Previous Next

The Command Line

Estimated Time: 1-2 hours

A command line interface (or the command line) is a tool for browsing and interacting with your computer by typing commands. By the end of this assignment, you'll know how to control your Cloud9 workspace (or, really, the Linux computer on which your Cloud9 workspace is installed) from the command line using basics commands that every developer uses. Specifically, you'll learn how to create, copy, move and delete files and folders and navigate your computer. You've probably done each of these things in the past using the graphical interface, but as a developer you'll do these things so often that you need a faster method.

It takes some time – probably a few weeks – to become comfortable at the command line. Until that happens keep practicing the skills you pick up in this assignment, and don't be afraid to make mistakes while you're still getting comfortable. The assignment is focused on understanding a small number of commands; at the end of the assignment you will find a list of commands you should memorize.

Orienting yourself

First of all you will need to fire up the console in your Cloud9 workspace. Then before we start to work with files and folders we will try to orient ourself a little. When you open up your terminal you will see something like this:

```
thinkful@thinkful-pip-1:~ $
```

This is called your *command prompt*. It is split into three sections. The first section, in this case `thinkful`, tells you which user you are working as. The second section, `thinkful-pip-1` tells which workspace you are working in. The final section, `~` tells you which directory you are in.

In tutorials you will often see the prompt abbreviated to just the final `$`. This indicates the start of where you type commands. The results of the commands will not have a `$` sign in front of them.

There is also a second way to find out which directory you are in: the `pwd` command. Try running the command by typing in `pwd` and hitting enter:

```
$ pwd
/home/ubuntu
```

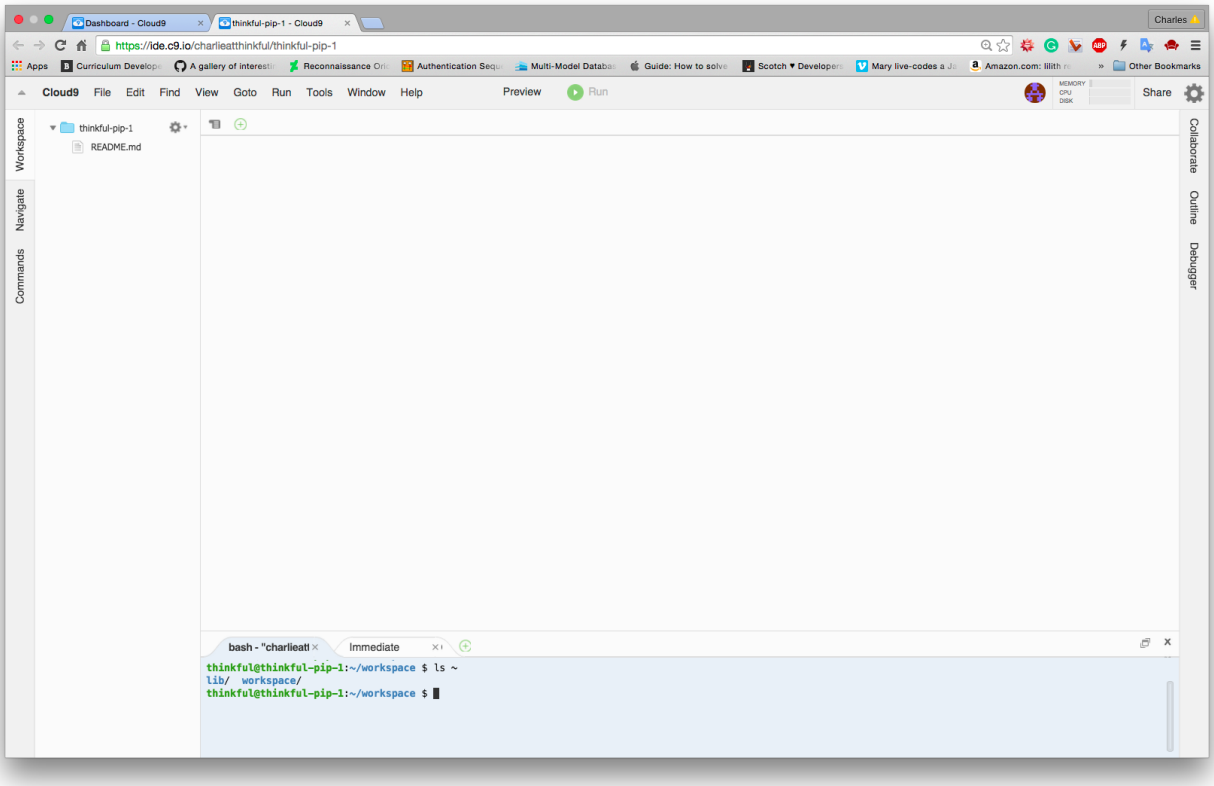
`pwd` stands for "print working directory." You'll want to remember this command, so when you type 'pwd' try saying "print working directory" out loud (unless you're working in a coffeeshop...)

In this example we are in the */home/ubuntu/workspace* directory. But our prompt is telling us that we are in the *~/workspace* directory. What gives? Well, the ~ character is the first of a number of shortcuts which you can use at the command line. It always refers to your home directory. So in this case ~ will expand to */home/ubuntu* behind the scenes.

Now that we know where we are, let's try to find out what's around us. To do this we can use the `ls` command to list the files and folders in our home directory:

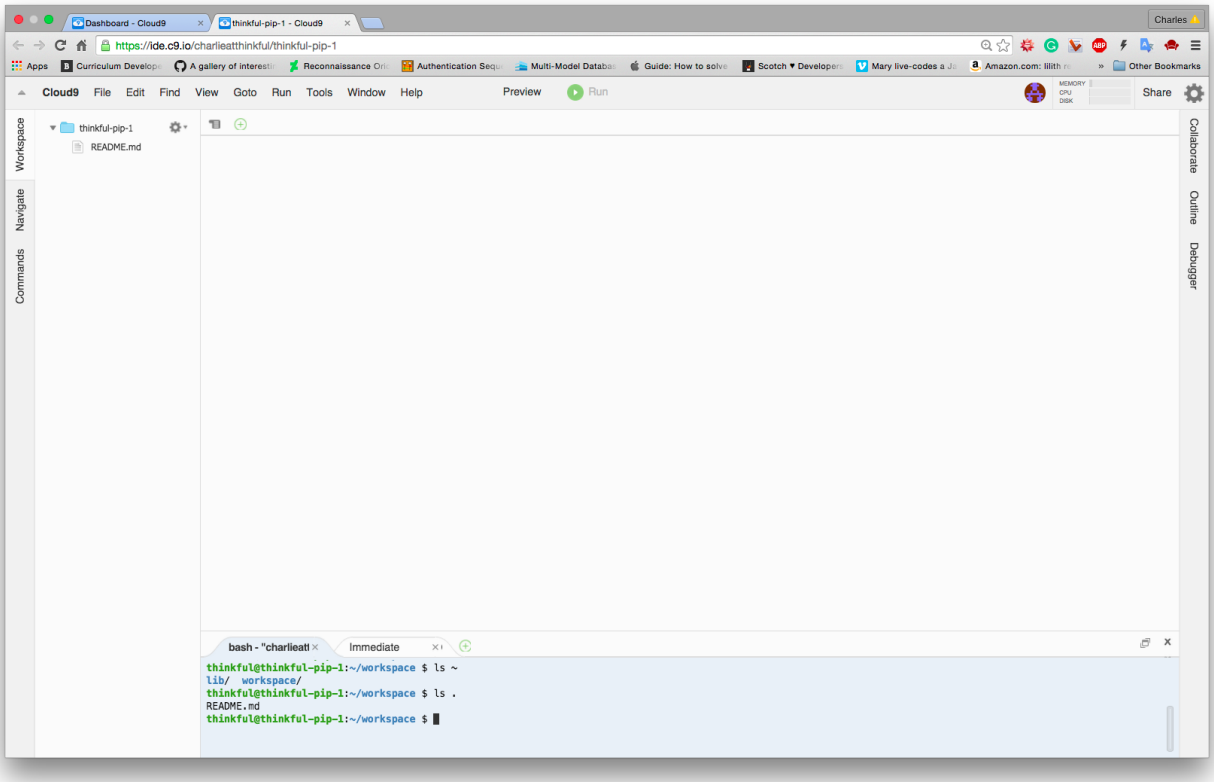
```
$ ls ~
```

Here's an example of this being run on a Cloud9 workspace. Notice how what is listed in the terminal corresponds to the graphical view above. *workspace* is colored blue, to indicate that it is a directory. Cloud9 expects all your working files and directories to be in the *workspace* directory, so we will see it a lot.



When running the `ls` command we have used our first *command-line argument* - the tilde ~ after the `ls` command. Command line arguments are inputs into command line applications. The number and style of arguments vary from program to program, but in general each argument should be separated by a space.

The `.` (dot) argument is the second shortcut which we will learn. A single dot always refers to the directory which we are currently in. So here we are instructing `ls` to list the files in the current directory.



Right now, this directory contains only the `README.md` file that was created along with the workspace.

Because listing the current directory is very common we can omit the argument and get the same result. Try it out by simply running `ls`!

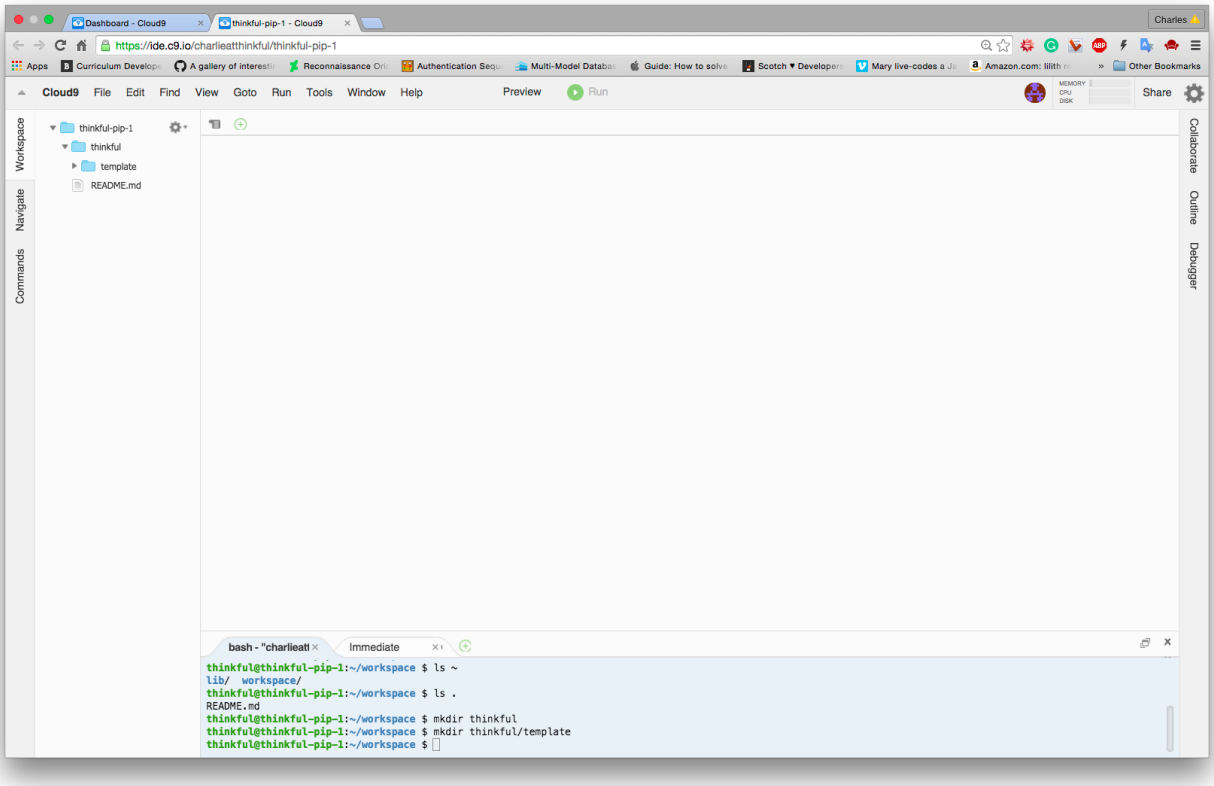
Creating a projects directory

Now that we know where we are we can start to build a series of folders which we will use to store our work throughout the courses.

let's create two folders, one inside another. The first folder will hold all of your work on Thinkful Python projects, and the second will act as a template which can be used whenever you want to start a new project. To create the directories we can use the `mkdir` command:

```
$ mkdir thinkful
$ mkdir thinkful/template
```

Here's the commands, run on Cloud9. You see that the new directories are also visible in the file manager. Our `mkdir` command works exactly as if we had created a new directory in the file manager. Notice how we can access deeper levels of our directory structure by separating the names of directories using forward slashes.



Next let's create an empty README file in the template folder. It is good practice to have a README file in every project to hold basic documentation such as how to install and run the project.

To create the file, right click on the template folder in the file manager, and select *New File*. Then give the file the name *README.md*.

Next we are going to start a new project called *hello_world* by making a copy of the *template* directory.

To copy files and directories we need to use the `cp` command. But first to make our life easier let's move into the *thinkful* directory using the `cd` command:

```
$ cd ~/workspace/thinkful
```

Notice how when you run this command your prompt changes to tell you that you are now working in the *~/workspace/thinkful* directory. This means that any paths we use will be relative to this directory. Let's see what that means in practical terms by copying the template directory:

```
$ cp -r template hello_world
$ ls
hello_world  template
$ cd hello_world
$ ls
README.md
```

The first argument to `cp` tells it what to copy – in this case the *template* directory. The second argument tells it where to copy to – the *hello_world* directory. Because we have changed directory we give the paths to these directories relative to the *thinkful* directory. We also need to give the `-r` argument. This means that it should copy the directory *recursively* (i.e. it should also copy all of the contents of the directory, and any directories below that, and so on).

Try It!

- Create a new file called *hello_world.py* inside the *hello_world* directory.
- Edit the file so that it prints the string "Hello World!"
- Run the file from the command line to make sure that it works

Creating a notes directory

Now we are going to rearrange your *thinkful* directory so it is split into two subdirectories – one for your projects, and one for any notes which you make

during the course.

First we need to get back from *~/workspace/thinkful/hello_world* to *~/workspace/thinkful*. In order to do that we can use yet another shortcut:

```
$ cd ..
$ pwd
/home/ubuntu/workspace/thinkful
```

The double-dots (`..`) always refer to the directory above your working directory. So here we are moving up one level of the directory tree.

Next we need to create a new directory to hold the projects:

```
$ mkdir projects
```

Then we can move the *template* and *hello_world* directories into the projects directory:

```
$ mv * projects
$ ls
projects
$ ls projects
hello_world  template
```

Here we use the `mv` command, which will move or rename files and folders. The arguments to `mv` are the same as those for `cp` : the first is what needs moving, and the second is where to move it to. For the first argument we use the * wildcard character. This can be used to match multiple files and folders. For example `ls *.mp*` will list any MP3, MP4 or MPG files in a directory. Note that this command will give a warning telling us that we are trying to move the *projects* directory (which is part of *) into itself. This can be safely ignored.

We can then create our *notes* directory:

```
$ mkdir notes
```

You might want to create a new text file inside your notes directory with some reminders on how to use the command line.

Try It!

Use the `cp` command to create a backup copy of your *hello_world* project, called *hello_world_backup*.

Deleting file and folders

The `rm` command is used to delete files and directories. To test this out we are going to delete the *hello_world.py* file which we have backed up:

```
$ cd ~/workspace/thinkful/projects
$ rm hello_world/hello_world.py
```

Then we can restore the project from our backup. This is a two-stage process. First we need to delete the *hello_world* directory:

```
$ rm -r hello_world
```

Notice how we again have to use the `-r` when deleting the directory. We can then rename the *hello_world_backup* directory using the `mv` command:

```
$ mv hello_world_backup hello_world
```

And that's it! If you know how to `pwd`, `ls`, `mkdir`, `cp`, `mv` and `rm` then you have covered most of the command line skills which you'll need to use throughout the course. Mastering these basics may take a while, but they

should soon become second nature. You'll be wondering why your computer came with a mouse in no time!

A final challenge

At the moment your Fizz Buzz project from Lesson 1 is languishing in your home directory, just waiting to be neatly organized inside your brand new projects folder. Use the command line to create a new project called *fizz_buzz* based upon your template. Then move your Fizz Buzz code into the project directory. Finally run your code to make sure that it has been moved over successfully.

☆ ☆ ☆ ☆ ☆ · Report a typo or other issue

Completed

Previous Next