

Beginning the UNIX command line

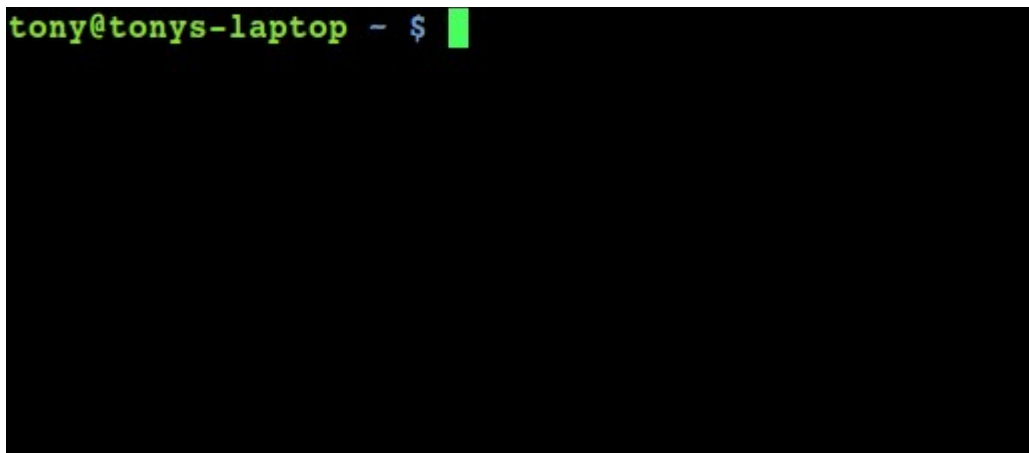
This page will hopefully help teach you a little bit about interfacing with and understanding the command line.

Unix commands are designed to do one thing and do that thing great. We will see why this philosophy makes the shell so powerful.

These tutorials are designed for BASH, and could probably work with other shells with minor modifications.

Wait, what is BASH you say? It stands for **B**ourne **A**gain **S**hell. A shell is a program that interprets user commands, i.e. a command line. There are many types of shells, each with subtle differences. To see what type of shell you are using type `echo $SHELL`

But first lets take a look a the terminal:

A screenshot of a terminal window with a black background. The prompt 'tony@tonys-laptop - \$' is displayed in green text at the top left. A green cursor is positioned at the end of the prompt.

- “tony” is my username
- “tonys-laptop” is the name of my computer
- “~” is the folder I'm currently in ('~' is a shortcut to your home folder)
- “\$” is the end of the “prompt”

So now let's see what shell we are running:

```
tony@tonys-laptop - $ echo $SHELL
/bin/bash
tony@tonys-laptop - $ █
```

- “echo” is the standard print command
- “\$SHELL” is a variable that stores the location of the executable for the current shell
 - Variables are very useful in the shell since they can represent files, folders, URLs, anything really.
 - To view the currently set shell variables run `env`
 - Here's an example of using variables:

```
tony@tonys-laptop - $ echo $name

tony@tonys-laptop - $ name=tony
tony@tonys-laptop - $ echo $name
tony
tony@tonys-laptop - $ name=rogers
tony@tonys-laptop - $ echo $name
rogers
tony@tonys-laptop - $
```

- Notice how when **setting** variables you don't include the \$
- “/bin/bash” is the location of the executable for the current shell, in this case 'bash'
 - The '/bin' folder is where all the core utilities reside
 - Run `ls /bin` to view them
 - A few notable commands in there that we will cover:
 - echo → prints stuff
 - ls → list contents of current folder (or a given one)
 - cat → dumps file contents to console (*meow*)
 - man → interactive manuals for *every* command (your friend)

- curl → dumps contents of a webpage to console
- grep → searches input for certain patterns (very powerful)
- sed → edits streams of text without human interaction

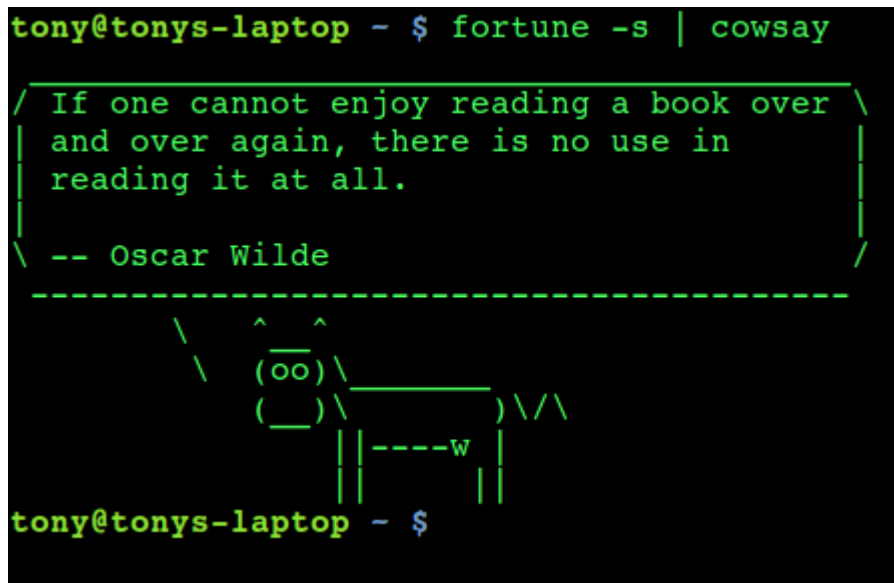
Now for a practical(ish) example

You may have seen the URL scraper on my website, if not that's cool, I think it's broken right now anyways.

You give it the URL of a webpage and it spits any URLs in that page. Pretty simple, yet interesting, if you think about it, eventually it could map the internet.

So how did I make this? Nothing more than a few the above commands and couple pipelines.

Wait what's a pipeline? A pipe is a way to link the output of one command to the input of another. Like so:



```
tony@tonys-laptop - $ fortune -s | cowsay
/ If one cannot enjoy reading a book over \
| and over again, there is no use in      |
| reading it at all.                      |
\ -- Oscar Wilde                          /
-----
      ^__^
      (oo)\_______
      (_____)       )\/\
      ||----w |
      ||     ||

tony@tonys-laptop - $
```

- “fortune” is a command that prints some quote or humor
- “cowsay” is a command that makes a cow say its input
- “|” is the pipeline (who woulda thought people actually used that?)

- What this does is take the output of the “fortune” command, and make it the input of the “cowsay” command
 - This is where the Unix philosophy of “do one thing and one thing well” comes in. It's designed as bunch small, flexible components that can be easily connected to create a much larger result.

Back to the URL scraper

So we want to get the content of a webpage, scan it for certain patterns, and print out a list. You could do this by hand, tediously, or write a quick script that does this for you.

Step 1: Get the webpage

- curl is the command for that.
- So let's run `curl www.reddit.com`
- You were probably not ready for that...
- How are we ever going to get useful info from that?

Step 2: Search for patterns.

- So running `curl www.reddit.com` prints a lot of stuff out but we only URLs...
- grep is the tool for that.
- grep searches input for specified patterns called regular expressions
 - Regular expressions (or regexes) are fucking hard to understand, they look like a jumble of characters, numbers, and symbols that can't possibly mean something, but they are one of the most powerful concepts in the terminal.
- So how do we use grep?
- `curl www.reddit.com | grep [REGEX]`
 - [REGEX] will be a regular expression
- So let's start building our regex...
 - We know that all most URLs start with **http://**
 - This will be our *anchor*

- After **http://** there is any number of alpha-numeric characters and dots and slashes, for example:
http://www.thisisarealsite123.co.uk/home/stuff/
- Here's what the regex would look like:
- `http://[A-Za-z0-9\./]*`
- “[]” will match any single character in the group
 - [A-Z] will match any character between A and Z
 - In this case the group contains the ranges A-Za-z0-9 and a dot or slash
 - The dot must be preceded by a ‘\’ because it is significant in regexes and must be escaped
- “*” will match any number of the preceding character (or group)
 - A* will match any number of A's
 - [A-Z]* will match any number of any character between A and Z
 - [A-Za-z0-9\./]* will match any number of alpha-numeric characters and dots and slashes
- Try running:
 - `curl -s www.reddit.com | grep “http://[A-Za-z0-9\./]* -o”`
 - This should spit out almost every URL in the page
 - However, this won't find any https pages or pages that contain non alpha numeric characters. See if you can figure how to make it!
 - I added the -s option to curl to suppress its progress bar
 - I added the -o option to grep to only print out matched portions

Step 3: Sorting and formating.

- So now we have a command that will spit out every* URL in a webpage.
 - This output is not sorted and there may be a lot of duplicates
- Guess what? There are commands that do each of those!
 - sort → sorts input in alphabetical (or other) order
 - uniq → removes duplicate lines from sorted list
- `curl -s www.reddit.com | grep "http://[A-Za-z0-9\./]*" -o | sort | uniq`
 - By piping the command through sort then uniq it will output an alphabetically sorted list of URLs with duplicates removed

Step 4: Applying it.

- We now have a command that prints out all URLs found in [“www.reddit.com”](http://www.reddit.com)
 - What if we want to make it so the user can pick which site to scrape URLs from?
 - Shell scripts are the answer for that
 - You can group together shell commands
 - You can pass in arguments
 - Here's an example using the nano text editor:

```
GNU nano 2.2.6                               File: ./url-scrape-example.sh
#!/bin/bash
curl -s $1 | grep http://[A-Za-z0-9\./]* -o | sort | uniq
exit 0
```

- First create a file called [nameofscript].sh
 - The .sh extension isn't vital but will help you remember it's a script
- “#!/bin/bash” will make it run in the bash shell if not specified
- Notice the \$1 where the URL used to be, this represents the first argument in the script for example:
 - `url-scrape-example.sh www.reddit.com`
 - There are few reserved variables in every script
 - \$# → number of arguments passed
 - \$@ → list of all arguments
 - \$N → the Nth number arguments
 - \$? → the return code of the last run command (used to check if something failed)
- “exit 0” tells the script to end with a return code of 0 (everything went fine)
- To run this simply type `bash /path/to/url-scrape-example.sh [URL-to-scan]`

There you go you just wrote your first Shell script. That wasn't so hard, was it?

There are still a few things I didn't touch on that may be worth a google:

- **Toggling executable bit on a script, removing .sh, and using `#!/bin/bash`**
- **Using the man pages for help, I highly recommend checking out the man page for each command we used. Type `man [command]` to view the man page.**
 - **Every command has many options that can make them more flexible or useful. Some are also easter eggs.**
- **REGULAR EXPRESSIONS**
 - **Fucking learn these, you will be able to search for anything.**