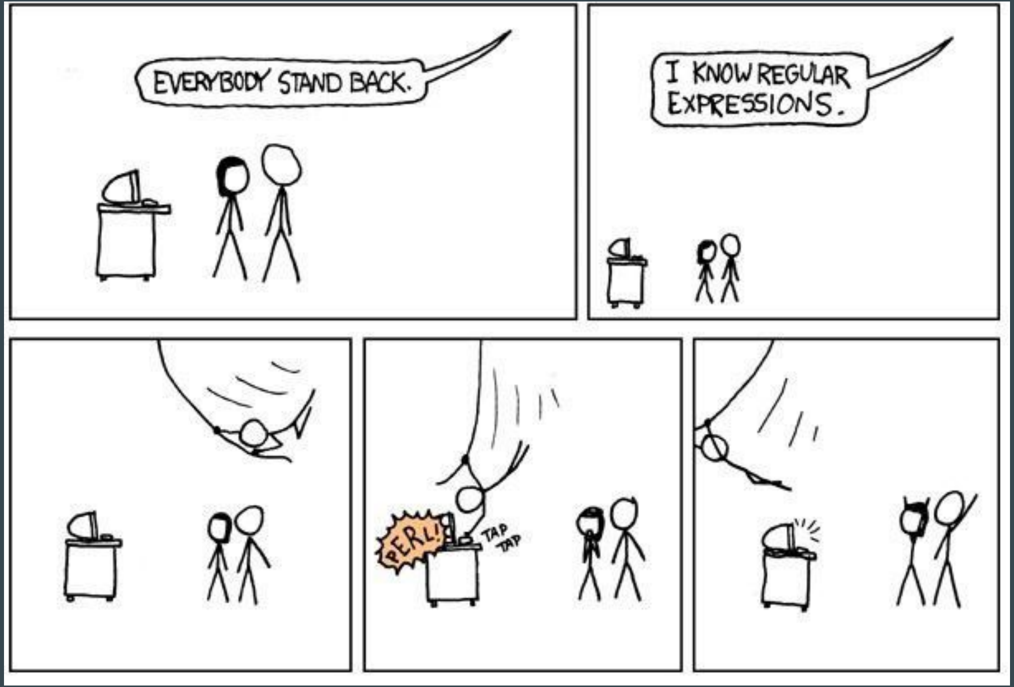
Basic Command Line Toolbox

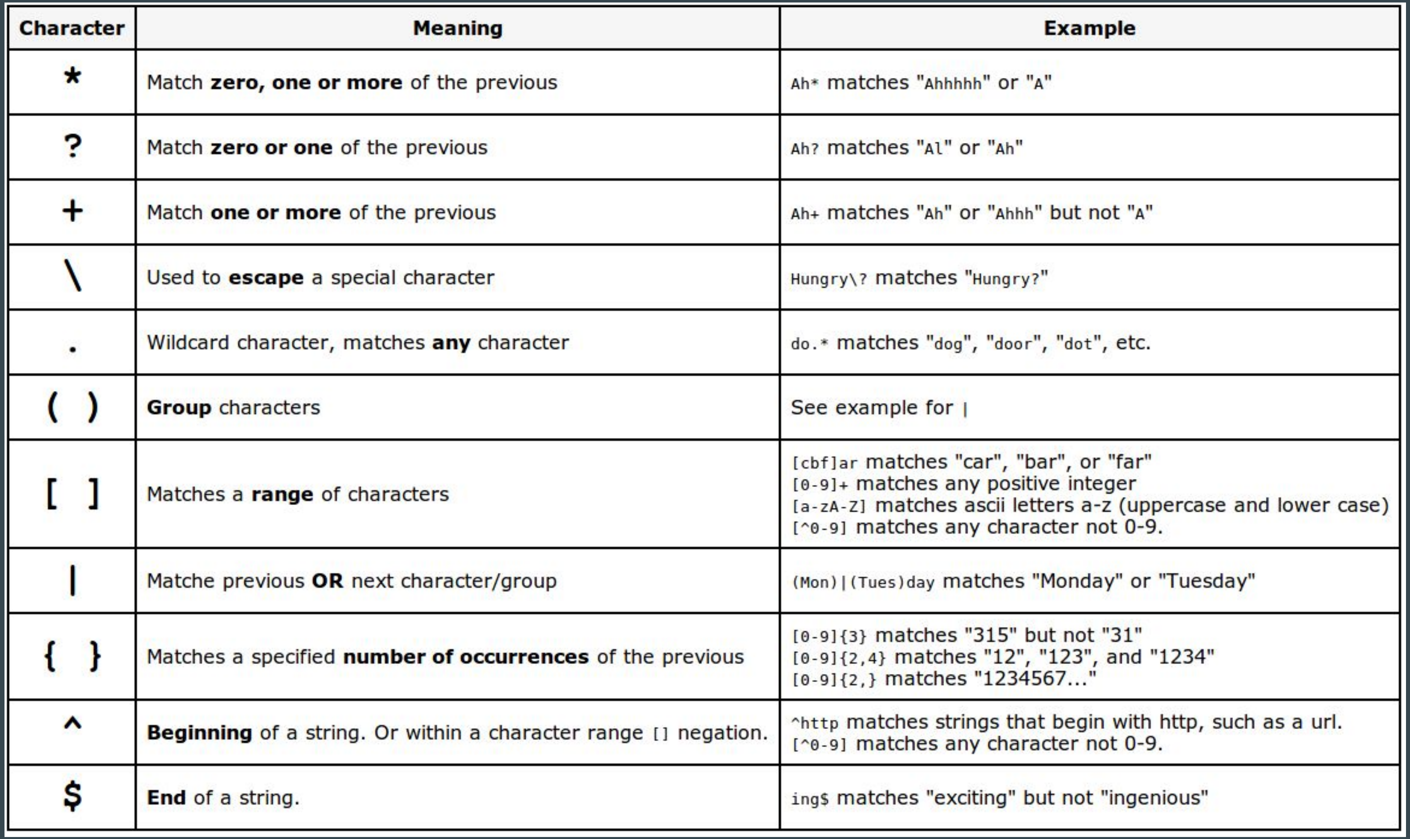
Cool tools for bash

First … a little about regular expressions

Regular expressions are super handy in coding, and also in navigating and using the command line powerfully



Special characters



Character	Meaning	Example
*	Match **zero, one or more** of the previous	`Ah*` matches "`Ahhhhh`" or "`A`"
?	Match **zero or one** of the previous	`Ah?` matches "`Al`" or "`Ah`"
+	Match **one or more** of the previous	`Ah+` matches "`Ah`" or "`Ahhh`" but not "`A`"
\	Used to **escape** a special character	`Hungry\?` matches "`Hungry?`"
.	Wildcard character, matches **any** character	`do.*` matches "`dog`", "`door`", "`dot`", etc.
( )	**Group** characters	See example for `|`
[ ]	Matches a **range** of characters	`[cbf]ar` matches "car", "bar", or "far" `[0-9]+` matches any positive integer `[a-zA-Z]` matches ascii letters a-z (uppercase and lower case) `[^0-9]` matches any character not 0-9.
|	Matche previous **OR** next character/group	`(Mon)|(Tues)day` matches "Monday" or "Tuesday"
{ }	Matches a specified **number of occurrences** of the previous	`[0-9]{3}` matches "315" but not "31" `[0-9]{2,4}` matches "12", "123", and "1234" `[0-9]{2,}` matches "1234567..."
^	**Beginning** of a string. Or within a character range `[]` negation.	`^http` matches strings that begin with http, such as a url. `[^0-9]` matches any character not 0-9.
$	**End** of a string.	`ing$` matches "exciting" but not "ingenious"

Making things appear

`touch`

Create a file (or update the timestamp on an existing file)

`mkdir`

Create a directory (use `-p` to create a nested directory)

`ln`

Create a link to another file or directory (with `-s` to create soft link)

Making things go away

`rm`

remove a file or directory (use `-r` to remove recursively, and `-f` to remove populated directories without answer any prompts)

Using output of one command as input to another

Use the pipe ‘|’ symbol to use the output of one command as the input of another command

```
ls | grep part_of_file_name
```

To search for a file in a directory

```
dpkg -l | grep some_package
```

To see if a particular package is installed on a linux system

Redirecting output to a file

`>`, `>>`

Redirects output from command on the left to file on the right.
A single > replaces any content in the file. A double >> appends to the file.
Examples:

```
dpkg -l > installed_pkgs.txt
echo “PATH=$PATH:~/app/bin” >> ~/.bashrc
```

`tee`

Pipe output to the tee command to redirect it to a file AND print to stdout. Use `-a` to append output to the file (otherwise it overwrites the file contents).
Example: `apt install some-package | tee log.txt`

Two kinds of output - stdout and stderr

Output of most commands/scripts goes to `stdout` (standard out)

Error output of most commands/scripts goes to `stderr` (standard error)

> `stdout` has the descriptor ‘1’
> `stderr` has the descriptor ‘2’

So we can redirect them to different files or commands if we want, for example:

```
./script.sh 2>errors.txt | tee log.txt
```

Or we can redirect stderr to stdout, which will add stderr output to stdout output:

```
./script.sh 2>&1 | tee log.txt
```

/dev/null - the bitbucket

We can send output to /dev/null if we just want to throw it away - that’s why we call it the bitbucket.

This is useful if you want to avoid having output of a command print to the screen, or want to ignore error output.

Example:

```
./script.sh >/dev/null
```

Looking at things

`cat`

Concatenate a file

`head`

Concatenate first 10 lines of a file (or use `-n` to specify more/less lines)

`tail`

Concatenate last 10 lines of a files (or use `-n` to specify more/less lines, or `-f` to continuously concatenate a file that's being updated, like a log file)

Finding things

`grep`

Find words in a file

`find`

Find files in directories

Filtering and formatting output

`awk`

> `awk` is super handy for filtering and formatting output. It is a whole language, and is really powerful. A simple and common use is to print particular fields of output to get a value.

Example: Get the name of the wireless network from the output of `iwconfig`

```
iwconfig 2>/dev/null | grep ESSID | awk -F'"' '{print $2}'
```

Build your toolbox and use the tools

Try writing a script to automate something you do every day - like some git interaction for example, or some build process you perform all the time.

Whenever you find yourself running some commands regularly, put them in a script to save time. It’s the best way to learn scripting, commands, and to get comfortable with the CLI!