#####################################################

# Name: Bash CheatSheet for Mac OSX

#

# A little overlook of the Bash basics

#

# Usage:

#

# Author: J. Le Coupanec

# Date: 2014/11/04

#####################################################

### # 1. Bash Basics.

echo $SHELL # displays the shell you're using

echo $BASH\_VERSION # displays bash version

bash # if you want to use bash (type exit to go back to your normal shell)

whereis bash # finds out where bash is on your system

clear # clears content on window (hide displayed lines)

!! # repeats the last command

exit # logs out of current session

#### # 1.1. File Commands.

ls # lists your files

ls -l # lists your files in 'long format', which

# contains the exact size of the file, who owns

# the file and who has the right to look at it,

# and when it was last modified

ls -a # lists all files, including hidden files

ln -s <filename> <link> # creates symbolic link to file

touch <filename> # creates or updates your file

cat > <filename> # places standard input into file

more <filename> # shows the first part of a file (move with

# space and type q to quit)

less <filename> # more robust file tabber than more

head <filename> # outputs the first 10 lines of file

tail <filename> # outputs the last 10 lines of file (useful with

# -f option)

mv <filename1> <filename2> # moves a file

cp <filename1> <filename2> # copies a file

rm <filename> # removes a file

diff <filename1> <filename2> # compares files, and shows where they differ

wc <filename> # tells you how many lines, words and characters

# there are in a file

chmod -options <filename> # lets you change the read, write, and execute

# permissions on your files

gzip <filename> # compresses files

gunzip <filename> # uncompresses files compressed by gzip

grep <pattern> <filenames> # looks for the string in the files

grep -r <pattern> <dir> # search recursively for pattern in directory

#### # 1.2. Directory Commands.

mkdir <dirname> # makes a new directory

cd # changes to home

cd <dirname> # changes directory

pwd # tells you where you currently are

#### # 1.3. SSH, System Info & Network Commands.

ssh user@host # connects to host as user

ssh -p <port> user@host # connects to host on specified port as user

whoami # returns your username

passwd # lets you change your password

date # shows the current date and time

cal # shows the month's calendar

uptime # shows current uptime

w # displays whois online

uname -a # shows kernel information

man <command> # shows the manual for specified command

df # shows disk usage

du <filename> # shows the disk usage of the files and directories

# in filename (du -s give only a total)

last <yourUsername> # lists your last logins

ps -u yourusername # lists your processes

kill <PID> # kills (ends) the processes with the ID you gave

killall <processname> # kill all processes with the name

top # displays your currently active processes

bg # lists stopped or background jobs ; resume a stopped

# job in the background

fg # brings the most recent job in the foreground

fg <job> # brings job to the foreground

ping <host> # pings host and outputs results

whois <domain> # gets whois information for domain

dig <domain> # gets DNS information for domain

dig -x <host> # reverses lookup host

wget <file> # downloads file

### # 2. Basic Shell Programming.

#### # 2.1. Variables.

varname=value # defines a variable

varname=value command # defines a variable to be in the environment of

# a particular subprocess

echo $varname # checks a variable's value

echo $$ # prints process ID of the current shell

echo $! # prints process ID of the most recently invoked

# background job

echo $? # displays the exit status of the last command

export VARNAME=value # defines an environment variable (will be

# available in subprocesses)

array[0] = val # several ways to define an array

array[1] = val

array[2] = val

array=([2]=val [0]=val [1]=val)

array(val val val)

$(UNIX command) # command substitution: runs the command and

# returns standard output

#### # 2.2. Functions.

# The function refers to passed arguments by position (as if they were

# positional parameters), that is, $1, $2, and so forth.

# $@ is equal to "$1" "$2"... "$N", where N is the number of positional

# parameters. $# holds the number of positional parameters.

functname() {

shell commands

}

unset -f functname # deletes a function definition

declare -f # displays all defined functions in your login session

#### 

#### 

#### # 2.3. Flow Control.

statement1 && statement2 # and operator

statement1 || statement2 # or operator

-a # and operator inside a test conditional expression

-o # or operator inside a test conditional expression

str1=str2 # str1 matches str2

str1!=str2 # str1 does not match str2

str1<str2 # str1 is less than str2

str1>str2 # str1 is greater than str2

-n str1 # str1 is not null (has length greater than 0)

-z str1 # str1 is null (has length 0)

-a file # file exists

-d file # file exists and is a directory

-e file # file exists; same -a

-f file # file exists and is a regular file (i.e., not a

# directory or other special type of file)

-r file # you have read permission

-r file # file exists and is not empty

-w file # your have write permission

-x file # you have execute permission on file, or directory

# search permission if it is a directory

-N file # file was modified since it was last read

-O file # you own file

-G file # file's group ID matches yours (or one of yours, if

# you are in multiple groups)

file1 -nt file2 # file1 is newer than file2

file1 -ot file2 # file1 is older than file2

-lt # less than

-le # less than or equal

-eq # equal

-ge # greater than or equal

-gt # greater than

-ne # not equal

if condition

then

statements

[elif condition

then statements...]

[else

statements]

fi

for x := 1 to 10 do

begin

statements

end

for name [in list]

do

statements that can use $name

done

for (( initialisation ; ending condition ; update ))

do

statements...

done

case expression in

pattern1 )

statements ;;

pattern2 )

statements ;;

...

esac

select name [in list]

do

statements that can use $name

done

while condition; do

statements

done

until condition; do

statements

done

### 

### 

### # 3. Command-Line Processing Cycle.

# The default order for command lookup is functions, followed by built-ins,

# with scripts and executables last.

# There are three built-ins that you can use to override this order:

# `command`, `builtin` and `enable`.

command # removes alias and function lookup. Only built-ins and commands

# found in the search path are executed

builtin # looks up only built-in commands, ignoring functions and commands

# found in PATH

enable # enables and disables shell built-ins

eval # takes arguments and run them through the command-line processing

# steps all over again

### # 4. Input/Output Redirectors.

cmd1|cmd2 # pipe; takes standard output of cmd1 as standard input to cmd2

> file # directs standard output to file

< file # takes standard input from file

>> file # directs standard output to file; append to file if it already

# exists

>|file # forces standard output to file even if noclobber is set

n>|file # forces output to file from file descriptor n even if noclobber is

# set

<> file # uses file as both standard input and standard output

n<>file # uses file as both input and output for file descriptor n

<<label # here-document

n>file # directs file descriptor n to file

n<file # takes file descriptor n from file

n>>file # directs file description n to file; append to file if it already

# exists

n>& # duplicates standard output to file descriptor n

n<& # duplicates standard input from file descriptor n

n>&m # file descriptor n is made to be a copy of the output file

# descriptor

n<&m # file descriptor n is made to be a copy of the input file

# descriptor

&>file # directs standard output and standard error to file

<&- # closes the standard input

>&- # closes the standard output

n>&- # closes the ouput from file descriptor n

n<&- # closes the input from file descripor n

### # 5. Process Handling.

# To suspend a job, type CTRL+Z while it is running. You can also suspend a

# job with CTRL+Y.

# This is slightly different from CTRL+Z in that the process is only stopped

# when it attempts to read input from terminal.

# Of course, to interupt a job, type CTRL+C.

myCommand & # runs job in the background and prompts back the shell

jobs # lists all jobs (use with -l to see associated PID)

fg # brings a background job into the foreground

kill -l # returns a list of all signals on the system, by name and number

kill PID # terminates process with specified PID

ps # prints a line of information about the current running login

# shell and any processes running under it

ps -a # selects all processes with a tty except session leaders

### # 6. Debugging Shell Programs.

bash -n scriptname # don't run commands; check for syntax errors only

set -o noexec # alternative (set option in script)

bash -v scriptname # echo commands before running them

set -o verbose # alternative (set option in script)

bash -x scriptname # echo commands after command-line processing

set -o xtrace # alternative (set option in script)

trap 'echo $varname' EXIT # useful when you want to print out the values of

# variables at the point that your script exits