Interfacing with the CLI...

...an opinionated guide, by Tao Tien.

me@taotien.com / tltien@dons.usfca.edu / Discord @taotien

What is the Command Line Interface?

Text is the purest and easiest way of interacting with the computer. When you open a terminal, you're opening a window into the soul of your system. Programs before graphical interfaces were introduced all used text to communicate with the user and each other. Most programs now still do.

All programs should accept some *arguments* and take `stdin`, and output to `stdout` and `stderr`. You'll learn what these concepts mean throughout this document.

The Setup

Before our fingers can even start flying over the keyboard, you need a *terminal emulator*. Personally, I use Wezterm, but feel free to look for one of your own choosing. The default one shipped with your system is usually quite barebones.

Navigation Basics

`ls`, `cd`, and *tab* are your friends. Whenever you open your terminal, it should put you in your *home* directory. If you now type `ls`, it should print out a list of directories where you're currently at. You'll probably see the usual suspects, `Desktop`, `Documents`, `Downloads`, etc.

Every command has *arguments / parameters* and *flags*. A flag usually selects functionality of a command, and arguments are the inputs to that command. For example, while `ls` defaults to listing the current directory's contents, you can also pass it a *path* and tell it to list what's there instead, like `ls Downloads`. Likewise, `cd`, which defaults to returning to home, also takes an argument for a directory you wish to change to. Some programs also take multiple arguments, whether a list of things to act on, or a source and a destination.

Flags are prepended with a single or two dashes, for short or long format arguments. `--help` or `-h` is your friend, as most programs will tell you how they work if you ask nicely. Flags can also take arguments of their own, and you can end up with long incantations like `tar -xvf file.tar -C Downloads/files`. Note how you can combine short flags, and/or have to split out ones that take arguments. lEttER cAse is also important, as the upper and lower case of the same character can be shared between functionality. Use a font that makes these things easy to distinguish!

Finally, paths have specific reserved characters that have special meaning. `.` (dot) means "here", `..` "parent", `~` "home", `-` "last". If I am in my project directory `/home/tao/cs315/hw/project01` and want to get to my lecture recordings at `/home/tao/cs315/lectures`, I can `cd ../../lectures`. Then, to get back to work, `cd -`.

command	description
`ls`	`(l)i(s)t`
`cd`	`(c)`hange `(d)`irectory
`mkdir`	`(m)a(k)e (d)`irectory
`cp`	`(c)o(p)y`
`mv`	`(m)o(v)e`, rename
`rm`	`(r)e(m)`ove

hotkey	description
tab	autocomplete
ctrl + c	interrupt, `(c)`ancel
ctrl + d	`(d)`isconnect, end
ctrl + shift + c	`(c)`opy
ctrl + shift + v	`(v)`paste

Getting Out of Dependency Hell

Provided with this guide is a file, `flake.nix`. It allows you to easily run every program used in this tutorial, without having to manually install everything. Otherwise, you'll have to figure out what I'm running based on the command shown. Nix is pretty magical, and a whole series of workshops unto itself, but all you'll need is to run the one-liner on this page.

Once you're done with that, you can download the file, `cd` to where it is, and run:

```
nix develop
```

The Cool Shit

If all this has been old hat for you, good. Here's where the fun begins.

If you've run the nix shell, then by all means start playing with the programs listed. Otherwise, you'll have to follow these links and install each thing manually.

Nushell

https://www.nushell.sh/

What you're interacting with in the CLI is actually the shell, which is a program that handles all the input and output. If you're on Linux, you're more than likely to be running `bash` by default. Mac users get `zsh` a *slightly* more modern extension of `bash`. Windows users often see the CMD command prompt, which is `DOS`-like, Powershell, or just `bash` if you're using the Windows Subsystem for Linux (WSL).

Bash is veeery old. It initially released FOURTY FIVE years ago! The scripting language is hard for beginners, and has many footguns. I still recommend learning it (although I have not), as many scripts and servers are guaranteed to be written in bash or have only bash available to you.

Nushell is brand spankin new, fast, and intuitive. You can use it like a calculator. The built-in command output is pretty, and errors are clear and understandable.

starship

https://starship.rs/

This just adds nice features to your prompt line and makes things pretty.

Helix
https://helix-editor.com/

Batteries included modal editor. A great introduction to modal editing. Neovim that you can learn to use (and exit lmao), and without the need of hours of configuration.

jujutsu
https://github.com/jj-vcs/jj

Your classes will eventually have you use `git`. I say, reject that nonsense and use the wonderful, intuitive, clean, and git-compatible version control system.

zoxide
https://github.com/ajeetdsouza/zoxide

No more `cd`-ing around manually. `cd` a bunch of times once, `z` to wherever, from wherever. For example, instead of having to type `cd school/cs315/hw/project01` or `cd school` + `cd cs315`..., instead I can just `z t01`.

direnv
https://direnv.net/