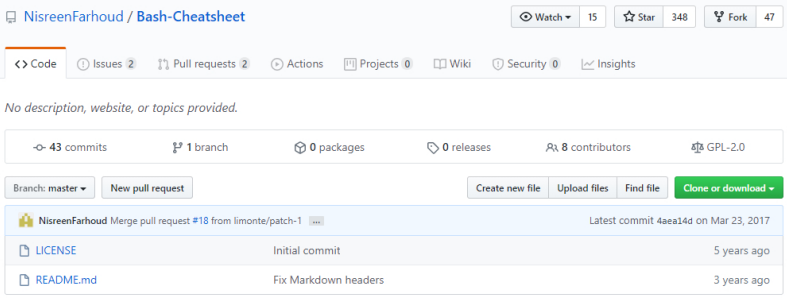

README.md

Bash Shell Cheatsheet

The main topics of this cheatsheet include an intro to the shell, navigating around the shell, common commands, environment variables, connectors, piping, I/O redirection, permissions, and keyboard shortcuts.

Introduction to the Shell

The shell is a program, in our case, called "bash" which stands for Bourne Again Shell. How the shell works is it takes your commands and gives them to the operating system to perform. In order to interact with the shell, we use "terminal emulators" such as the gnome-terminal, eterm, nxterm, etc.

Navigating Around The Shell

On a Linux system, files are organized in a hierarchical directory structure. This means there is a starting directory called the root directory. This directory contains files and subdirectories that lead into other subdirectories.

pwd

The `pwd` command, short for print working directory, displays your current location in the directory structure.

cd

The `cd` command allows you to enter a new directory.

Syntax	Explanation
cd	navigate to home directory
cd ~	navigate to home directory
cd ..	navigate backwards to parent directory
cd -	navigate to previous working directory
cd Directory1	navigate to directory named Directory1
cd Directory1/Directory2	navigate to directory, Directory2, through path

mkdir

The `mkdir` command makes a new directory in your current directory.

Common Commands

man

The `man` command directs you to the command manuals.
For example, the following command gives us all the information we need about the command `cat`.

```
$ man cat
```

cat

The `cat` command reads a file passed as a parameter and by default print its contents to standard output.
Passing multiply files as parameters concatenates the files and then prints to standard output.

echo

The `echo` command prints its arguments to standard output.

```
$ echo Hello World
Hello World
```

If you call `echo` without any parameters, the command prints a new line.

head

The `head` command reads the first 10 lines of any passed in text and prints its contents to standard output. You can change the default 10 lines to any number by manually passing in the desired size. For example, the following prints all 50 lines of the file.

```
$ head -50 test.txt
```

tail

The `tail` command reads the last 10 lines of any passed in text and prints its contents to standard output. You can change the default 10 lines to any number by manually passing in the desired size. For example, the following prints all 50 lines of the file.

```
$ tail -50 test.txt
```

You can also view in real time any text appended to the file with the `-f` flag.

```
$ tail -f test.txt
```

less

The `less` command gives you a way to navigate through a passed file or block of text. Unlike the `more` command, less allows you to move backward through the file as well.

```
$ less test.txt
$ ps aux | less
```

Common less keyboard shortcuts	Description
`G`	Moves to end of file
`g`	Moves to beginning of file
`:50`	Moves to the 50th line of the file
`q`	Exits less
`/searchterm`	Searches for any string matching 'searchterm' below the current line
`/`	Moves you to the next match for your previous 'searchterm' below the current line
`?searchterm`	Searches for any string matching 'searchterm' above the current line
`?`	Moves you to the next match for your previous 'searchterm' above the current line
`up`	Moves up a line
`down`	Moves down a line
`pageup`	Moves up a page
`pagedown`	Moves down a page

true

The `true` command always returns the exit status zero to indicate success.

false

The `false` command always returns the exit status non-zero to indicate failure. ### `$?` `$?` is a variable that will return the exit code of the last command you ran.

```
$ true
$ echo $?
  0
$ false
$ echo $?
  1
```

grep

The `grep` command is a search function.
Passing a string and a file searches the file for the given string and prints the occurrences to standard output.

```
$ cat users.txt
user:student password:123
user:teacher password:321
$ grep 'student` file1.txt
user:student password:123
```

`grep` can take multiple files as parameters and regular expressions to specify a pattern in text.

Common flags	Description
`-i`	remove case sensitivity
`-r`	search recursively through directories
`-w`	search only whole words
`-c`	prints number of times found
`-n`	prints line found on with phrase
`-v`	prints invert match
See regex tutorial	

sed

The `sed` command is a stream editor that performs text transformations on an input.
Common use of this command is to replace expressions which takes the form `s/regexp/replacement/g` For example, the following replaces all occurrences of the phrase "Hello" with "Hi".

```
$ cat test.txt
Hello World
$ sed 's/Hello/Hi/g' test.txt
Hi World
```

See sed tutorial

history

The `history` command prints out an incremented command line history.
It is common to use the `grep` command with the `history` command in order to search for a particular command. For example, the following searches your history for all occurrences of the string `g++`.

```
$ history | grep g++
155  g++ file1.txt
159  g++ file2.txt
```

export

The `export` command sets an environment variable to be passed to child processes in the environment.
For example, the following exports the variable "name" with the value "student".

```
$ export name=student
```

ps

The `ps` command, short for process status, prints out information about the processes running.

```
$ ps
PID TTY          TIME CMD
35346 pts/2    00:00:00 bash
```

There are four items displayed:

- process identification number (PID)
- terminal type (TTY),
- how long process has been running (TIME)
- name of command that launched the process (CMD)

awk

The `awk` command finds and replaces text by searching through files for lines that have a pattern.
Syntax: `awk 'pattern {action}' test.txt`

wget

The `wget` command downloads files from the web and stores it in the current working directory.

```
$ wget https://github.com/mikeizbicki/ucr-cs100
```

nc

The `nc` command, short for netcat, is a utility used to debug and investigate the network.
See nc tutorial

ping

The `ping` command tests a network connection.

```
$ ping google.com
PING google.com (74.125.224.34) 56(84) bytes of data.
64 bytes from lax17s01-in-f2.1e100.net (74.125.224.34): icmp_req=1 ttl=57 time=7.82 ms
--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 8ms
rtt min/avg/max/mdev = 7.794/8.422/10.792/0.699 ms
```

The statistics at the end show an overview of how many connections went through before we called `^C` and how long it took.

git

`Git` is a version control system that is commonly used in the industry and in open source projects.
See git tutorial

Environment Variables

Environment variables are named variables that contain values used by one or more applications.
The `PATH` variable contains a list of directories where systems look for executable files.
The `HOME` variable contains the path to the home directory of the current user.
The `PS1` variable is the default prompt to control appearances of the command prompt.

Connectors

Connectors allow you to run multiple commands at once.

Connector | Description --- | --- | --- `&&` | first command always executes and the next command will only execute if the one before it succeeds `||` | first command always executes and the next command will only execute if the one before it fails `;` | first command and the following commands always execute

```
$ true && echo Hello
Hello
$ false || echo Hello
Hello
$ echo Hello ; ls
Hello
test.txt file1.txt file2.txt
```

Piping

Pipes connect multiple commands together by sending the stdout of the first command to the stdin of the next command. For example, the following sends the `ls` output to `head` so that only the top 10 items get printed.

```
$ ls -l | head
```

Input/Output Redirection

Output Redirection

Standard output redirection uses the symbols `>` and `>>`.
For example, the following sends the output of `ls` into the file instead of printing to the screen.

```
ls > files.txt
$ cat files.txt
file1.cpp sample.txt
```

If the file isn't already in your working directory, the file gets created. If the file already exists, then the contents of the command overwrites what is already in the file.
To avoid overwriting a file, the `>>` command appends to the end of the file instead.

Input Redirection

Standard input redirection uses the symbol `<`.
For example, the following causes `sort` to access its input from the file instead of the keyboard.

```
$ cat files.txt
c
b
$ sort < files.txt
b
c
```

The `sort` command prints the contents of the file and prints to the screen because we haven't redirected its output. But we can combine I/O redirection into one command line, such as:

```
$ sort < files.txt > files_sorted.txt
```

Advanced Redirection

Adding a `&` with the `>` symbol results in redirecting both standard out and standard error. For example, the `test.cpp` file prints the string "stdout" with `cout` and the string "stderr" with `cerr`.

```
$ g++ test.cpp
$ ./a.out >& test.txt
$ cat test.txt
stdout
stderr
```

The `>` symbol alone only redirects standard output.
If you only want to redirect a specific file descriptor you can attach the file descriptor number to `>`.

Name	File Descriptor	Description
stdin	0	standard input stream
stdout	1	standard output stream
stderr	2	standard error output stream
For example, if I only wanted to redirect "stderr" to the file `test.txt` from the above example, I would do the following:		

```
$ g++ test.cpp
$ ./a.out 2> test.txt
stdout
$ cat test.txt
stderr
```

Permissions

The command `ls -l` prints out a lot of information about each file that is informative about the permissions.

```
$ ls -l test.txt
-rw-rw-r--  1  user  group  1097374 January 26 2:48 test.txt
```

Output from example above	Description/Possible Outputs

- | File type:
  `-` = regular file
  `d` = directory rw- | Permissions for owner of file rw- | Permissions for members of the group owning the file r-- | Permissions for all other users user | name of user owning the file group | name of group owning the file

chmod

The `chmod` command, short for change mode, changes the permissions of a file.
There is a combination of letters that need to be known in order to change specific users' permission.

Letter	User
u	User who owns it
g	Users in the group
o	Other users not in the group
a	All users
You call `chmod` by describing which actions you want to perform and to which file.	
The `-` symbol represents taking away permissions while the `+` symbol represents adding permissions.	
The following example makes the file readable and writable to the user who owns it and the group.	

```
$ chmod ug+rw test.txt
$ ls -l test.txt
-rw-rw----  1  user  group  1097374 January 26 2:48 test.txt
```

Alternatively, we can use `chmod` with hex numbers. You can think of each permission setting as a bit where it is a `1` if there is permission for the file and `0` otherwise.

```
rwx = 111 = 7
rw- = 110 = 6
r-x = 101 = 5
r-- = 100 = 4
```

Each set of permissions represents a single digit so the following commands have the same outcome as above.

```
$ chmod 660 test.txt
```

See permissions tutorial

Keyboard Shortcuts

Shortcut	Description
CTRL-A	Move cursor to beginning of line
CTRL-E	Move cursor to end of line

CTRL-R	Search bash history
CTRL-W	Cut the last word
CTRL-U	Cut everything before the cursor
CTRL-K	Cut everything after the cursor
CTRL-Y	Paste the last thing to be cut
CTRL-_	Undo
CTRL-L	Clears terminal screen