14 Developer Tools: Command Line

"If you have any trouble sounding condescending, find a UNIX user to show you how it's done."

— Scott Adams, Dilbert Cartoonist

You know that super cool green-on-black hacker-looking screen you always see in the movies? It's not that cool in real life… it's way cooler. Most developers need this tool. This checkpoint will familiarize you with the basic commands and principles of the command line.

What Is the Command Line?

The command line allows developers to navigate their filesystem, alter files, execute programs, install software, create new applications, and much more. We use the terms command line, command line prompt, terminal, and **shell** interchangeably in the developer community.

Unix offers "flavors" of shells that function on any Unix-based operating system such as Linux and OS X. But, the content in this checkpoint is shell and operating system agnostic.

Why Do I Need to Learn the Command Line?

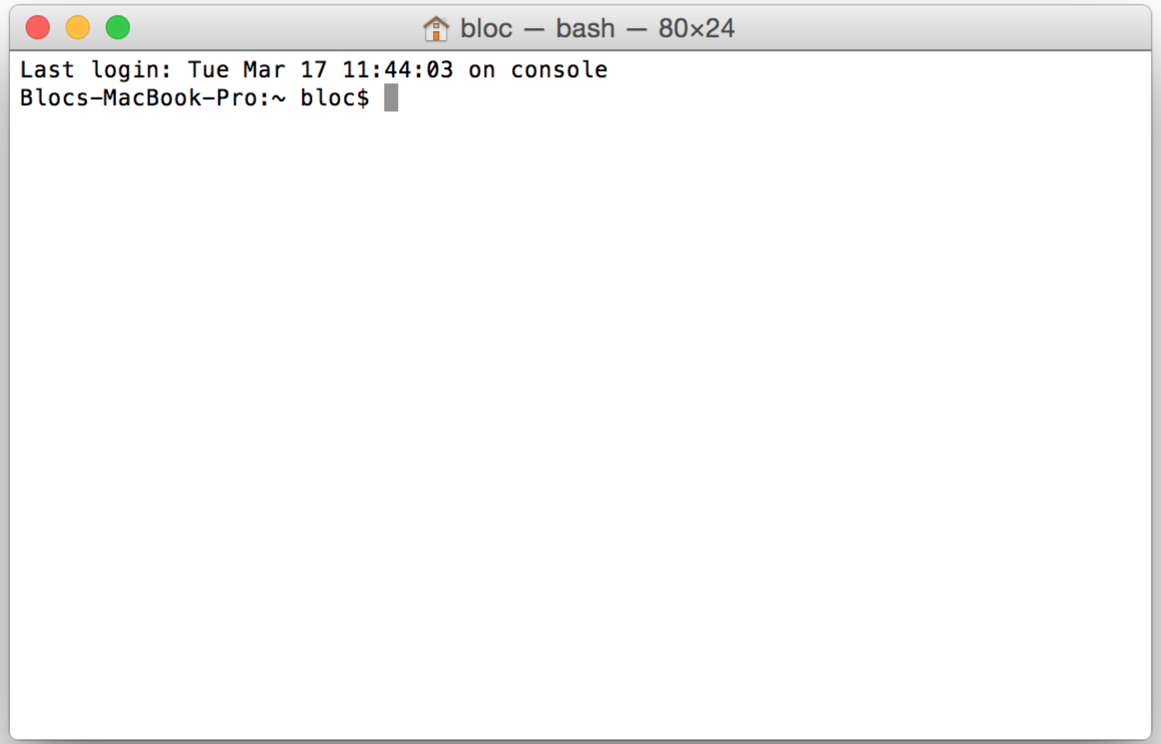
The command line, once you become familiar with it, is an efficient way to access systems, install software, and execute programs and tests. As you learn with Bloc, you'll use the command line to perform actions, and become comfortable with the command line in the process.

How Do I Open the Command Line?

Use the following instructions to open the command line:

Operating System	Guide
OS X	The command line utility packaged with the operating system is **Terminal**. To access Terminal, use Finder and go to the `Applications > Utilities` folder or use Spotlight and type *Terminal*.
Windows (Rails Students)	For Rails students, we recommend using **Cloud9**. By default, the **box at the bottom of the screen** is the Cloud9 Terminal.
Windows	On Windows, we recommend using **Git BASH** for all command line activities. Download it from the Git BASH website, install it, and open it when the installation finishes.
Linux	Read Ubuntu's **"Starting a Terminal"** section.

Where Am I?



```
Last login: Tue Mar 17 11:44:03 on console
Blocs-MacBook-Pro:~ bloc$ 
```

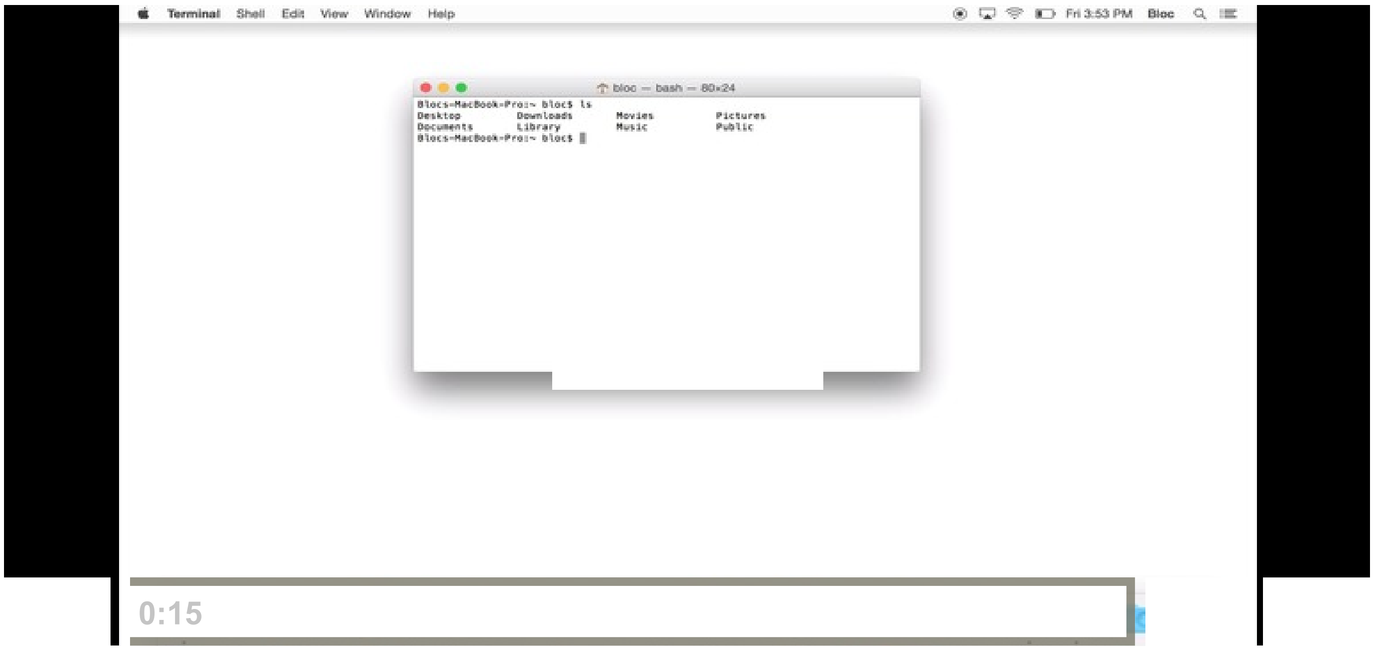
> Terminal in OS X

When we open the shell, it presents a prompt. At this point, the shell is complacent and waiting for instruction. We can dictate any command to the shell. When we press return (↵), the operating system executes the command. Type `ls` (lower case 'L', lower case 'S') and then press return ↵. After executing `ls`, type `open .` (don't forget the space) and press return ↵. Feel free to copy the commands:

~

```
$ ls
Applications    Desktop        Downloads        Movies        Pictures
Documents           Library        Music            Public
$ open .
```

Here is a video demonstration for OS X:



```
Blocs-MacBook-Pro:~ bloc$ ls
Desktop         Downloads       Movies          Pictures
Documents       Library         Music           Public
Blocs-MacBook-Pro:~ bloc$
```

The operating system executed the `ls` command after we pressed return. The `ls` command listed the contents of our current directory. A directory in Unix is the same thing as a folder in Finder (OS X) or a folder in File Explorer (Windows). Notice the names in the output of `ls` match the folder titles in Finder.

The `open .` command directed the operating system to open 'this', the period, which refers to the current directory. The operating system chose Finder to open the directory and display its contents.

> The `open .` command does not work in Git BASH.

Like the Finder window in OS X, the shell has a default location when opened. The Finder will default to the last folder that we were operating in when we closed it. But the shell always defaults to a directory specified in its configuration file. By default, the shell places us in our `HOME` directory.

You can also type `echo $HOME` to find the actual directory that your Git BASH/terminal defaults to, so you know what folder to open to get to your code.

What Can I Do Now?

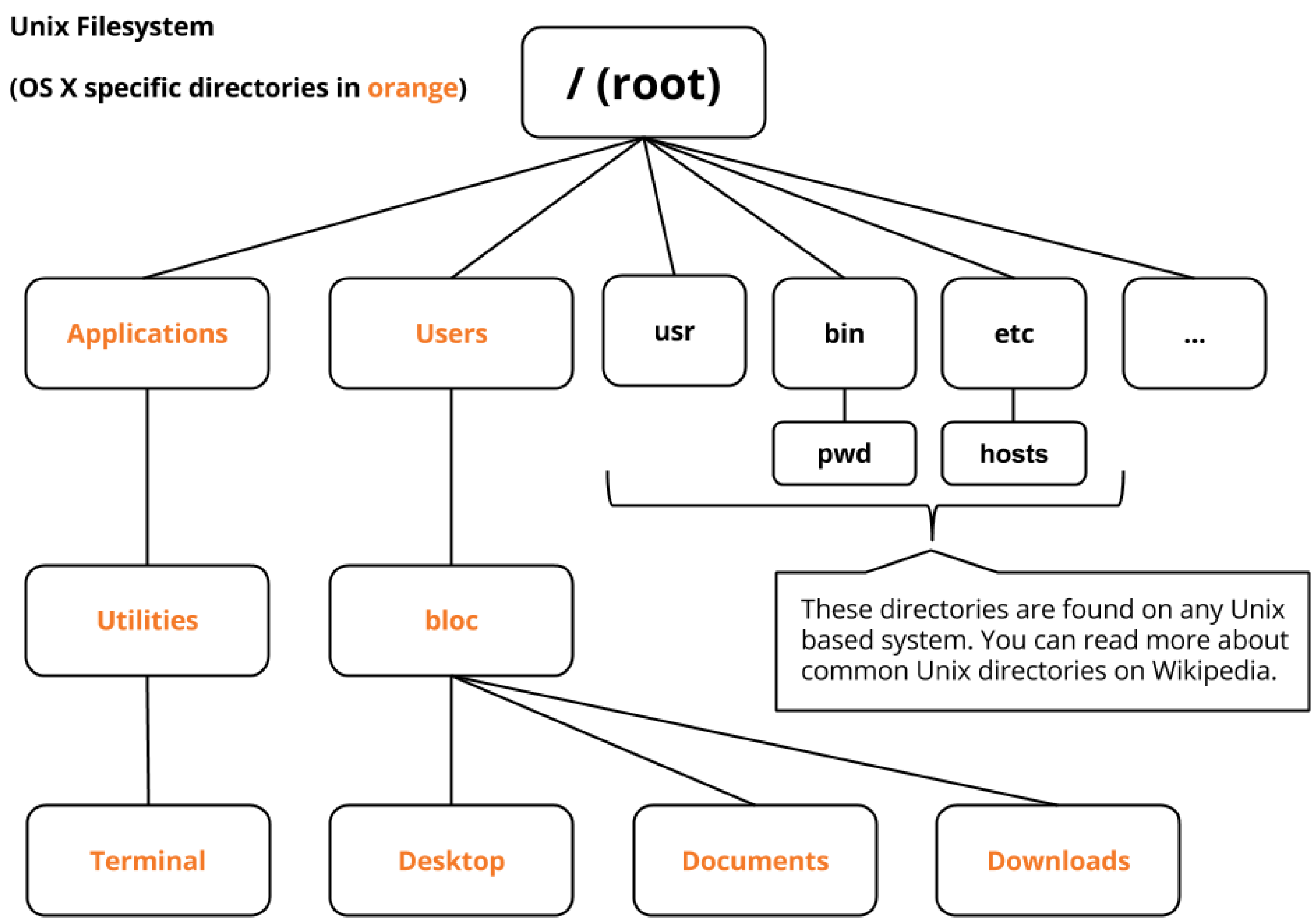
Now that we have a frame of reference, let's go over the basic commands that help us traverse the filesystem. Verify the current working directory by executing the `pwd` command. The `pwd` command stands for 'print working directory'.

~

```
$ pwd
```

The `pwd` command prints the name of our current directory to the command prompt. For OS X users it will be `/Users/your_username`, Windows users will see `/c/home/your_username` or `/home/ubuntu/workspace` on Cloud9, and Linux users will see `/home/your_username`. This is our `HOME` directory. The `~` character is also a reference to the `HOME` directory in any Unix-based system.

The filesystem is a tree, at the top of the tree is the root denoted by `/`. Below the root there are branches and other **vertices**. The following graphic illustrates parts of an example filesystem:


Unix Filesystem
(OS X specific directories in orange)
/ (root)
Applications
Users
usr
bin
etc
...
pwd
hosts
These directories are found on any Unix based system. You can read more about common Unix directories on Wikipedia.
Utilities
bloc
Terminal
Desktop
Documents
Downloads

The `cd` (change directory) command offers the ability to traverse the branches to get to other vertices. Let's move through the filesystem by using the `cd` command:

~

```
$ cd ..
```

This command navigates the shell to its immediate *parent* directory. The parent of any vertex in the graph is the vertex that is connected directly above. Given the graph above, the parent of `bloc` is `Users`. The `..` is another way to reference the parent directory from the current working directory in the shell.

Perform a `pwd` again:

~

```
$ pwd
```

The output should display the parent vertex of the `bloc` directory: `/Users` on OS X, `/c/home` in Git BASH, `/home/ubuntu` on Cloud9, and `/home` on Linux.

Pay special attention to the naming. In Windows or Linux the `/c/home` and `/home` directories are *not* the `HOME` directory, but are the parent vertices of the `HOME` directory, respectively.

Performing an `ls` will display the `HOME` directory:

~

```
$ ls
Guest     Shared    your_username
```

To get back to the `HOME` directory, use `cd` again:

~

```
$ cd your_username
```

On Cloud9, use:

~

```
$ cd workspace
```

Executing the `ls` command again will output the contents of the `HOME` directory.

Paths

In our previous example, we used the `..` operator to move to the directory above our `HOME` directory. The `..` operator is an example of a *relative path*. Relative paths refer to other files or directories in the filesystem compared to the current working directory.

In the graphic above, the relative path from the `Desktop` directory to the `Users` directory is: `../../` since the `Users` directory is two directories above the `Desktop` directory in the filesystem hierarchy. Imagine we want to get a reference to the `hosts` file from the `Desktop` directory. Using our graphic above, the relative path from `Desktop` to `/etc/hosts` is `../../../etc/hosts`.

Unix also gives us the ability to reference the `hosts` file using *absolute paths*. An absolute path is the fully qualified path of a file or directory starting from the root. The absolute path of the `hosts` file is `/etc/hosts`. The absolute path of the `Desktop` directory is `/Users/bloc/Desktop`. Notice how both examples start at the root `/`.

Addresses are like paths in Unix. We can say Charlie's house is two doors down from mine. Or, we can address his house by the full address: 1234 Same Street. The former is relative as it's a correlation between a point of reference and Charlie's house whereas the latter is the exact, or absolute, address. They both reference the same house, but in a different manner.

Read more about **Unix paths**.

How Can I Manipulate Files and Directories?

We can make our way through the filesystem, but that doesn't offer us much utility. This is a list of useful commands:

Command	Guide
mkdir	Make a directory.
file	Determine a file's type.
touch	Create file or change file modification access time.
cp	Copy a file.

mv	Move or rename a file.
rm	Remove a file.
rmdir	Remove a directory.
history	Display a chronological list of the previously issued commands.

Let's go through an example that uses each of the above commands. First issue the `mkdir` command:

~

```
$ mkdir bloc
$ ls
bloc
$ file bloc
bloc: directory
```

We created a directory named `bloc`. `file` defines its file type. `cd` into `bloc` and issue the `ls` command:

~

```
$ cd bloc
$ ls
```

The output is blank. Use `touch` to create a new file:

~/bloc

```
$ touch bloc_file
$ ls
bloc_file
$ file bloc_file
bloc_file: empty
```

`touch` created an empty file named `bloc_file` with an unspecified type. Copy `bloc_file`, rename it and remove the original:

~/bloc

```
$ cp bloc_file new_bloc_file
$ ls
bloc_file    new_bloc_file
$ mv new_bloc_file renamed_bloc_file
$ ls
bloc_file    renamed_bloc_file
$ rm bloc_file
$ ls
renamed_bloc_file
```

Remove `renamed_bloc_file`, traverse up and out of the `bloc` directory and remove the `bloc` directory:

~

```
$ rm renamed_bloc_file
$ ls

$ cd ..
$ rmdir bloc
$ ls
```

The output of `ls` will show that `bloc` no longer exists. Finally, issue the `history` command to display the list of commands we executed in chronological order with the oldest at the top.

~

```
$ history
285  mkdir bloc
286  ls
287  file bloc
288  ls
289  cd bloc
290  ls
291  touch bloc_file
292  file bloc_file
293  ls
294  cp bloc_file new_bloc_file
295  ls
296  mv new_bloc_file renamed_bloc_file
297  ls
298  rm bloc_file
299  ls
300  rm renamed_bloc_file
301  cd ..
302  ls
303  rmdir bloc
304  history
```

What Are Man Pages?

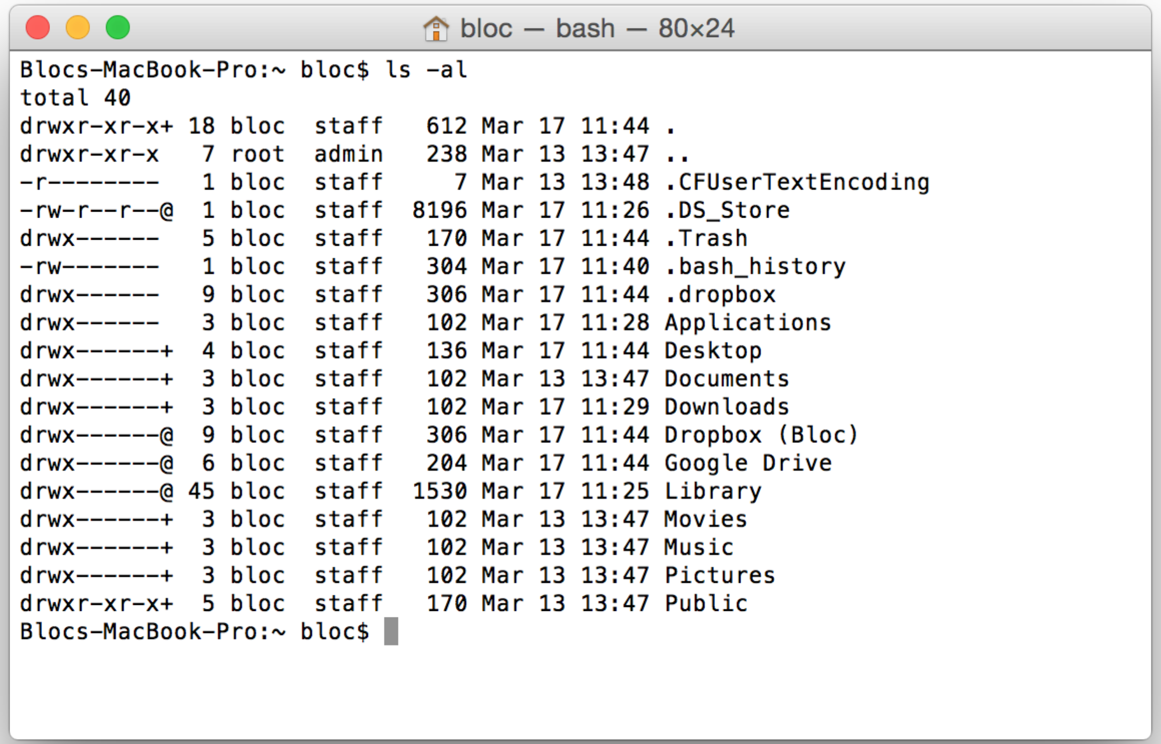
Every Unix distribution includes Man pages, **short for manual pages**. Man pages can be terse but are a valuable resource. Let's see an example of a man page:

```
$ man ls
```

> Use the `j` and `k`, the spacebar and `u` keys, or ↓ and ↑ to scroll up and down. Use the `q` key to exit the man page program.

What Are Command Options?

Notice in the description of the `ls` man page, it lists a lot of options. This is typical of most Unix commands. Most commands have the ability to accept input. Options are also commonly referred to as *flags*. The `ls` command can take flags and exemplifies a typical Unix command. Issue a `cd` with no options; this will navigate the shell to our `HOME` directory and then perform `ls -a -l`:



```
Blocs-MacBook-Pro:~ bloc$ ls -al
total 40
drwxr-xr-x+ 18 bloc  staff   612 Mar 17 11:44 .
drwxr-xr-x   7 root  admin   238 Mar 13 13:47 ..
-r--------   1 bloc  staff     7 Mar 13 13:48 .CFUserTextEncoding
-rw-r--r--@  1 bloc  staff  8196 Mar 17 11:26 .DS_Store
drwx------   5 bloc  staff   170 Mar 17 11:44 .Trash
-rw-------   1 bloc  staff   304 Mar 17 11:40 .bash_history
drwx------   9 bloc  staff   306 Mar 17 11:44 .dropbox
drwx------   3 bloc  staff   102 Mar 17 11:28 Applications
drwx------+  4 bloc  staff   136 Mar 17 11:44 Desktop
drwx------+  3 bloc  staff   102 Mar 13 13:47 Documents
drwx------+  3 bloc  staff   102 Mar 17 11:29 Downloads
drwx------@  9 bloc  staff   306 Mar 17 11:44 Dropbox (Bloc)
drwx------@  6 bloc  staff   204 Mar 17 11:44 Google Drive
drwx------@ 45 bloc  staff  1530 Mar 17 11:25 Library
drwx------+  3 bloc  staff   102 Mar 13 13:47 Movies
drwx------+  3 bloc  staff   102 Mar 13 13:47 Music
drwx------+  3 bloc  staff   102 Mar 13 13:47 Pictures
drwxr-xr-x+  5 bloc  staff   170 Mar 13 13:47 Public
Blocs-MacBook-Pro:~ bloc$ 
```

> Our output of `ls -a -l` in Terminal

In the example, the `-a` instructs `ls` to "include directory entries whose names begin with a dot `.`". Any file that begins with a dot in Unix is a hidden file, so we instructed `ls` to display all hidden files. The `-l` flag tells `ls` to display the contents of the directory in long format. Use man pages to read more about the options for `ls` or any other command.

What Is Tab Completion?

The command line has a lot of keyboard shortcuts. One of the major keyboard shortcuts is tab completion. Try the following, remembering to press the **tab** key after typing `p` instead of pressing ↵ (return):

```
$ ls -l /bin/p
pax   ps    pwd
```

After hitting `tab`, the shell shows all files and directories in `/bin` that start with the letter 'p'. The shell also shows commands like `pax`, `ps`, and `pwd`. If the next character we type is 's' and hit `tab` again, then the shell will remove `pax` and `pwd` from the list:

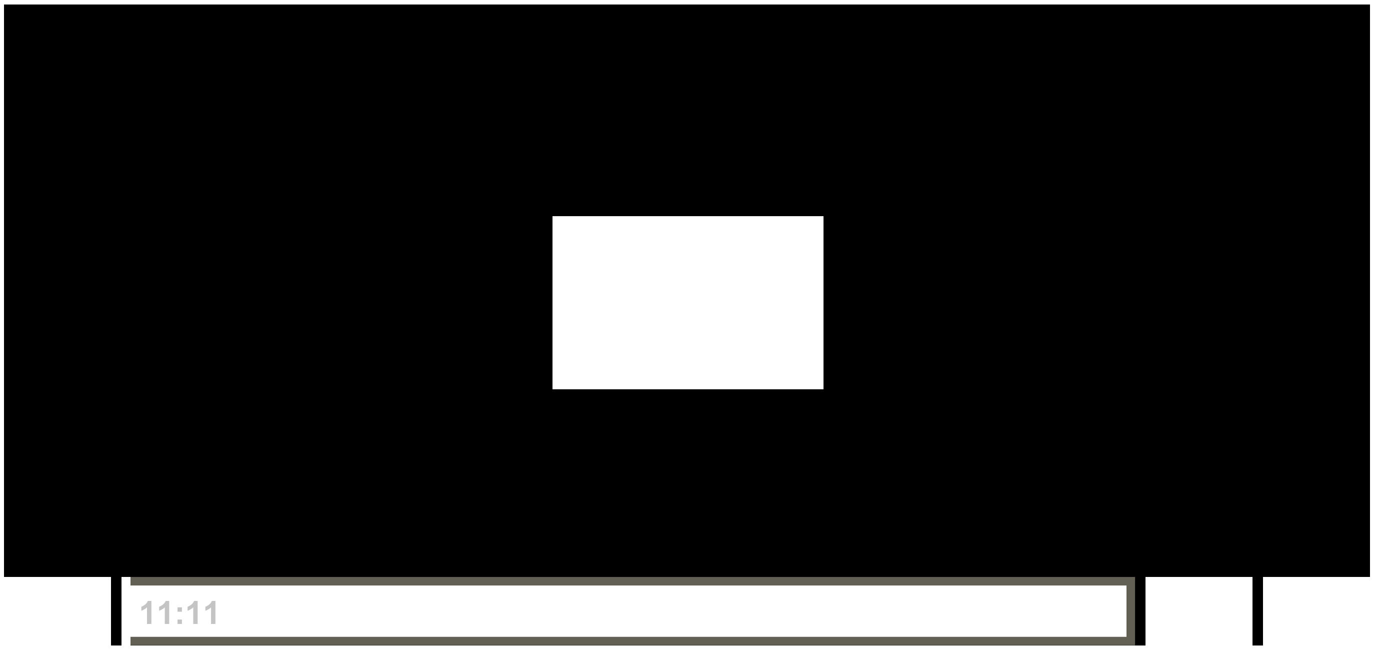
```
$ ls -l /bin/ps
-rwsr-xr-x  1 root  wheel  46688 Sep  9  2014 /bin/ps
```

Tab completion works with most Unix commands and is a handy feature.

How Can I Get More Help?

A large number of operations exist that we haven't covered in the checkpoint. For more information, we recommend:

- **The Command Line Crash Course**
- **Bash Guide**
- Our video demonstrating the commands in this checkpoint



Using the command line:

- Create a directory named `checkpoint_exercise` in your `HOME` directory.
- Navigate into `checkpoint_exercise`.
- Create a file named `bloc_cl_checkpoint.txt` in the `checkpoint_exercise` directory.
- Open and edit `bloc_cl_checkpoint.txt` so that it reads, "Hello World!"
- Print the contents of the file in your shell.
- Rename `bloc_cl_checkpoint.txt` to `cl_checkpoint.txt`
- Print the contents of the renamed file in your shell again.
- Copy and paste the output from your `history` command to show your mentor how you did it.

> - Hint (OS X): use `open bloc_cl_checkpoint.txt` to open the file in TextEdit.
> - Hint (Git BASH): use `notepad bloc_cl_checkpoint.txt` to open the file in NotePad.
> - Hint (Cloud9): use the sidebar file navigation to open the file in the Cloud9 Editor.
> - Hint: you can use the `cat` command to display the contents of a file.

Use **Markdown formatting** to format any code snippets you include in your description. Here is an example of how to format the `ls` command in markdown.

~~~
```bash
$ ls
```
~~~

assignment completed