Reading: Intro to CLI 101

Overview

Learning Objectives

After completing this lesson, you will be able to do the following with a command line interface (CLI):

- Run commands with arguments and modifiers
- Navigate the file system
- Create folders and files

Overview

The command line interface (CLI) is a way to communicate directly with a computer. The user types out commands in plain text and it executes on the computer without the need for buttons or menus. The CLI used to be the only way you could communicate with the computer, but after the invention of graphical user interfaces (GUI) the average computer user in today's world doesn't need the CLI anymore. So why use a CLI? As a developer, you can perform actions more efficiently, access areas of the computer that aren't accessible in a GUI, and configure your machine for programming.

Running Commands

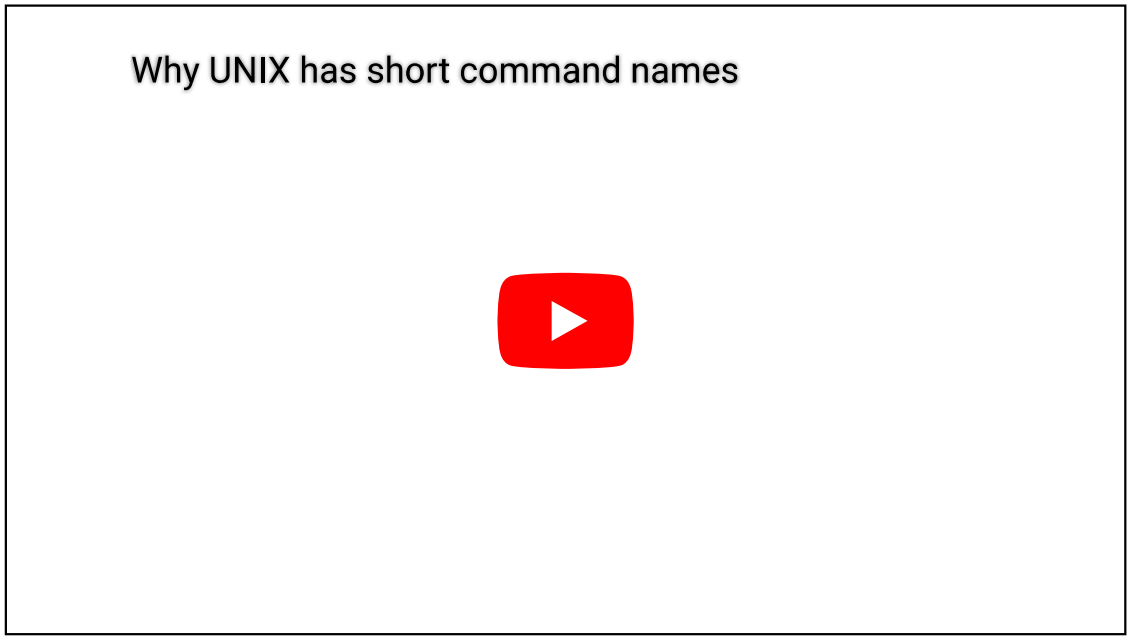
To get started, most operating systems have a built-in application called "terminal". Launch this and you should see a blank screen with a cursor that lets you input a single line of text. This is called the prompt. Often, the prompt is a single character, such as the $ sign: `$`. Many programmers choose to customize their prompt to show them useful information, but $ is a popular default. Run your first command by typing `pwd` at the prompt and pressing the enter key, which may be labeled `return` on your keyboard. You should see some text printed to the terminal (though your output will be a little different): `$ pwd`

```
/Users/chrisaquino
```

You just asked your computer to "Print Working Directory". Great! ...what's a working directory? A computer's hard drive is like a board game. You have a game-piece somewhere on the board. You use the `pwd` command to find out the current location of your game piece. By default, a new terminal window starts in your home directory, which is the one that holds your `Documents`, `Movies`, `Music`, and `Pictures` directories.

Why are the command names so short?

Back in the 70s, when terminals were the primary way to interact with a computer, programmers had to decide what to name all the basic commands. They chose short, but memorable command names. According to one of the pioneers of early computing: the real reason for short command names is that the keys were hard to press 😂



Source: catonmat.net/why-unix-commands-are-short

Using Arguments

The next command you're going to learn is `ls`, which is the list command. Run that command and press enter. You will see a list of files in your directory

```
$ ls

Applications Desktop Documents Downloads Library Movies Music Pictu
res Public
```

You can customize the output by adding command line arguments. In computing, an argument is additional information you provide to a command.

Let's add an argument to get the long version of the output. Type `ls -l` and press `return`. You will see a vertical list like the following:

```
$ ls -l

drwx------@  5 chrisaquino  staff   160 Jun  4 10:13 Applications
drwx------@ 15 chrisaquino  staff   480 Oct 15 17:42 Desktop
drwx------@ 10 chrisaquino  staff   320 Jun  4 09:48 Documents
drwx------@ 64 chrisaquino  staff  2048 Oct 15 16:18 Downloads
drwx------@ 70 chrisaquino  staff  2240 Jun 15 13:54 Library
drwx------+ 44 chrisaquino  staff  1408 Jun  5 15:37 Movies
drwx------+  6 chrisaquino  staff   192 May  1 08:40 Music
drwx------+  7 chrisaquino  staff   224 Nov  7  2019 Pictures
drwxr-xr-x+  4 chrisaquino  staff   128 Oct  9  2019 Public
```

This is where the CLI is more powerful than a GUI. You didn't have to change your app preferences or run a different program to get customized output. You simply ask ls for more information.

The output from `ls -l` shows quite a lot of information. Though it looks cryptic now, you will already recognize some of what's displayed. For example, your username in

one of the columns. This column shows the owner of the file. Another column shows when the file was created.

Another useful argument is `-a` which shows hidden files. Try it, and you'll see output like this:

```
$ ls -a
.bash_history                   Documents
.bash_profile                   Downloads
.bash_sessions                  Library
.bashrc                         Movies
.gitconfig                      Music
.ssh                            Pictures
.vscode                         Projects
Applications                    Public
Desktop
```

The output shows the same items as before, but also some new ones that start with a `.` -- these are the hidden files. Hidden files and directories are used by the Operating System to store configuration information. As a developer, you'll occasionally edit these.

Customizing the Prompt

Let's try changing one of these hidden files by customizing your prompt. The terminal will use any text you want as the prompt. Some users include things like:

- The working directory

- The date/time
- Emojis

We'll start with a simple but useful customization: showing the working directory as part of your prompt.

First, let's find out what terminal program you're using:

```
echo $SHELL
```

The output should contain one of these two words:

- `bash`
- `zsh`

Each of those is a *shell* program. The purpose of a shell is to interpret and run your commands. Your terminal will likely be configured to use one of those two.

For `bash` Users:

If your shell is `bash`, use this command:

```
code .bash_profile
```

At the end of the file, add a new line with this text:

```
PS1='[\w]$ '
```

Now skip to the All Users section below.

For `zsh` Users:

If your shell is `zsh`, this command is for you:

```
code .zshrc
```

Add this line to the end of that file:

```
PROMPT='[%.]$ '
```

For All Users

Save your file, close your terminal, and open it again.

Your prompt will now look like this:

```
[~]$
```

The square brackets are purely decorative. They are meant to help you see the text between the brackets, which now shows you what directory you're in as you move from one directory to another.

The `~` is the "tilde" character and it is shorthand for your "home directory".

Combining Arguments

What if you wanted to see a long listing that includes hidden files? You could pass multiple arguments to the `ls` command:

```
[~]$ ls -l -
a
-rw-------    1 chrisaquino  staff  10937 Oct 16 15:41 .bash_histor
y
-rw-r--r--    1 chrisaquino  staff    208 Oct 13 14:24 .bash_profil
e
drwx------  108 chrisaquino  staff   3456 Oct 16 11:31 .bash_sessio
ns
-rw-r--r--    1 chrisaquino  staff    695 Sep 21 12:47 .bashrc
-rw-r--r--    1 chrisaquino  staff    464 May  8 16:02 .gitconfig
drwx------   12 chrisaquino  staff    384 Oct 15 11:00 .ssh
drwxr-xr-x    4 chrisaquino  staff    128 Nov 18  2019 .vscode
drwx------@   5 chrisaquino  staff    160 Jun  4 10:13 Applications
drwx------@  15 chrisaquino  staff    480 Oct 15 17:42 Desktop
drwx------@  10 chrisaquino  staff    320 Jun  4 09:48 Documents
drwx------@  64 chrisaquino  staff   2048 Oct 15 16:18 Downloads
drwx------@  70 chrisaquino  staff   2240 Jun 15 13:54 Library
drwx------+  44 chrisaquino  staff   1408 Jun  5 15:37 Movies
drwx------+   6 chrisaquino  staff    192 May  1 08:40 Music
drwx------+   7 chrisaquino  staff    224 Nov  7  2019 Pictures
drwxr-xr-x   16 chrisaquino  staff    512 Oct  7 16:31 Projects
drwxr-xr-x+   4 chrisaquino  staff    128 Oct  9  2019 Public
```

However, it's easier to write it like this:

```
[~]$ ls -la
-rw-------    1 chrisaquino  staff  10937 Oct 16 15:41 .bash_history
-rw-r--r--    1 chrisaquino  staff    208 Oct 13 14:24 .bash_profile
drwx------  108 chrisaquino  staff   3456 Oct 16 11:31 .bash_sessions
-rw-r--r--    1 chrisaquino  staff    695 Sep 21 12:47 .bashrc
-rw-r--r--    1 chrisaquino  staff    464 May  8 16:02 .gitconfig
drwx------   12 chrisaquino  staff    384 Oct 15 11:00 .ssh
drwxr-xr-x    4 chrisaquino  staff    128 Nov 18  2019 .vscode
drwx------@   5 chrisaquino  staff    160 Jun  4 10:13 Applications
drwx------@  15 chrisaquino  staff    480 Oct 15 17:42 Desktop
drwx------@  10 chrisaquino  staff    320 Jun  4 09:48 Documents
drwx------@  64 chrisaquino  staff   2048 Oct 15 16:18 Downloads
drwx------@  70 chrisaquino  staff   2240 Jun 15 13:54 Library
drwx------+  44 chrisaquino  staff   1408 Jun  5 15:37 Movies
drwx------+   6 chrisaquino  staff    192 May  1 08:40 Music
drwx------+   7 chrisaquino  staff    224 Nov  7  2019 Pictures
drwxr-xr-x   16 chrisaquino  staff    512 Oct  7 16:31 Projects
drwxr-xr-x+   4 chrisaquino  staff    128 Oct  9  2019 Public
```

Arguments will always be specific to the command. To find out how to use a command, there's a special argument built into many commands: `--help`. To see the help text, you would type `ls --help`. (The output is usually pretty long. You can scroll in the terminal just like you do a web page.)

This is an example of a *long* argument that uses two dashes and a word (instead of a single letter).

Using Modifiers

You can add modifiers to a command to change its behavior further. One of the most useful skills when using the CLI is navigating the filesystem. The main navigation command is `cd` which stands for "change directory". That is, it changes the working directory by "moving" your prompt to a different part of the hard drive.

To change directories, you have to tell the system which directory you want to change to.

For example, to go from your home directory to your Downloads directory, you type:

```
cd Downloads
```

If you followed along earlier and customized your prompt, it should look like this:

```
[~/Downloads]$
```

(In case you skipped that part, you can check your working directory with `pwd`.)

Since folders can be put inside folders, it's important to know how to move backwards and forward (upwards and downwards) in a file system. Moving forward you just type the name of the directory you want to change into (ex: `cd Downloads`) and `../ modifier` is used for going backwards (ex: `cd ../`).

Here is a quick cheat sheet for how.

Meaning	Meaning	Example
`/`	Root of drive	`cd /`
`./`	Current Directory	`open ./Downloads`
`../`	Previous Directory	`cd ../`

You can begin to combine these concepts to do some powerful things with the CLI. If you wanted to go back two directories, you can type `cd ../../`.

Creating Files & Directories

Creating new files and directories are two different commands in the CLI. You create directories with `mkdir` and create new files with `touch`. Keep in mind that new files are just plain text files, so when you want to create a new file for a graphical user interface program (ex: Microsoft Word) you have to create the new file from within the application. But as a computer programmer, code files are all stored in plain text, which makes it easy to set up a project's file and directory from inside the CLI.

Summary

The command line interface (CLI) is a powerful tool for the user to interact with the computer directly without needing a graphical user interface (GUI). Typically computer programmers use the CLI to be more efficient, access areas of the computer that aren't accessible from a GUI, and configure your computer to do things for programming needs (start servers, modify access rights, etc.).

Here are the basics you learned:

- How to execute a command with `pwd`

- How to add arguments to a command with `ls -la`
- How to add modifiers to a command with `cd Downloads`
- How to navigate the file system with `cd`
- How to create files and folders with `mkdir` and `touch`
