

# Command Line Interface Bootcamp

Presented by ACM & WACM

Slides: <http://bit.ly/2C561ad>

If you're on Windows, make sure you're using Git Bash.

Download it at <https://git-scm.com/downloads>

*Command Prompt will not work.*

Linux/Mac users: you can use the Terminal application.

# Syllabus

CLI Introduction

Usage

file system navigation & manipulation

`alias` and `.bash_profile`

`curl`

`grep`

`diff`

`nano`

Java in the CLI

Scripting

I/O redirection and the Pipe

Processes

Q&A

# What is a command line interface?

You may think of a command line interface (CLI) as a means of interacting with your operating system's *kernel*.

You can perform various tasks (writing files, executing programs) that you normally would in computer usage, and many more with it, just with a *shell* instead of graphical user interfaces (GUIs).

Kernel = OS - GUIs

# CLI Alternatives

## **Text Editor**

E.g., Atom, Sublime

## **IDE (integrated development environment)**

Usually have debuggers and code completion

E.g. BlueJ, Eclipse, Visual Studio

## **Editors and IDEs are examples of GUIs (graphical user interfaces)**

“GUIs have the advantage of being able to visualize data in a meaningful way” —Konrad Rudolph

But using a CLI is often more efficient for programmers!

# Why use a CLI?

Workflow (efficiency, organization, speed )

Can be used to do things in a second that a single user may take hours to do in a GUI (e.g. with scripting)

Control

Users have more control over files and process (programs in execution)

Resources

CLI requires much fewer system resources than GUI

CLI vs GUI

# The File System

On Windows, you can look through your folders and files via File Explorer.

On a Mac, use Finder to do the same thing.

These are graphical representations of the *file system*.

With the CLI, we can navigate the file system via a text-based CLI instead, which is a major boon if you know what to do...

# file system navigation

After opening a new terminal window, we can print our current working directory (folder) with the `pwd` command:

```
$ pwd
```

```
/Users/myName
```

---

# file system navigation

View files and folders in the current working directory with the ls (“list”) command

```
$ ls
```

```
Applications  
Desktop  
Documents  
[...]
```

---



# file system navigation

To go into the Desktop directory (file), we can use the cd (“change directory”) command

```
$ cd Desktop
```

```
Desktop: myName$
```

---

# file system navigation

If we want to go back up a directory, we can use `cd ..`

```
Desktop: myName$ cd ..
```

```
~myName$
```

---

# file system navigation

Every directory has hidden pointers to files called `.` and `..`

`.` = pointer to the current folder

`..` = pointer to the above folder

Pointers that start with `.` are hidden, but we can look at them by adding `-a` to `ls`:

```
$ ls -a
```

---

# file system navigation

Finally, if we want to go back to my home directory from anywhere, we can use:

```
$ cd ~
```

---

# file system navigation

We can make a new directory with the `mkdir` command:

```
$ mkdir CLI
```

```
$ cd CLI
```

```
CLI myName$ mkdir bootcamp
```

---

# file system navigation

In CLI/, use `touch` to create a new file called `file.txt`:

```
$ touch file.txt
```

Move the file to a new directory with `mv`

Example:

```
$ mv file.txt ~
```

---

# file system navigation

Remove the file with `rm`

```
$ rm ~/file.txt
```

Remove an empty directory with `rmdir`

```
$ rmdir bootcamp
```

Remove a non-empty directory with `rm -R`

```
$ mkdir temp
```

```
$ touch temp/empty.txt
```

```
$ rm -R temp
```

---

# alias

“In computing, **alias** is a **command** in various **command line interpreters** (shells) such as Unix shells ... which enables a replacement of a word by another string.”

[https://en.wikipedia.org/wiki/Alias\\_\(command\)](https://en.wikipedia.org/wiki/Alias_(command))

We can make an *alias* to save our favorite custom commands.

**Syntax :**

```
$ alias myAlias='[command]'
```

**Example :**

```
$ alias goToDocs='cd ~/Documents/'  
$ goToDocs
```

---



# echo

```
$ echo Hello World
```

```
Hello World
```

The echo command will simply repeat whatever string follows it. (Hence the name)

---

# echo to a file, cat

Printing what we just typed is a bit boring.  
What if we could save it to a file instead?

```
$ echo "Hello World" >> hello.txt
```

Now display the contents with cat.

```
$ cat hello.txt
```

---

# curl

A tool for transferring data to or from a server using one of various protocols.

```
$ curl [url]
```

Retrieves the file given by the url.

Example:

```
$ curl www.something.com
```

Prints the HTML code that creates this page.

(Yes, this is something that somebody made)

---

# curl

Using the `-o` flag, we can output the contents of the result to a local file.

Syntax:

```
$ curl [url] -o file
```

Example:

```
$ curl  
http://mathcs.pugetsound.edu/~aasmith/cs361/alice.txt  
-o alice.txt
```

---

# grep

Searches any given input, printing lines that match one or more patterns.

Run `grep` to find lines containing references to the Mad Hatter. Be sure to do this in the directory containing `alice.txt`

```
$ grep Hatter alice.txt
```

Can be used in conjunction with *regular expressions*

---

# nano

view/edit text

Nano is a simplistic editor you can use without leaving the CLI.

```
$ nano alice.txt
```

Use arrowkeys to move the cursor, and backspace to delete characters.

Try replacing `Lewis Carroll` in the file with your name.

*To save a file, hit **ctrl + o** then enter*

*To exit nano, **ctrl + x**, enter (if you haven't made any changes)*

# diff

Difference operator  
on two I/O objects

```
$ diff [a] [b]
```

Will print the lines that are different between a and b.

Use the up arrow key to find your previous curl call to retrieve `alice.txt`, but output it to a new file called `alice2.txt`.

If you changed the author's name in `alice.txt`, what should this do?

```
$ diff alice.txt alice2.txt
```

---

# Bash Scripts

*A script* is a file with a list of commands to be executed sequentially.

Let's make a script to build a directory that you might use to organize your CS 261 content.

We're going to write this in `nano`.

---



# Bash Scripts

Work on a new script file:

```
$ nano script.sh
```

Add these lines to the file:

```
mkdir CS261
```

```
cd CS261
```

```
mkdir homework lab notes slides
```

```
echo "script.sh script completed"
```

---

# Bash Scripts

Finally, we can execute the script via

```
$ bash script.sh
```

---

# Redirecting I/O with < and >

We used `>>` earlier to *append* text to a file. We were actually redirecting the output of the command (`echo`) from the terminal into a destination.

`>` works the same way, but it *replaces* the contents in the destination with the output of the (left) command. (Try it on one of your non-important files)

`<` redirects *input* of the command to the right. Here's an example:

The `sort` command will print the lines of input in alphabetical order.

Try writing a file called `shopping-list.txt`. We can sort the lines via  
`$ sort < shopping-list.txt`

# Java compilation and execution

After we write a Java program, can compile it this way:

```
$ javac MyClass.java
```

Then run the compiled code (class) with

```
$ java MyClass
```

---

# Sample Java program

1. Create a file in nano called `Sample.java` with these contents.

```
1 public class Sample {  
2     public static void main(String[] args) {  
3         for (int i = 0; i < args.length; i++) {  
4             System.out.println(args[i]);  
5         }  
6     }  
7 }  
8
```

2. Compile it as we did in the previous slide.  
3. Now when we execute it, provide some arguments and redirect the output to a file like so:

```
$ java Sample command line args > out.txt
```

---

# The Pipe

`|` combines `<` and `>` into a single operation.

Example: Suppose I run the Java class `MyClass` and I want to compare the program's output against the professor's expected output.

```
$ java MyClass | diff expected-output.txt
```

How it works: `MyClass` has output that gets redirected into the pipe. `diff` then reads its input from the pipe (in addition to `expected-output.txt`)

Make sense? When it all does, you're on your way to becoming a great CLI user!

# less

Lets you scroll through output, instead of having to read it in the terminal output.

```
$ ls | less
```

As with `man`, press q to quit.

---

# Processes

Use `$ ps` to show what (terminal) *processes* (programs in execution) are running.

`$ ps -e` will show all of your kernel's processes.

Try `$ sleep 5`.

To use our terminal for other things while we wait, we can run it as a *background process* like this:

`$ sleep 100 &`



# Processes

You can add `&` to any process that will take a while, so you can do other stuff in the same terminal while you wait for it to finish.

If you want to close the terminal, you'll have to `disown` first.

Continuing the previous slide's example, `$ ps -e | grep sleep` will display processes that contain `sleep`, if there are any.

# Processes

What will this output?

```
$ ps -e | wc -l
```

*Exercise:* look up what `wc` does and figure it out.

# bash\_profile

You can save your aliases for later use in your `.bash_profile` file *if **bash** is your shell (terminal)*.

The equivalent file in other shells is usually called `.[shell]_profile` or just `.profile`.

Navigate to your home directory and run `ls -a`.

Hopefully, you'll see a file titled (something like) `.bash_profile`.

```
$ cd ~
```

```
$ ls -a
```

```
.bash_profile [ ... ]
```

# echo, source

In conjunction with `>>`, which places output in a destination, we can write to a file with `echo`. In one line:

```
$ echo "alias gtd='cd ~/Documents/'" >>  
~/.bash_profile
```

This will append the string containing the alias to the `.bash_profile`.

To use the alias without restarting the terminal:

```
$ source ~/.bash_profile
```

---

# man

If you'd like to learn more about any command, just enter `$ man [command]`

Example: try `$ man ls`

Scroll down to read more. Press q to quit.

---

# Unix tutorial

A very good place to start reading on your own about these commands is  
<http://www.ee.surrey.ac.uk/Teaching/Unix/>

# Telnet

```
$ telnet towel.blinkenlights.nl
```

(Use `ctrl-]` `ctrl-c` to exit)

If you want to learn what this is, take CS 325... for now, just enjoy.

```
$ cowsay "Thank you for listening"
```

< Thank you for listening >

-----

```
      ^  ^  
      _  
  \  (oo)\_____  
   \  (__)\       )\/\  
      ||----w |  
      ||     ||
```



# Shameless Plug

If you're eager to learn how to get a job as a software engineer, or are just interested in computer science, please attend:

Association of Computing Machinery (ACM)

~~Meets Sundays @ 5pm, Thompson 409~~

Women in the ACM (WACM)

**Meets Mondays @ 5pm, outside Thompson 409**

# Q&A, acknowledgements

Have fun with it!

## ***Special Thanks***

Prof. David Chiu (slide revision, promotion)

Prof. Adam Smith (promotion)

Alex Harris (poster, slides)

Gabe Pinkard (assistance)

Puget Sound ACM & WACM

The programmers of the UNIX OS (ca. 1970)

# Bonus knowledge: Git



(This was part of Spring '17's CLI bootcamp but has been deprecated)

# Git

Git is a commonly used version control tool.

Can be used to associate code directories with repositories (repos). Then it keeps track of all the changes.

GitHub is a site for hosting Git repos and shows how files have changed over time, and by whom.

# Git

Clone a repository from the internet into a directory with the repo name:

```
$ git clone [url].git
```

Or, initialize current directory as a Git repository:

```
$ git init
```

Pull new files from remote repository:

```
$ git pull
```

# Git

Add files to a commit to the remote repo:

```
$ git add [filename]
```

This can actually be any pointer in your rep! `$ git add .` is a common use case.

Commit your files to your repository:

```
$ git commit -m "your commit message"
```

Push your files to your repository

```
$ git push
```

# Git

Remove file from project:

```
$ git rm [filename]
```

Change filepath:

```
$ git mv [current_path] [new_path]
```

# Git

Show modified files in your working directory:

```
$ git status
```

Show what has been changed but not yet added/committed:

```
$ git diff
```

Show all commits in current active branch history:

```
$ git log
```



# Git

Learn more commands with this cheat sheet:

<https://education.github.com/git-cheat-sheet-education.pdf>