**What is a Unix-style shell?**

Compatible with shells used on Linux and other Linux systems (majority of web servers run on)

**Git Bash** is the most popular Unix-style shell for Windows

* **Terminal** is an **interface to the Shell** and to other command line programs that you run
  + Similar to how web browsers is an interface to websites and other services
* You can have multiple instances of terminal open at once – each one w/ its own instance

**First Shell Command (echo)**

* Get Shell to **print messages** (console.log in JS or print () in Python)
* There are characters that have special meaning in the Shell – the **exclamation mark (!)**
* **Dollar symbol ($)** indicates that it is a Shell variable
  + echo $COLUMNS x $LINES prints out **size** of terminal

**Navigating Directories (ls, cd, ..)**

**Directory** (CLI term) and **folder** (GUI term) are basically the same thing

The command **ls** (list):

* List the contents of a directory in the Shell
* Interacts with the **current directory**

The command **cd** (change directory)

* Changes to a different folder
* *cd ..* goes back a directory (parent directory)
* **Semicolon (;)** lets you write two commands on the same line

**Print Working Directory (pwd)**

The working directory is another word for whatever directory the shell is currently looking at

* Unix system uses forward slash to separate the names of directories
  + String composed of names separated by slashes is the **path**
* **dot dot (..)** is the **parent directory** while **dot (.)** is the **current directory**
* **Tilde (~)** stands for my **home directory**

**Shell Parameters and Options**

Many Shell commands also support options, or flags, or parameters that turn on extra behavior

**ls -l** lists the content of that directory in long format

Shell allows for match filenames with a **pattern**

* *ls -l Documents/\*.pdf* will list all documents that are pdfs
* *ls bear\** will list all files whose names start with the word bear

**Organizing Your Files (mkdir, mv)**

*mkdir Documents/Books* will make a folder calledBooks in the parent folder Documents

*mv item\_from* item\_to

*mv Documents/\*.epub Documents/Books* will move all filenames .epub in Documents to Books

*mv ‘Documents/Books/\*’ Documents* is telling the Shell to look for file whose name is **literally** Documents/Books/\* so this will not work in the example problem

Go for **single quotes** in the Shell if you’re unsure between double or single quotes

**Download File From the Web (curl)**

**Curl** stands for **C URL → see URL**

*Curl -L ‘http://google.com’* will follow redirects to see the source code of google.com homepage

Curl is useful for **downloading files** or **pages by URL**

*curl -o google.html -L ‘http://google.com’*

* The *-o google.html* will save under the name google.html
* *-L* will follow all redirects of google.com

Good habit to put single quotes around URLs whether or not they would be correct without them

**Viewing Files (cat, less)**

**cat (short for concatenate)** reads the file and outputs the contents (can read any num. of files)

**less** shows one screenful of the file at a time (quit with q)

* Space or arrow keys to scroll down
* B to go back
* Slash to search

**Removing Things (rm, rmdir)**

**rm (remove)** will remove the file and not place them in the Trash Bin

* *rm -i* (i for interactive) will prompt you for every file it’s going to remove

**rmdir (remove directory)**

*rm \*’Bad F’\** will remove all files that have the string *Bad F*

**Searching and Pipes (grep, wc)**

**grep** searches a text file for lines with particular content

*grep shell dictionary.txt* reads the file and outputs all lines that contain that word

*grep shell dictionary.txt | less* vertical bar character (pipe) will output to another command (in this case grep’s output is the input to less)

*curl -L* [*https://tinyurl.com/zeyq9vc*](https://tinyurl.com/zeyq9vc) *| grep fish* will take the output of curl and give it as the input to grep (in this case will find all occurrences of fish)

**How matches were there?**

* Use wc -l where -l means to count lines
* grep -c fish where -c means count

Can use grep with **regular expressions (regexes)**

**Shell and environment variables**

numbers=’one two three’

echo $numbers → one two three

For **variables**, there must be **no space** between the equal sign

**Shell Variables** ($COLUMN or $LINES) **vs. Environment Variables**

* Shell variables are internal to the shell program itself
* Environment variables are shared with programs that you run within the shell
  + The $PATH variable (tells system where program files are)

**Add a Directory to Path (so that programs in it can be found)**

PATH=$PATH:/new/dir/here

* If you do it within the shell prompt, that change will only last until you close the shell

**Startup Files (.bash\_profile)**

Files containing Shell commands are called Shell scripts

Putting that directory into the Shell’s configuration file to make it **sticky**, last even if you close

For Mac, Windows, the Shell will run the commands in a file called **.bash\_profile**

For Linux, **.bash\_profile** will only run for some Shell sessions, specifically the login Shell sessions. Non-login Shells run a file called **.bashrc** instead

Any command you put in this config. file will be run every time you start the shell

* Include things like variable assignments like changing dollar sign path
* Anything you’d like to see at the start of your Shell session (date or friendly message)

**Look up the Bash Manual (man bash)**

Control the Shell prompt by putting the code in the **PS1 shell variable** (in your .bash\_profile)

* Use bashrcgenerator.com to really customize your Shell Prompt
* Use PS1 = ‘$ ‘ to keep it simple

**Aliases (make long shell commands shorter)**

*alias ll=’ls -la’* the alias command **does not** like spaces

* To see all the aliases that you already have, run the alias command with no arguments
* Aliases will only last as long as the Shell is open for
* If want them sticky, put them in the .bash\_profile

**Shell Resources to Keep Learning**

* The Bash Academy
* Bash Beginners Guide
* Bash Programming HOWTO
* Regexr - Learn Regular Expressions
* mywiki.wooledge.org/BashGuide
* tldp.org (Advanced Bash-scripting Guide)

**Types of Commands**

1. An **alias** is a way to shorten a command (only used in **interactive** shells and not in scripts). An alias is a word that is mapped to a certain string.
2. A **function** can be used in **scripts**; it contains Shell commands, and can take arguments and create local variables. When a function is called, the commands in it are executed
3. **Builtins** are basic commands that are built into Bash (*cd, rm, echo, etc..)*. They are basically functions that are already provided.
4. **Keywords** are like builtins, with the main difference being that special parsing rules apply to them. *[* is a Bash builtin, while *[[* is a Bash keyword.

**Special Characters in Bash**

A group of characters, when used, are evaluated by Bash to have a non-literal meaning

* “ ” Whitespace - to determine where words begin and end
* ‘ ’ Single quotes - protect the text inside them so that it has **literal** meaning
* \ Escape - prevents the next character from being interpreted as a special character
* # Comment - notes of explanation and are not processed by the shell
* [[]] Test - an evaluation of a conditional expression to determine “true” or “false”
* >< Redirection - redirect a command’s *output* or *input*
* | Pipe - redirects output from an initial command to the input of secondary command

**What are Parameters?**

**\*\*\*\*\*\*\*\*\***You should **always** keep **parameter expansions properly quoted**. This prevents the whitespace or the possible globs inside of them from giving you gray hair or unexpectedly wiping stuff off your computer. The only good PE, is a quoted PE.**\*\*\*\*\*\*\*\*\***

* A sort of named space in memory you can use to retrieve or store information.
  + Two types of parameters: *variables* and *special parameters*
  + Variables are parameters that you can create and update yourself
  + Special parameters are read-only, pre-set by BASH, and used to communicate some type of internal status
* A few examples of *variables* that the Shell provides for you:

1. **HOSTNAME**: contains the hostname of your computer
2. **PWD**: contains the current working directory
3. **RANDOM**: generate a (pseudo)random number between 0 and 32767
4. **COLUMNS**: number of characters that fit on one line in your terminal
5. **LINES**: number of lines that fit in your terminal
6. **HOME**: the current user’s home directory
7. **PATH**: colon-separated list of paths that will be searched to find a command, if not an alias, function, builtin command, keyword
8. **PS1**: contains a string that describes the format of your shell prompt

**Variable Types**

* **Array:** declare -a variable: array of strings
* **Associative Array:** declare -A variable: associative array of strings
* **Integer:** declare -i variable: holds an integer (triggers arithmetic evaluation)
* **Read Only:** declare -r variable: can no longer be modified or unset
* **Export:** declare -x variable: marked for export (it will be inherited by any child process)

**Parameter Expansion**

Any operation that causes a parameter to be expanded (replaced by content).

*$ echo “‘$USER’, ‘$USERs’, ‘${USER}s’”*

*‘Lhunath’, ‘’, ‘lhunaths’*

The second PE results in an empty string because the parameter *USERs* is empty. You need to use curly braces to mark the beginning and end of the parameter name as seen in third PE

*$ for file in \*.JPG \*.jpeg* used to rename all JPEG files with a .JPG or a .jpeg ext. *do mv -- "$file" "${file%.\*}.jpg"* to have a normal .jpg extension. Expression *${file%.\*}* cuts *done* everything from the end starting with the last period (.)

**Good Practice for Parameter Expansions:**

You may be tempted to use external applications such as *sed, awk, cut, perl* or others to modify your strings. Be aware that **all of these require** an **extra process** to be started, which in some cases **can cause slowdowns**. Parameter Expansions are the perfect alternative.

**Bash Patterns**

Serves two roles in the Shell: selecting filenames within a directory, or determining whether a string conforms to a desired format**.** First type of pattern is the use **globs** (form of patterns that can easily be used to match a range of files, or to check variables against a simple rule). Second type of pattern involves **extended globs**, which allow for more complicated expressions. **Regular expressions** are also supported in Bash.

* Globs are basically patterns that can be used to match filenames or other strings
* Metacharacters have special meanings:
  + **\***: matches any string, including the null string
  + **?**: matches any single character
  + **[...]**: matches any one of the enclosed characters

A glob must **match** the whole string (filename or data string)

* When globs match patterns, the / restriction is removed

*$ for file in \*; do rm "$file"; done  
$ ls*

* Glob expands in the proper form and removes everything in the directory
* Also be used to check whether data matches a specific format

*$ filename="somefile.jpg"* [[ keyword and the case keyword offer opportunity *$ if [[ $filename = \*.jpg ]]; then* to check string against a glob - either regular globs, *> echo "$filename is a jpeg"* or extended globs *> fi  
somefile.jpg is a jpeg*

**Extended Globs**

By default are turned off, but can be used with *shopt* command: *shopt -s extglob*

* ?(list): Matches zero or one occurrence of the given patterns
* \*(list): Matches zero or more occurrences of the given patterns
* +(list): Matches one or more occurrences of the given patterns
* @(list): Matches one of the given patterns
* !(list): Matches anything but the given patterns

The list inside parentheses is separated by the | character

*$ ls* Expand to anything that does not match the \*jpg or *names.txt tokyo.jpg california.bmp* the \*bmp pattern. The .txt file is then expanded. *$ echo !(\*jpg|\*bmp)  
names.txt*

**Regular Expressions**

Similar to globs, but they can only be used for pattern matching, not for matching filenames

**Tests and Conditionals**

1. **Exit Status**
2. **Control Operators (&& and ||)**
3. **Grouping Statements**
4. **Conditional Blocks (if, test and [[)**
5. **Conditional Loops (while, until and for)**
6. **Choices (case and select)**