**Week 1 - Vagrant**

All commands require a Vagrantfile in the current directory.

* vagrant up
  + (start machine)
* vagrant ssh
  + (log in)
* vagrant halt
  + (stop machine)
* vagrant reload
  + (stop+start machine (for config update))
* vagrant destroy
  + (delete machine)
* exit
  + logout from vagrant

**Debian Apt**

* sudo apt install <package name>
* sudo apt remove <package name>
* sudo apt update
  + -download new list of packages, but don’t install yet
* sudo apt upgrade
  + upgrade

**Shell**

* whoami
  + print username
* ls
  + list files
* ls -l
  + list files with permissions
* ls -a
  + list all files including hidden ones
* man [section] COMMAND
  + e.g. man 1 printf vs man 3 printf

Shell expansion

* cp [-rfi] SRC..DEST
  + -r recursive
  + -f overwrite readonly
  + -i ask before overwrite
* mv [-nf] SRC..DEST
  + n – no overwrite
  + f -force overwrite
* find DIR [EXPRESSION]
  + e.g. find /path/to/search -name filename.txt
    - search for files named "filename.txt" within the specified directory and its subdirectories.
  + find all files in directory recursively that match the expression
* Shell takes names between spaces as separate arguments unless with quotes “ “

**Pipe/ Regular Expression**

The vertical bar or "pipe" symbol (|) is used to combine the output of one command with the input of another.

* ls -l | head
  + display the first 10 lines of ls -l to standard output
    - ls -l | head -n 5
      * only the first 5
    - ls -l | head -n -1
      * drop the last from first 10
* ls -l | tail
  + display the last 10 lines of ls -l to standard output
* ls -l | grep software | sort -r
  + sort: read all lines into buffer and sort output
* ls -l | sort | uniq
  + uniq: remove duplicates following sort
* grep [-iv] FILENAME
  + -i: case insensitive
  + -v: print non-matching
* ls -l | grep -v -i software | wc -l
  + wc [-l]
  + word count if no -l
  + count for lines if wc -l
    - report number of lines of printed files that do not match “software” from ls -l
* cat [filename [filename] ..]
  + writes the contents of one or more files to standard output. This is a good way of starting a pipe.
* [Command] > [file]
  + >: redirect output from command to file (overwrite file)
* [Command] >> [file]
  + >>: append the output from command to file
* COMMAND > FILE 2> FILE2
  + Redirect stdout to FILE and stderr to FILE2
* COMMAND > /dev/null
  + Redirecting output to /dev/null essentially discards the output
* Program that uses standard stream can be told to use file instead
  + cat < input.txt
    - cat command reads the content of input.txt as its standard input instead of waiting for input from the keyboard.
  + cat input.txt > output.txt
    - The content of input.txt is read by cat, and the standard output is redirected to output.txt. If output.txt exists, its content will be overwritten.
  + program\_with\_error 2> error.log
    - The error message produced by program\_with\_error will be written to the error.log file. If error.log exists, its content will be overwritten.
  + program –
    - program that expects a filename can be told to understand stardard input/output instead
* ls -l | tee FILENAME
  + takes FILENAME as arugment and write copy of input into it, also to standard output
* ls -l |less
  + is a pager: it displays text on your screen, one page at a time.
* echo "Hello World" | sed -e 's/World/Universe/'
  + uses sed (stream editor) to replace the word "World" with "Universe" in the input string "Hello World." The expression 's/World/Universe/' is a substitution command that instructs sed to replace the first occurrence of "World" with "Universe."
* cat <(echo “hi”)
  + pipe (echo “hi”) in cat function
* cat words | head -n 6171 | tail -n 1
  + This command uses head to get the first 6171 lines and then uses tail to get the last line from that subset, effectively retrieving the 6171st word.
* cat words | grep ‘Q’
  + catch words containing letter Q
* cat words | grep -i ‘Q’
  + catch words containing letter Q (case-insensitive)
* cat words | grep '^Q'
  + catch words containing letter Q at the start of string
* cat words | grep 'Q$'
  + dollar sign must be quoted!
  + catch words ending with ‘Q’
* cat words | grep -E 'kp' | grep -Ev 'ckp'
  + All words containing the sequence "kp", but not "ckp"
* cat words | grep '^..$' | tail -n 15
  + last 15 words with exactly 2 letters
* cat words | grep '^...$' | tail -n 15
  + last 15 words with exactly 3 letters
* cat words | grep -i '^..e.ded$'

**Week 2 Git – all about tracking changes to source code**

Configuration (only need to do once when install git. not everytime when you creat a new repo)

* Git config --global user.name “YOUR NAME”
* Git config --global user.email “YOU EMAIL”
* Git config --list
  + To view git configurations

Starting a repository

* Git init
  + Initialise a hidden directory called “.git” in the project root directory.
* Git status

Staging files (for git to track)

* git add <YOUR FILE>
  + OR git add <FILE1> <FILE2> <FILE3>
  + OR git add . (all new changed files in the current folder)
  + OR git add --all ("Find all new and updated files everywhere throughout the project and add them to the staging area.") == git add -A
  + check with git status and can see tracked file(changes to be committed)
* git rm --cached <FILENAME>
  + to remove file from staging area
* git reset <FILENAME>\

Git ignore

* add .gitignore file and in each line add expression to say which file/folder to be ignored

Commit changes

* git commit -m “message”
  + if -m flag omitted, git will bring to system default editor to write one

Go back and look another commit/

* git log
  + take note at least the first 6 characters of the commit HASH of the commit
* git checkout HASH
* git checkout main
  + back to latest version of your file

Undo commit – two options

* git revert HASH
  + adds a new commit that returns the files to the state they were before the commit with this HASH
* git reset HASH
  + undoes commit by moving HEAD pointer back to the commit of given hash, but leaves the working copy alone (safe when have not pushed yet effect is as if commits have never happened)
* git reset <branch>~1
  + reset the <branch> to the commit before the last one
* git revert <branch>
  + reverse, undo all changes

Git forge

* Two ways to interact with remote repository
  1. Via HTTPS – ok if just cloning public repo
  2. Via SSH – using keys(public key, not private)
* git clone <git@github.com:USERNAME/REPONAME.git>
* git status
  1. Up to date
  2. Ahead of remote – made commits locally not yet pushed to remote
  3. Behind remote – someone else made commit to remote but I don’t have
  4. Diverged from remote – my computer and remote have had different commits since the last time I synchronised.

Git conflict (working with others on git)

* Git push work flow
  + git fetch
  + if no update – there were no changes on remote since the last fetch
  + if get output -> do git status
    - if BEHIND REMOTE -> git pull
    - if AHEAD REMOTE (I made changes) -> git push (send changes to the remote)
* Push workflow
  + Git fetch (any changes in remote?)-> git status (ahead of origin?) ->git push
* Two ways to solve conflict
  1. git rebase origin/main
     + if user 2 to rebase, it is pretending user 2 had already fetched user 1 before starting
  2. Make a merge
     + Will see diverged when git status.
     + if different users edited different files, merge should be successful
     + git merge <branch\_name2>
* Real conflict
  1. When running the git push workflow, will see CONFLICT message after git pull
  2. Solve the conflict and then:
     + git add <filename>
     + git commit (git will suggest Merge branch main..)
     + run another git push workflow (git fetch, git status), then should see 2 commits ahead. Finish with git push.

Branching

* git branch
  + list all branches
  + \* = the current branch I am in
* git branch <new\_branch name>
  + create new branch
* git checkout <branch\_name>
  + switch to <branch\_name>
* git checkout -b <new\_branch\_name >
  + creating git branch <branch> and git checkout that branch into one command
  + this branch only exists in local repository, will get error if git push a commit
  + need to git push --set-upstream origin <new\_branch-name>
* git merge <branch\_name>
  + Workflow: Need to check out the main branch first then merge the secondary branch into main. 1)git checkout <the main branch> then 2) git merge <secondary branch>.
  + git merge <branch\_name1> <branch\_name2><main branch>
    - merging changes in 1 and 2 into main
* git branch -d <branch\_name>
  + after merging the seconday into main, we can delete the secondary by adding “-d” like above. If there are unmerged changes, Git will prevent deletion.
* git remote show origin
  + display something like this

Remote branches:

develop tracked

main tracked

Local branches configured for 'git pull':

develop merges with remote develop

main merges with remote main

Local refs configured for 'git push':

develop pushes to develop (up to date)

main pushes to main (up to date)

* git branch -a
  + will show you all the branches, with names like remotes/origin/NAME for branches that so far only exist on the origin repository. You can check these out like any other branch to see their contents inyour working copy.
* git checkout <Abranch>; git rebase <Bbranch>
  + <Abranch> is now rebased to <Bbranch>

Choosing a commit from one branch and applying to another

* On the branch I want to apply the commit to
* git cherry-pick <commit-hash>
  + <commit-hash> = the commit I want to pick

**Week 3 – Admin, Shell Scripting& Build Tools**

* ls -loh
  + -o flag hide group column
  + -h flag make human readable
* sudo adduser NAME
  + create a new user
* sudo addgroup GROUPNAME
  + create a user group
* sudo adduser <USERNAME> <GROUP>
* tail-n Number FILE
  + display last 10 lines of a file by default
* su USERNAME
  + change to username
* chgrp -R <GROUPNAME> <DIRECTORY>
  + change group of the directory
* chmod g+r / g-r <directory> (i.e.+: add/ -:remove)
  + u = owner
  + g= group
  + o = others
  + a = all
  + r for read
  + w for write
  + x for execute
* chmod go-rwx <directory>
  + remove group and others’ right of rwx.
* chown newuser file.txt
  + changes the owner of file.txt to the user newuser.
* chown user:group mydirectory
  + This command changes the owner of mydirectory to the user user and the group to group.
* chown -R newuser:newgroup mydirectory
  + ecursively changes ownership for all files and subdirectories under mydirectory.
* chown :newgroup file.txt
  + This command changes only the group of file.txt to newgroup, leaving the owner unchanged

**Shell scripint - creating and writing a series of commands for a command-line interpreter or shell to execute.**

Shell script

* starts with #! /bin/sh (for posix shellscript) or with #! /usr/bin/env bash (for BASH script)
* then need to chmod +x <script.sh>

Basic syntax

* A; B
  + Run A then run B
* A | B
  + Run A and feed its output as input to B
* A && B
  + Run A if successful, run B
* A || B
  + Run A if not successful, run B
  + 0 indicate success
  + >0 indicate failure

Variables

* Creating variable
  + GREETING=”Hello World!”
    - No space around the “=”
* Use variable
  + echo “${GREETING}”
* To make variable exist in the program
  + export GREETING
* get rid of variable
  + unset GREETING
* Undefined variable = whitespace

Standard variables

* ${0}
  + Name of script
* ${1} to ${3} ….
  + Arguments passed to the script
* ${#}
  + Number of arguments passed to the script
* ${@} and ${\*}
  + All the arguments

Control Flow

* If statements
  + If XXXX; then
  + YYYY
  + fi
* for statement
  + for XXX; do
  + YYYY
  + Done
* Case statement
  + case XXX in
  + YYY
  + ZZZ
  + Esac
* echo "$(basename "${SHELL}")":
  + This uses the basename command to extract the file name (the last component) from the path stored in SHELL. The basename command takes care of stripping the path information and returns just the file name.
* echo "$(dirname "${SHELL}")":
  + This uses the dirname command to extract the directory portion of the path stored in SHELL. The dirname command returns the path without the last component (the file name).

Shell script in Make

.PHONY: all clean

figures=$(patsubst .dot,.pdf,$(wildcard \*.dot))

all: hello coursework.zip ${figures}

clean:

git clean -dfx

hello: hello.c library.o extra-library.o

%.zip: %

zip -r **$@** $<

%.pdf: %.dot  
dot -Tpdf $< -O **$@**

* $<: the first pre-requisite (e.g. matching .dot file)
* $@: the target (e.g. pattern ending with .pdf)

Java Maven Buildtool -> **Refer to exercise PDF!**

* The javac compiler turns source files (.java) into .class files;
* The jar tool packs class files into .jar files;
* The java command runs class files or jar files.

**Week 4 – Debugging**

**Debugging Tool - gdb**

* compile with -0g -g
  + cc -0g -g journal.c -o journal
  + now -g adds debugging info
  + -0g optimize for debugging
* Then run the program: gdb ./journal
  + Will enter (gdb) prompt
* (gdb) run <<<”Hello”
  + <<< : run the program with input “Hello” in. this case
  + In gdb now, run with “Hello” stdin
* (gdb) bt
  + Backtrace the function calls that led to the current point
* (gdb) b journal.c:14
  + Add breakpoint to line 14 of journal.c
* (gdb) d
  + Delete all breakpoints
* (gdb) inspect XXX (XXX:a variable)
* (gdb) x ADDRESS(0x…..)
  + Examine a variable/pointer at the given address
* (gdb) c
  + Continue after hitting a breakpoint
* (gdb) info
  + Get info about registers or variables
* (gdb) get help
  + Get help

**Debugging Tool – strace / ltrace**

* strace:
  + Trace system calls of a program: strace ./your\_program
  + Can look at opennat
  + Strace -e ‘/open.\*’ ./your\_program <<< hello 2>&1
* ltrace:
  + Trace library calls of a program: ltrace ./your\_program
* diff -u your\_program{2,3}.c
  + display difference between your\_program2.c and your\_program3.c
* valgrind:
  + valgrind ./your\_program
  + check for memory leaks