delete is cos it’s no longer necessary  
git branch –d deletes local branch  
get branch –a we still have branch-name branch on our local repository  
git push origin –delete branch-name to remove a branch from remote repository  
git branch –a now all we have is our local and remote master branch

CODECADEMY – GIT

Git is a software that allows you to keep track of changes made to a project over time. Git works by recording the changes you make to a project, storing those changes, then allowing you to reference them as needed.

Instructions:

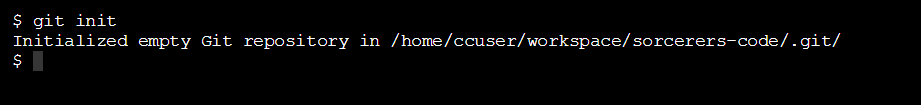
1. We’ll get started by taking a look at the screenplay project.  
   In scene-1.txt, add this text:  
   Harry Programmer and the Sorcerer’s Code: Scene 1  
   Then press enter to create a new empty line. Once you've created the new line, click Run.



Now that we have started working on the screenplay, let’s turn the sorcerers-code directory into a Git project. We do this with:  
git init  
The word init means initialize. The command sets up all the tools Git needs to begin tracking changes made to the project.

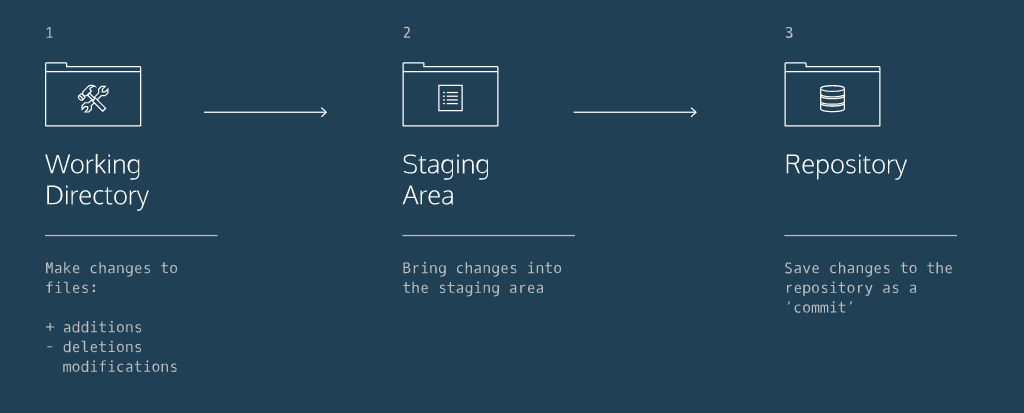
Instructions:

1. In the terminal, initialize a new Git project.  
   Notice the output:  
   Initalized an empty git repository in /home/ccuser/workspace/sorcerers-code/.git/The Git project was created. Click Next to continue.



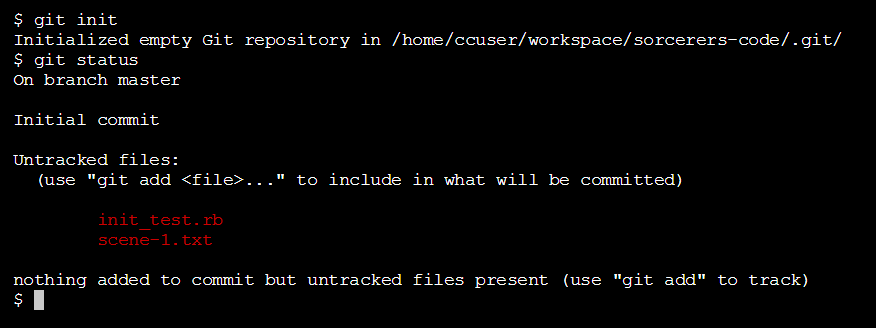
Nice! We have a Git project. A Git project can be thought of as having three parts:

1. A Working Directory: where you'll be doing all the work: creating, editing, deleting and organizing files
2. A Staging Area: where you'll list changes you make to the working directory
3. A Repository: where Git permanently stores those changes as different versions of the project

The Git workflow consists of editing files in the working directory, adding files to the staging area, and saving changes to a Git repository. In Git, we save changes with a commit, which we will learn more about in this lesson.  
Take a look at the diagram. Before we move on, it will help to be familiar with the three parts of the Git workflow.  


As you write the screenplay, you will be changing the contents of the working directory. You can check the status of those changes with:  
git status

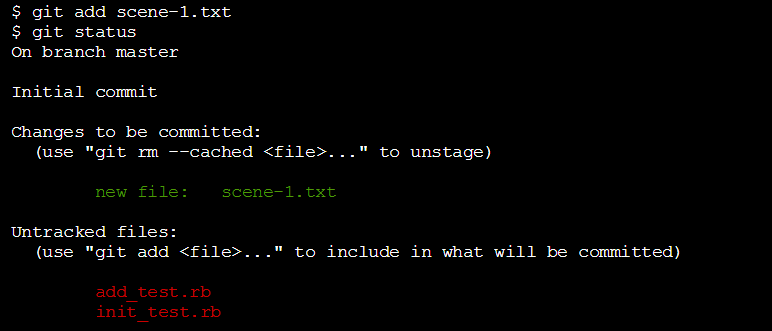
Instructions:

1. From the terminal, check the status of the sorcerers-code project.  
   In the output, notice the file in red under untracked files. Untracked means that Git sees the file but has not started tracking changes yet.  
   

In order for Git to start tracking scene-1.txt, the file needs to be added to the staging area.  
We can add a file to the staging area with:  
git add filename  
The word filename here refers to the name of the file you are editing, such as scene-1.txt.

Instructions:

1. Add scene-1.txt to the staging area in Git. Recall that you will need to identify the file by its name.
2. Check the status of the project in Git.

In the output, notice that Git indicates the changes to be committed with "new file: scene-1.txt" in green text. Here Git tells us the file was added to the staging area.  


Good work! Now you know how to add a file to the staging area.  
Imagine that we type another line in scene-1.txt. Since the file is tracked, we can check the differences between the working directory and the staging area with:  
git diff filename  
Here, filename is the actual name of the file. If the name of my file was changes.txt the command would be   
git diff changes.txt