

Introduction to the Command Line

Mark Gross

Slides available →

<https://raw.githubusercontent.com/MarkusG/UCI-Slides/master/CLI.pdf>



About Me

- Mathematics student at Saddleback College
- Software Developer at Roland DGA
- Programming since 2016
- Using Linux since 2018
- In the top 10% of ranked Tetris players worldwide

Purpose of this Presentation

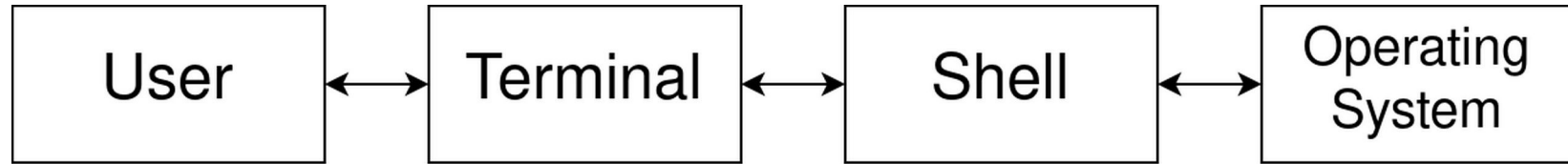
- **Showcase the power of the command line**
- **Give you an introductory knowledge of how to use it**

Why Use the Command Line?

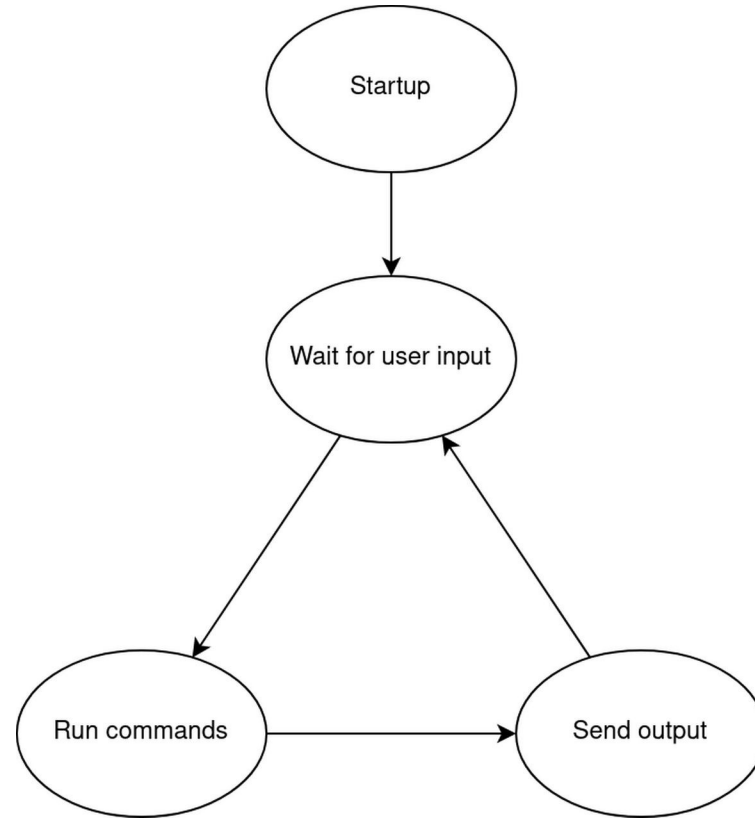
- **It's powerful**
 - “Graphical user interfaces make easy tasks easy, while command line interfaces make difficult tasks possible.” (TLCL xvii)
- **It's everywhere**

Some Quick Terminology

- **Terminal** - displays text and receives keystrokes from the user
- **Shell** - interprets and runs commands
- **Operating system** - handles the computer's resources
 - Responsible for spawning new programs



From the Shell's Perspective



Shells Across Operating Systems

- **Windows**

- Command Prompt/Windows Terminal
- PowerShell
- Windows Subsystem for Linux → bash

- **Linux and OSX**

- bash, zsh, fish, and countless others

Anatomy of a Shell Command

```
[jdoe@linux ~]$ ls -la --color=always some_directory
```

- **Prompt** – Tells you who and where you are
 - Current directory
- **Command** – The application to run
- **Short/long options** – One-time application settings
- **Argument(s)** – Tells the application what to operate on

Getting Help

- **man** - “manual” pages for commands (and more!)
- **whatis** - short, one-line help information

Working with the File System

- **ls** - “list” files/directories in the current directory
- **cd** - “change directory”
- **mkdir** - “make directory”
- **mv** - “move” a file/directory
- **rm** - “remove” - delete
- **ln** - “link” - shortcuts

- **Let’s try it!**

Kicking Things up a Notch with Expansions

- **The shell can evaluate/expand expressions**
 - `*` - “wildcard” expansion – matches files in the current directory
 - `{a..b}` – range expansion – `{0..3}` expands to `0 1 2 3`
 - `{a,b,c}` – set expansion – expands to `a b c`
 - `$((expr))` – arithmetic expansion - `$((1 + 1))` expands to `2`
 - Not too useful outside of scripts
- **Let’s try it!**

Input/Output

- **cat** - “concatenate” files
- **echo** - “echo” the argument back to you
- **grep** - “it’s a long story” - search for a pattern in a file
 - Regular expressions coming up in the next presentation!
- **wc** - “word count”
- **head** - view the first few lines of a file
- **tail** - view the last few lines of a file

- **Let’s try it!**

Redirection

- **The Unix Philosophy**

- Each program should do one thing
- The output of one program can be the input to another

- **How do we do it?**

- `cmd1 | cmd2` – “pipes” output of `cmd1` to the input of `cmd2`
- `cmd > somefile.txt` – writes the output of `cmd` to `somefile.txt`
- `cmd < somefile.txt` – runs `cmd`, taking input from `somefile.txt`
- `cmd1 | cmd2 | cmd3 | cmd4 | cmd5` ad infinitum

Scripting

- **The shell is a program like any other, so it can take input like any other!**
- **This allows us to write a sequence of shell commands and then tell the shell to execute them**
- **The shebang**
 - Describes what program to run, taking the rest of the file as input

`/bin/someprogram` **is equivalent to** `echo <stuff> | someprogram <stuff>`

Scripting - Variables

- **We can save the output of commands with variables**
- **Variables are accessed using \$ or \${}**

```
my_name="Mark"
```

```
greeting="Hello, ${my_name}!"
```

```
ls_output="$(ls)"
```

Scripting - Control Flow

- **if** - runs based on whether or not a condition is true
- **case** - works like a series of if statements
- **for** - a “for each” loop; runs once for everything in a given set
- **while** - runs while a certain condition is true

Scripting - test

- **if works based on exit codes**
- **We need a command to output exit codes based on what we want to compare!**

`test expression`

or

`[[expression]]`