ESSENTIALS

Ctrl + plus key or minus = increese or decrease font size in terminal window
echo “Hello World” = types Hello World
Up and down arrows = move to previous next commands (command history)
exit = closes the window
Ctrl + a = move the cursor to the beginning
Ctrl + e = move the cursor to the end

Tab = Will try to guess what you want to type
Tab + Tab = Will show you a list of all possible matches for autocomplete
Ctrl + l = clear the screen

COMMAND STRUCTURE

command options (arguments of options) arguments

echo $SHELL = what shell will you be launched as (login shell)
echo $0 = what is the current shell in use
sh = move to the sh shell

HELP

man echo = shows you the manual page for the echo commands
man man = shows you the manual page for the man commands
apropos ban = searches for possible matches for commands containing ban
whatis banner

MOVING AROUND DIRECTORIES

pwd = present working directory, where i am located right now

ls = list the contents of pwd
ls -l = list vertical format
ls -a = list dot files which are invisible config files
ls -h = returns the size of the files listed

cd nameofdirectory = changes pwd to nameofdirectory directory
cd .. = changes directory to parent directory
cd Library/Preferences = moves two directories at once
cd ../.. = go to the parent of the parent of this directory
cd / = go to root of our hard drive
cd /Library/Preferences = move to absolute path
cd ~ = moves you to the user directory
cd - = toggles you between current directory and last dir you were in

cd Application\ Support or cd “Application Support” will move into the Application Support directory. As a general rule, try not to use spaces or symbols except the _ on file names. Use numbers and letters. Most unix systems are case sensitive so MyFile and myfile are two different files.

CREATE FILES

touch newfile.txt = create a file
touch existingfile.txt = update the file's access time

nano = enter the nano editor

nano newfile.txt = it will enter the editor and will have that name waiting for you.

Within nano you can move forwards and backwards, cut and uncut (paste) text, help menu, search for, etc..
at www.nano-editor.org is a resource to learn more
READ FILES

you can read files using
nano filetoread.txt
to read the file

1. cat = reads and output the first file and read and concatenate second file
if you pass only one file, you just use it to read that filetoread.txt

example cat short_file.txt newfile.txt
outputs the two files concatenated

2. more = output one page of text and you can move forward pressing spacebar

3. less = same as more but loads one page into memory and you can scroll backwards. Even if you type more, less will execute, because less is an improvement on more.
    f = go forwards
    b = go backwards
    q = quit
    shift+g = go to the end of the doc
    g = takes you to the top of the doc

less -M = get info on scrolling
less -N = get line numbers

head somefile.txt = shows you the first lines of a file
tail somefile.txt = shows you the last lines of a file
tail -f = it will watch for changes in the file.txt on a different terminal window.
    ctrl+c = exit

DIRECTORIES

mkdir directoryname = creates directory
mkdir directoryname/test1 = creates directory inside another
to create all parent directories at once mkdir -p directory1/test1/test2/test3

ls -la testdir/
shows you what is in there

MOVING AND RENAMING FILES AND DIRECTORIES

mv newfile.txt testdir/newfile.txt
or
mv newfile.txt testdir
moves the file into the testdir directory

mv newfile.txt ../newfile.txt
or
mv newfile.txt ..
moves the newfile.txt file into the parent folder

mv test1/test3 test1/test2
moves test3 directory inside the test1 directory into the test1/test2 directory1

TO RENAME A FILE

mv newfile.txt new_file.txt
renames newfile as new_file

mv newfile.txt testdir/new_file.txt
renames new file and moves it inside testdir
mv -n = no overwriting – dont accidentally overwrite a file.txt
mv -f = force overwriting
mv -i = ask me if i want to overwrite
mv -v = gives you reporting information

by default mv is mv -f

COPY FILES

cp filetobecopied.txt destinationofcopy.txt

cp -n = no overwriting
cp -f = force overwriting
cp -i = ask me if i want to overwrite
cp -v = get information

cp -R = copy a directory and all of its contents
cp -R directorytocopy copyofdirectory

REMOVING FILES AND DIRECTORIES

rm filetoremove.txt

completely deletes a file

rmdir directorytodelete

completely deletes an empty directory1

rm -R directorytodelete

completely deletes a directory and all its contents

LINKS IN UNIX

ln filetolink hardlink

creates a link(shortcut) to a file.txt like in windows
except hardlinks don't break if original file is deleted

ln -s filetolink symlink
creates a symbolic link

Symbolic links reference the path to the file, not the file itself. Hardlinks keep track of the file, symbolic links just of the path. They break if the file is moved or deleted.

SEARCHING FILES AND DIRECTORIES

find path expression

find ~/Documents -name “someimage.jpg”
where -name and “someimage.jpg” are arguments not options.
That will return searching in the User/Documents folder anything with the name someimage.jpg

Wildcards:
* = 0 or more characters can be represented
? = any one characters
[] = any of the characters within the brackets

you can do more complex searches for size, read the man pages.
find ~ -name *.txt -and -not -path *Quicktime*

Searches in the user folder anything with .txt in their name and not within a path that contains Quicktime in the route.

FILE OWNERSHIP AND PRIVILEGES

whoami = tells you which user you are

cd ~ = will send you to the folder of whoever you are logged in as

echo $HOME = that's where your home directory is (for your user)

GROUPS

Each user can belong to a group. File permissions can be assigned to users or groups of users. By adding a user to a group they can gain access to files.

Groups = shows you the groups you belong to

FILE AND DIRECTORY OWNERSHIP

The owner is the third column, the group is the fourth column.
We can set permissions based on the owner or the group.
Any file you create is yours to own

chown kevin:staff ownership.txt
chown kevin ownsership.txt
chown :staff ownership.txt
chown kevin:staff directory1
changes ownership of ownership.txt to kevin owner and staff group

chown –R kevin:staff testdirectory
changes ownership of testdirectory and all its contents to kevin and the group staff

sudo chown lynda:staff ownership.txt
changes ownership as superuser

FILE AND DIRECTORY PERMISSIONS

in ls -la first colum:
first character: d directory, - file, l for symbolic link
following characters: user, group, other
r read the file, w write the file, x execute the file or directory

example:
drwxrw-r--
a directory where the user can read, write, and execute, a group that can read and write but not execute, and others who can only read the file

chmod
change mode or change permissions

structure:
chmod mode filename

user: u, group: g, other: o

chmod ugo=rwx filename
give read write and execute permissions to user group and other

chmod u=rwx, g=rw, o=r filename
chmod ug+w filename

for user and group add the write permissions

chmod o-w filename

for other remove write permissions

chmod a+rw

add read write permission to all (a) (user, group, other)

chmod -R g+w test1

will changes write permissions for groups on the test1 directory and everything inside of it

OCTAL NOTATION TO CHANGE FILE PERMISSIONS

r = 4, w =2, x= 1

rwx = 7, rw=6, rx=5, w=2 and so on

chmod 777 filename

give all the permissions to everyone

chmod 764 filename

user has all privileges, group read and write, and others only read

chmod 000 filename

take all privileges from everyone

SWITCH TO A DIFFERENT USER IDENTITY

The root user can do absolutely anything on the unix system. Not bound by normal user permissions

sudo = substitute the user and do

sudo ls -la
won't require a password for five minutes after entering the password the first time.

sudo -k
to ask it to ask you a password

sudo whoami
(it will show current user as sudo)
sudo -u lynda whoami
(it will say lynda)

sudo without any options will make you into the root user.

Only admins can do sudo.

UNIX COMMANDS AND PROGRAMS

commands are executable files, they are shorthands of files and you pass arguments the file executes.

Whereis echo
which echo
whatis banner

get information about commands

it's common you can pass -v, –version, or –help
after the command to learn more about commands

To exit programs, most common:
q, x, ctrl + x, ctrl + q, or ESC

you can force quite Ctrl + c.

you can put semicolons between commands to execute two commands in sequence

MANAGE COMMANDS

echo $PATH
outputs a colon separated list of directories where to look for the command echo
it will look from left to write

PATH=directory/subfolder:directory2/subfolder
to set the path, it only lasts for the current session

which ruby

it will tell us the path we'll get if we type the command ruby

USEFUL COMMANDS

date
tells you the current date the computer has been set to

uptime
tells us the time the computer has been turned on

who
shows us a list of all users and what they are doing
(each new terminal open adds an instance of the same user)

uname
returns the operating system name

uname -mnrsvp
uname -ap
either tells you details about the operating system, release, hardware

hostname
the host you are on

domainname
the domain you are on in a shared server

MORE COMMANDS

df
displays disk freespace

df -h
humanize the data about diskfreespace

df -H

humanize the data about diskfreespace with alternative calculation basis for gigabites... df -h is the one we normally use.

du ~/unix_files
disk usage for user/unix_files directory1

if you accidentally just type du, it will show disk usage for every directory in the computer, hit ctrl + c to exit

du -ah ~/unix_files
show also files not only directories and their sizes

du -hd 1 ~/unix_files

shows disk usage for current directories plus 1 in humanized form

disk usage shows the size that has been set aside for the file or directory, not the actual size of the file.
ls reports the actual size of the files.

MANAGING PROCESSES

ps
gives a snapshot of processes running by your user

ps -a
gives a snapshot of proceses running by all users, not you

ps aux (no hyphen)
the more commmon implementation of ps -a
a: show me processes for all users
u: show me processes include a column showing me the user who owns the process
x: show me processes runnning in the background

top
shows top processes live
q will exit out

q -n 10
show me top 10 processes

1 -n 10 -o cpu -s 3 -U kevin
top ten processes by kevin sorted by cpu with 3 second refresh

if you open firefox, firefox will appear dynamically on the list

ctrl + c
tells unix stop that process, let's go back to the command line

each process has a unique id
kill 1837
kills process 1837, to know the id of the process, runnning
ps aux

kill -9 1837
kill it forcibly (don't let the computer decide whether to kill it or not)

you can kill a process you don't have control over, but be careful with usage

TEXT FILE HELPERS

wc file.txt
outputs 3 numbers, number of lines in the file (normally paragraphs, everythin until a line return in a file), number of words in the file, and the characters in the file.txt

sort file.txt
sorts lines in a file alphabetically. Capital letters and lowercase first letters are treated different.

Sort -f file.txt
sorts treating capitalized letters and lower case first letters as the same

sort -r file.txt
reverse sort

sort -u file.txt
sorted and unique – sorts and gets rid of duplicates.

uniq file.txt
gets rid of duplicates (de-duplicates)

uniq -d file.txt
returns lines that are repeated

uniq –u file.txt
shows unduplicated lines

UTILITY PROGRAMS

cal
you get the current month's calendar

cal 12 2020
see december 2020

cal -y 2000
show the calendar for all year 2000

ncal
rotates the calendar

bc
runs a calculator
    to get out type quit

ex:
(3+4) * 210.3
output: 1472.1

```
scale=10
1000/9
output: 111.111111111
```

```
expr 1 + 1 (you need to add spaces between characters)
```

```
expr 1122 \* 3344
```
you need to escape special characters
```
output: 3751968
```

```
units
You have: 1 foot
You want: meters
outputs: 0.3048 meters
                    /3.2808399 reverse calculation
```

```
You have: 72 degF
You want: degC
```

to exit = Ctrl+c

also:

```
units '2 liters' 'quarts'
```

COMMAND HISTORY

In User/.bash_history you can review the history of commands

```
history
```
shows all recent commands, you can use the reference number of the commands to execute a command. Asterisk shows edited commands.

```
!1
```
runs the first command

```
!-2
```
runs a command two commands back, relative to where you are now

```
!expr
```
executes the most recent command you did that started with expr

```
!!
```
is the same as !-1

```
sudo !!
```
is the same as the last command prefixed with sudo

!$
references the arguments from previous commands

cat !$
references directories and files of previous commands.

For example if you wrote nano file.txt
you then do cat !$ it will do
cat file.txt

history -d 27
deletes line 27 of history of commands

history -c
deletes all of history

STANDARD INPUT AND OUTPUT

sort file.txt > outputfile.txt
instead of sending this info to the screen, send it to the outputfile.txt

any command that we output to the screen we can output to a file.
ls -lah > content.txt
history > history.txt

this command overwrites a file if already existing.

cat new_file.txt
cat newer_file.txt
cat new_file.txt newer_file.txt > joined.txt

echo “Claire” >> people.txt
it appends “Claire” to the end of people.txt, it doesn't destroy it..

sort < file.txt
take this file.txt and use it to sort them
using the contents and pass it to sort of whatever function

echo “(3*4)+(11*37)” > calculation.txt
bc < calculation.txt

take the results of calculation and use it to execute bc commands

remember: arguments to > and < need to be file or directory

uniq < sorted_fruit.txt > unique_sorted_fruit.txt
save into unique_sorted_fruit.txt the result of uniq procesing sorted_fruit.txt

echo “Hello World” | wc
took the output from one command piped it into another (wc)

with the pipe (|) we work with commands, not files like with < >
the output should be output from a command, not a file

cat fruit.txt | sort
cat fruit.txt | sort | uniq

piping from one command to another, catting, sorting and unique showing the contents of fruit.txt

ps aux | less
we get pagination of ps aux

SUPPRESSING OUTPUT

ls -la > /dev/null
cat lorem_ipsum.txt > /dev/null

it's a file that always stays empty, there is no output. For more advanced use.

CONFIGURE WORKING ENVIRONMENT – CUSTOMIZING BASH

when you login a new shell:
the computer reads one and only one of these files in this order of priority
~/.bash_profile, ~/.bash_login, ~/.profile, ~/.login
when you open an additional terminal window
~/.bashrc
when you logout, the computer runs
~/.bash.logout

Add to ~/.bash_profile:

```
if [ -f ~/.bashrc ]; then
	source ~/.bashrc
fi
```

this allows us to put all of our configurations in one file and have them execute in both the first time we open a terminal window, and anytime we open an additional terminal window without having to modify two different configuration files.

COMMAND ALIASES

alias
will return a list of all currently defined alias

alias nameofalias= 'command and options'

example
```
alias ll='ls -la'
alias hello='echo “Hello World”'
```

aliases only last for the current login
you can save your aliases into the .bashrc file so that it is saved every time you open a terminal.

Might be a good idea
```
Alias mv='mv -i'
Alias cp='cp -i'
Alias rm='rm -i'
Alias df='df -h'
Alias du='du -h'
```

ENVIRONMENT VARIABLES

```
echo $SHELL
```

```
MYNAME='Jose Iriarte'
echo $MYNAME
```

variables also die in the current terminal session
you can save them to .bashrc for them to be available the next time.

To export variables we need to use the export command:
```
export MYNAME
```
everytime it launches a program it will make it available to all programs, for example LESS – which is a program

```
source .bashrc
```
to execute the .bashrc programs

PATH – MODIFY IT

```
echo $PATH
```
unix will look those directories in that order

```
PATH=''
```
unsets the path for the current terminal session

```
PATH=”/usr/local/bin:$PATH”
```
you might want to do this ammendment in the .bashrc file to modify what we get at the begining of the session.

Path is the most important environment variable you can set

```
HISTSIZE=10000
```
will remember last 10000 commands

HISTFILESIZE=1000000
will make the history file size 1million kb

HISTTIMEFORMAT= %b %d %I: %M %p
formats each history file

HISTCONTROL=ignoreboth
you can also set it to ignoredups, we dont want history to record the same line multiple times,
ignorespace, we dont want history to remember commands that have a space in front of it

HISTIGNORE=”history:pwd:exit”
the commands history will not save in history.txt

history | tail -8
will show the last 8 commands you did

CUSTOMIZE THE UNIX COMMAND PROMPT

PS1=”--> “

for our prompt to always work like this, you need to include it into .bashrc

PS1=”Kevin “
PS1=”\u > “ (username)
\s (current shell)
\w (current working directory)
\W (basename of current working directory)
\d (date in “weekday month date” format
\D(format) (date in strftime format)
\A time in 24 hr HH:MM format
\t time in 24 hr HH:MM:SS format
\@ time in 12 hr HH:MM am/pm format
\T time in 12 hr HH:MM:SS format
\H hostname
\h hostname up to first “.”
\! history number of this command
\$ when UID is 0 (root), a #, otherwise a “$”
\\ a literal backslash

you can combine these

nano .bash_logout
choose what to do when you logout, apart or instead of closing the window
for example clearning temporary files

POWERFUL UNIX COMMANDS – SERIOUS WORK

grep: search for text using regexs

(global regular expression print)

grep apple fruit.txt
searches for the string “apple” in fruit.txt file
returns the entire line where that word appears
it is case sensitive

grep -i Apple fruit.txt
case insensitive searches

grep -w apple fruit.txt
it only finds whole word matches

grep -v apple fruit.txt
we get the lines that don't match the string “apple”

grep -n apple fruit.txt
shows line numbers and results
grep -c apple fruit.txt
count the number of times it finds it

grep -R apple /Users/kevin/unix_files
searches all the files in that directory recursively (inside all the dirs inside the specified directory)

grep -Rh apple .
Suppres the file name and search recursively in . Which is the current directory1

grep -L apple .
Shows all files that don't match

ps aux | grep Terminal
shows you only the Terminal named running processes

history | grep unix_files

grep –-color lorem lorem_ipsum.txt
colors the word lorem every time it shows a line on the lorem_ipsum.txt file

REGULAR EXPRESSIONS FOR GREP

Its a good idea to put quotes around regular expressions

. a wildcard that replaces any characters

ea[cp]
third character must be c or p

(print regex document from programming course) what to watch out for using regex with grep:

1.
```
grep 'ap*le' *fruit.txt
```
the first asterisk means the p is repeated 0 or more times,
the second asterisk is a wildcard for the file system.

2. the plus sign is an extended regular expression, for it to work we need to use

```
grep -E 'ap+le' fruit.txt
```

```
grep 'apple|pear' fruit.txt
```
also needs the -E
```
grep -E 'apple|pear' fruit.txt
```
to find apple OR pear, and not the literal string “apple|pear”

TRANSLATE

```
echo 'a,b,c' | tr ',' '-'
a-b-c
```

```
echo '14252436524' | tr '123456' 'EBGDAE'
EDBABDGEABD
```

```
echo 'This is ROT-13 encryp'ted.' | tr 'A-Za-z' 'N-ZA-Mn-za-m'
Guvf vf EBG-14 rapelcgrq.
echo 'Guvf vf EBG-14 rapelcgrq.' | tr 'A-Za-z' 'N-ZA-Mn-za-m'
This is ROT-13 encrypted
```

```
echo “abc1233dreeee567f' | tr 'bedf5-9' 'x'
axc1233x4xxxxxxxx
```

```
tr 'A-Z' 'a-z' < people.txt
```
(replaces all upper case to lower case)

```
tr '[:upper:]' '[:lower:]' < people.txt
```

```
echo “trés animée” | tr 'é' 'e'
```

for csv files

```
tr ',' '\t' < us_presidents.csv > us_presidents.tsv
```
replaces commas with tabs and outputs to a different tsv file

TR OPTIONS

-d (delete characters in a listed set)
-s (squeeze repeats in listed set)

-c (use complementary set)

```
echo “abc1233deee567f” | tr -d [:digit:]
abcdeeef
```

```
echo “abc1233deee567f” | tr -dc [:digit:]
1233567
```

```
echo “abc1233deee567f” | tr -s [:digit:]
123deee567f
```

common uses:
remove all non printable characters off a file

```
tr -dc [:print:] < file1 > file2
```

remove all double spaces from a file

```
tr -s '  ' < file1 > file2
```

STREAM EDITOR SED

```
sed 's/a/b/'
```

s: substitution
a: search string
b: replacement string

```
echo 'upstream' | sed 's/up/down/'
downstream
```

```
echo 'upstream and upward' | sed 's/up/down/g'
```

(g: global)

```
downstream and downward
```

you can replace the / for : or | in cases where / and or : are already in use in the text

SED will take a second argument which is the file you want to username

```
sed 's/pear/mango' fruit.txt
```

```
sed 's|colour|color|g' fruit.txt
```

```
echo 'During daytime we have sunlight' | sed -e 's/day/night/' -e 's/sun/moon/'
During nighttime we have moonlight
```

SED AND REGEX

```
echo “Who needs vowels?” | sed 's/[aeiou]/_/g'
We_ n__ds v_w_ls?
```

Sed -E 's/<>//g' homepage.html
-E (uses extended set of regexs)
strips out html tags

echo 'daytime' | sed -E 's/(...)time/\1light/'
where \1 is the first backreference
\2 the second backreference

echo “Dan Stevens” | sed -E 's/([A-Za-z]+) ([A-Za-z]+)/\2, \1/'
Stevens, Dan

sed -E 's/(apple|pear|plum|peach)/\1 tree/ fruit.txt
(replaces peach for peach tree, pear for pear tree and so forth)

CUT

cut can cut -f bytes, -c characters and -f fields

cut -c 2-10 dir_content.txt
cuts and shows the second through 10 characters in each line

cut -c 2-10,30-35,49- dir_content.txt
cuts and shows this set of characters 49- cuts from 49th character through the end

ps aux | cut -c 11-15, 72-
(you get the process id and what is running)

cut -f 2,6 us_presidents.tsv
takes tabs as delimiters from files, 2 and 6 are the columns

for csv files we have to use an option
cut -f 2,6 -d “,” us_presidents.csv
takes commas as delimiters from files, 2 and 6 are the columns

DIFF – COMPARE TWO FILES

diff original_file.txt revised_file.txt
(diff reports the changes between files)
2d1
a deletion on the second line of the first file, and the first line on the second file.
6c5
11a11

c is for change
a is for append

diff -i (case insensitive comparison)

diff -b (ignore changes to blank characters)
diff -w (ignore all whitespace)
diff -B (ignore blank lines)
diff -r (recursively compare directories)
diff -s (show identical files)

DIFF OPTIONS

these options vary the output of the diff command

diff -c original_file.txt revised_file.txt
(shows both files one on top of the other, -deleted, !changed, +added)

diff -y original_file.txt revised_file.txt
(one file on the left one file on the right)

diff -u original_file.txt revised_file.txt
(merged files)

diff -q original_file.txt revised_file.txt
(tells you if the files differ)

diff -u original_file.txt revised_file.txt > original_revised.diff
(outputting diff into .diff file is standard, and certain text editors have color coding for .diff files)

diff -u original_file.txt revised_file.txt | diffstat
(shows statistics for changes)

XARGS – EXECUTE AS ARGUMENTS

echo 'lorem_ipsum.txt' | xargs wc
counts the words of lorem_ipsum.txt as a file and not as a string

echo 'lorem_ipsum.txt' | xargs -t wc
does the action and shows you what its doing first

echo 'lorem_ipsum.txt us_presidents.csv' | xargs -t -n1 wc
loops through and executes xargs for the first file and then for the second file

ls | xargs -n3 echo
shows lines of filenames three at a time

cat fruit.txt | xargs -I {} echo “buy more: {}”

buy more: (each line of fruit.txt)
i.e.
buy more: pears
buy more: apples

etc...

XARGS EXAMPLES

```
cat file_manifest.txt | xargs cat | less
```

concatenates files

```
cat fruit.txt | sort | uniq | xargs -I {} mkdir -p ~/Desktop/fruits/{}
```

(we get a folder for each of the fruits listed in fruit.txt)

```
ps aux | grep 'badprocesses' | cut -c 11-15 | xargs kill -9
```
(we just killed all of our bad processes)

```
grep -l 'apple' *fruit.txt | xargs wc
```
we can find the wc of each file containing fruit.txt in the file name

```
find . -name “*fruit.txt” -print0 | xargs -0 -I {} cp {} ~/Desktop/{}.backup
```
-print0 (make sure the null character is used to separate them)
-0 (makes sure xargs uses that null character)

look for everything in the current directory whose name matches fruit, and copy each one of those arguments and copied it as a backup file on my Desktop

```
find ~/Desktop/ -name “*.backup” -print' | xargs -p -0 -n1 rm
```
remove those files one by one