Linux CLI

This is a compilation of the notes from the sessions taken by Chap to teach the basics of Linux CLI to domain team members. Recordings of the sessions can be accessed here

Introduction

This is notes authored by Chanukya Nanduru, from a series of training sessions conducted by Prasad Chodavarapu, originally for scientific team members at Aganitha who wished to develop their technical skills. In hindsight, this material is proving useful to new technical team members as well. You can either read through the content here and wherever you find yourself lost, switch over to the video recording for the corresponding session. Recordings for all sessions are available under training > Tech for Domain team members > Recordings > Linux CLI folder in our google drive. To play a session's recording, click on the mp4 file in the folder for that session.

Each session's title is also hyperlinked to the corresponding video recording.

Session 1: Logging in

More often than not, computational chemists & bioinformaticians need high performance servers/clusters of servers to submit computationally heavy jobs. The first step to be able to do that is to 'log in' to these machines. Modern way to do this is using SSH (Secure Shell) keys.

Traditional way to do it has been the passwords, which is a kind of 'Shared Secret Authentication'. The problem with this method is:

- Password has to be available for authentication on the other side before one can login
- Both parties have to know the exact same information (i.e. 'Shared Secret').

A more preferred method is PKI based authentication (Asymmetric key encryption) in which we use 2 keys:

- A Public key: e.g., id_rsa.pub
    - Can be made public. Usually these are copied and put on the server (where an SSH server is running) to recognize users who are allowed to access a server.
- A Private key: e.g., id_rsa
    - This has to be private. Unless the private key corresponding to an authorized public key is accessed and utilized by the client who is trying to authenticate with the server, authentication will not be successful.

Digital certificates, https, GitHub & GitLab rely on this. This is like opening a bank locker which requires the keys from both the bank manager (public key) as well as the customer (private key). This concept also allows 'integrity preservation' i.e., not only is the information exchanged kept confidential (due to encryption), but also they are not allowed to meddle with the information (due to mapping of public<->private key pair).

A user can either have the same public<->private key pair on all his/her machines or have different key pairs in different machines.

SSL->TLS->SSH: SSL was the original idea that leveraged PKI for secure communication. TLS allows this to happen on the same ports where insecure communication is also happening. SSH leverages SSL/TLS for remote shell.

If there are 100 servers to which we need access to, copying the public key in all the 100 servers is not practical. To get around this, we can use a 'Directory' server where all users' public keys are published. LDAP (Lightweight Directory Access Protocol) provides a standard protocol to talk to directory servers such as Microsoft Active Directory.

Session 1: Using applications on Linux servers

E.g., sam tools, bam tools are examples of applications installed on servers

- Structure of a command line
    - command -o is same as command --option
    - Similarly, command -h is same as command -- help
    - command --option=value arg1 arg2
- GNU, an organisation that built UNIX tools, brought in some standardisation of commands
    - Parameters in [ ] are optional.
    - -- indicates end of options, to get around scenarios where an argument starts with a -
    - Variables can be assigned before a command is run.E.g., A=1, B=2 grep -v -- hello *.txt. These assignments are restricted to this command only and don't affect the current shell's environment values also.
- Spaces are to be avoided to the extent possible in file names & variable names as shell scripts have difficulty in handling them properly
- Shell is the program that reads the command, manipulates/processes it in its own way (e.g., wildcard expansion such as *, file redirections, running in foreground/background) and then passes it to the application we want to use.
- Every program in UNIX has 3 standard file handles available
    - stdin : it can read from
    - stdout: Print, echo statements write to this file handle by default.
    - stderr: Errors are written to this file.
- Programs will have flush output from memory for it to appear on console/in output files.

Session 2: Setting up the environment

- Along with the command input, the environment also has a bearing on the program output.
- Any environment in a nutshell is a bag of variable:value pairs. Some important variables are:
    - PATH: Shell depends on this.
        - PATH is such an environment variable that keeps track of certain directories. By default, the PATH variable contains the following locations:
            - /bin: Core programs present in all Linux distributions
            - /sbin: Accessible only to system administrators
            - /usr/bin: Some customization by distributions
            - /usr/sbin: Some customization by distributions
            - /usr/local/bin: User instalments go here
            - /usr/local/sbin
            - /opt : Optional. Similar to /usr/local/bin
        - /var: Growing files should go here. Hierarchy can be found here
        - $PATH is used to append to another path. You can also prepend to PATH using $PATH:
        - "No such file or directory"=> Not available in PATH
    - HOME: Shell depends on this. Indicates location of current user's home folder
    - PWD & OLDPWD: Shell uses this to go back to previous path when user types cd -

Command can manipulate values on the environment variables e.g., PWD.

However, these values are restricted to that particular shell by default (i.e., 'somatic mutation' & not 'germline mutation'). These values are not inherited to another shell/descendent applications unless they are exported (e.g., the command, export A = 1)

If we want to set the values on the environment variables & reuse them, we can store it. The files in which you cant store depends on the type of shell. In bash, this is done in bash_profile or bashrc. The former is read and executed when Bash is invoked as an interactive login shell while the latter is executed for an interactive non-login shell. These 2 files are automatically sourced when a shell starts. So, we should ensure that the scripts in these 2 files are idempotent/don't cause harm when run again.

Custom files can be maintained and sourced too using 'source' command

Quoting

', ", `, {

- ' : Values inside single quotes are taken as-is w/o any interpolation or expansion
- ":
  - Values inside double quotes undergo variable interpolation, symbol expansion (e.g., ~ means HOME, * is a globe wildcard character) etc.
  - Useful when you want to include spaces. E.g., cd "a b"
- `: Used when you want a command to be run and its output to be used as part of a string.e.g. below. It is advised to not use it. What you think of as the start of the second reverse quoted phrase will be seen as end of the first reverse quoted phrase by the shell. If you really need to nest commands, there's a safer syntax bash offers.

```
$ which grep
/bin/grep
$ ls -l `which grep`
-rwxr-xr-x 1 root root 219456 Sep 18  2019 /bin/grep


$(command1 arg1 arg2 $(command2 arg) arg4)
```

- {:Similar to ". For more, see 3.5.3 here

Session 3:Troubleshooting

What can go wrong when a command is run:

- Path problems: 'Command not found'
  - Mistyped the command or
  - It is correct but the PATH is not correct (because the first possible executable file is run and that is not what you may want to run) OR it is not found in the PATH
    - Fix by export PATH =
- In case of 'Permission denied' error, use 'which' command to know the path and review the permissions required(1st column below)
- In a group folder, 'sticky bit'(s) is required for a group to own the file and access it, though a particular user of the same group created it. So, if a user creates a file, without 'sticky bit'(s), group members cannot access it.
- Permissioning system: On a basic Linux distribution (unlike 'secure Linux' which additional restrictions set at global level)
  - Understanding permissions when ls -l is run:

lrwxrwxrwx	# files in folder	User	Group
• Permissions required to use the file • 1st bit: link (shortcuts in Windows) *or* directory *or* file • r,w,x for user, group, others =>3*3 =9 bits	1 if single file e.g., .txt file	• Though User/Group name is shown, UID/GID is mapped in the backdrop • This matters when we deal with Docker	

- File permissions
  - r-Read, w-write, x-Execute
  - (x)Execute on directory means permissions to enter it. .txt files does not have 'x'
  - Default permissions a file gets in a directory
    - Depends on the UID and GID creating it & 'umask'
  - File ownership:
    - Ownership of the file is assigned to the UID of the process that created a file.
    - Because, some special programs (such as setUID programs) like 'password' need to be run as a 'admin'(root) no matter which UID executes it.
    - Most of the times, it is your UID
  - Changing permissions
    - Numbering system
      - (r)Read-2^2 | (w)Write-2^1 |(x)Execute-2^0
      - 6 => r + w
    - Readable system: + / - r,w,x on ugo
  - Even if a folder is created as accessible to GID, everybody in the GID cant access it unless 'sticky' is declared
- UID
  - User ID: Every user is also a group. So, every UID has a GID too
- GID
  - Group ID

- First line in .sh (shell) file is magic line or #! (hasbang/shebang) line. Whatever follows #! Is considered the command to be run. E.g., #!/user/bin/python means that it is to be executed as a python script. To use whatever is in path, we should use /user/bin/env
- Fork-exec: Cloning the process before it starts another process(Fork) and replace with new code(exec)

Session 4: Performance monitoring

- Interactive command (e.g., grep) that quickly gives the output vs. Checking if a long running command/process is alive, running as intended, not facing any resource constraints
  - One way to monitor if the size of the destination file/folder such as log files is increasing (assuming the output is being redirected)?
- Using 'top' output, core-wise utilisation can be seen (Press 1)
  - Load avg is a moving average of # processes requesting for a processor. Anything < # cores is good. If > # cores=> queuing of processes waiting for processor
  - Also, memory breakdown is given: Free, used vs. buffering

- Resource utilisation: Bottleneck may be any of these. Performance improvement can be achieved through improvements across all these layers through vertical integration (e.g., How MacM1 was engineered)
    - Parallelizability=PROD(Nodes→Processors(CPUs,GPUs)→Cores→Threads)
    - Compute resources; Processors-CPU vs. GPU
        - These are the chips on the motherboard
        - GPU handles the matrix operations involved in computer vision (object recognition, image blending) better. Required for video games, zoom meeting backgrounds. Previously a graphics card handled these aspects leaving CPUs for other general purpose processes
            - SIMD allows parallel processing
        - Both have their own RAMs and there is a cost incurred in transfer from CPU's RAM to GPU's RAM. In MacM1, they are unified
        - Not advisable to run programs written for GPU on CPU e.g., GROMACS
            - GPU programs are CUDA (originating from C programming)
    - Memory
        - Normal memory vs. Swap memory
    - Storage volume/Disk
        - Slow: Old, Magnetic (Rotating)...Hard disk
            - RAID is a way to provide redundancy in data
        - Fast:Solid State Disks such as Flash drives
            - Mac has its image stored on SSD=> Faster bootup
    - Network: Interaction with other machines

Session 5: Editing & copying files

- Option 1: Using visual studio code (Preferred unless connecting via SSH is difficult in cases such as lack of direct connectivity e.g., connecting to a machine before connecting to the final machine of interest)
    - Connecting to a Linux server using VS code using 'View'> 'Command palette'>'Connect to Host…' where you can see all the hosts declared in the config file, connect to the remote machine and browse through files. You can also add a new host from 'Command palette'('Remote SSH' extension from Microsoft to be installed)
        - .ssh
        - config file which has aliases of machines & host-specific configurations
            - Host <alias name>
            - Host name
            - user (can log in as a cog as well)
            - SSH keys
- Option 2: Editors
    - vi
        - Command mode: Default when we enter. So, typing keys won't affect file
            - 'Esc' to come back to command mode from edit mode
        - Edit mode:
            - i - Insert | Starts editing at cursor location (Command -> edit mode)
            - a - Append | Starts 1 character to right (Command -> edit mode)
            - o - Opens new line | Starts a new line (Command-> edit mode)
            - To save file, first come into command mode and then,

            - :q! - Quit without saving. If no !, you will be asked if you are sure
            - :w - Write
            - :wq - Save and quit
        - :1 - To get to first line of file
        - :$ - To get to last line of file
        - :n - To get to nth line of file
        - Ctrl + G: Current location in the file
        - y - yank (Copy)
            - y1y - Copy 1 line
            - y100y - Copy 100 times
        - p - put/paste
            - p100p
        - If the file is long and spread across multiple screens (like Pgup, Pgdown)
            - Ctrl + b: Back one screen
            - Ctrl + f: Forward one screen
        - d - delete
            - dd or d1d: Delete 1 line
            - d100d - Delete 100 lines
            - : .,$ d - Delete(d) from current line (.) to last line ($)
            - :1,$ d - Delete from 1st to last line
            - :1,. d - Delete from 1st to current line
            - x - To delete 1 character at a time
            - 5x - To delete 5 characters at a time
            - Shift + d - Delete entire line
        - r - replace character
        - u - to undo last operation
        - . - to repeat last operation
        - Moving within a line
            - ^ - To move to beginning of line
            - $ - To move to end of line
    - emacs
        - Ctrl + Ctrl (because 26 commands are not enough!)
            - Ctrl + a : Start of line
            - Ctrl + e : End of line
            - Ctrl + k : Remove rest of line
            - Esc + d : Remove one word
        - bash is default in emacs
        - Gnu emacs vs. X emacs
    - vim (vi improved)
    - gvim (Graphical vim)
- Copying files
    - scp (Most popular)
        - ssh makes it possible
        - means ‘secure copy’
        - scp chanukya@own3.aganitha.ai:/~~home~~/chanukya/work…/<file> <destination> or . (current directory)
            - Put user name (chanukya) if local user name and that on machine is not same
    - rcp

        - rsh made it possible
        - Before public key<->private key authentication
        - password less login
        - Less secure. Replaced ‘r’ with ‘s’(secure)
    - rsync
        - Syncs only the changes if the file is already present instead of downloading the entire file again
        - Breaks each file into chunks and compares signatures between existing file and file to be copied
        - Hence, it is faster for cases where the file is already downloaded once
        - -t : To preserve the timestamp
        - atk sync.sh wraps rsync (Refer guides.own3.aganitha.ai)
            - Syncs the local work folder with that in the remote machine. But the danger of losing the folders available only locally exists.
            - Push excludes Notebooks
    - ftp
        - Important because large files cant be transferred using http (due to timeout issues). ftp has ‘ls’(listing) capability while http does not have that capability
        - To do anonymous FTP from servers that allow it, login as anonymous with email address as password.
        - ncftp is more advanced and accepts URLs. Logs in by default anonymously
            - ncftp <URL>
            - help
        - ncftpget <URL of the file>
- Text manipulation
    - :1,$ s/^/#/
        - Substitute(s) 1 to last line ($) ^ with #

Session 6: Text processing

All text processing in Unix one of two ways to represent patterns: Glob, Regex

- Search
    - Glob:
        - *.txt : matches any characters including none
        - ?.txt : matches any single character
        - [abc] : matches any one character in the bracket
        - [-] : matches character from range in the bracket

Limitation of Glob: Cannot search for multiple matches e.g., Multiple [abc] or ?. This is addressed by Regex by using the same characters used in Glob. But, they act on precedents e.g., * in Glob=> .* in Regex

    - Regex (Autoregex will simplify this using AI. But good to know the below basics)
        - Made famous by Perl.
        - Regex engines are different and relevant library/documentation has to be referred
        - Classical regex
            - | : or
            - [, ;]: , or ; i.e., Match with any single character within the bracket
            - () : To restrict application

                -
                - Quantifiers
                    - *: 0 or more of preceding; Doesn’t match itself
                    - *+ : 1 or more of preceding
                    - *? : 0 or 1 of preceding
                    - [abc] : matches any one character in the bracket
                    - [0-9] : matches character from range in the bracket
                    - [A-Za-z0-9_]: Multiple range expressions - A-Z, a-z, 0-9, _
                    - {0,n}: 0 to n of preceding i.e., {i,j}: i to j times of preceding
                - Other metacharacters
                    - . : Matches any character
                    - .* : Empty string or anything
                    - .+ : Except empty string
            - Perl introduced notations
                - \s: Match whitespace
                - \S: Match anything except whitespace
                - \w: Match any literal character/word i.e., [A-Za-z0-9_<all languages><punctuation>]
                - \W: Match anything except the above
                - \d: Match any digit
                    - \d\d or \d+: 2 digits such as 99
                - \D: Match anything except the above
                - \b: Word boundary
                    - Can be beginning or ending of line and not necessarily ‘ ‘
                - ^.*$: Match any character any # of times entire line
                - ^$: Match empty line
                    - ^ : Match beginning of line
                    - $: End
                - \
                - \. : To match .
                - Modifiers:
                    - /gm (Global + Beginning & end of line)
                        - /g (Global) : Only 1st match
                        - m (One line only)
                    - /i : Case insensitive
                    - /U: .*Can match 1 character or 100 characters. They will go for the highest match => Greediness, by default
- Replace
    - sed
    - awk
        - Space is by default the field separator. Can be changed to ‘ ‘
        - Comes in for multiple column files: Cut and join can be used too. But csvkit makes all this easy (To be installed using pip install csvkit)

There are three flavours of regular expressions that GNU `grep` and `sed` support.

1. Basic RegExp (BRE): Basic is the default

2. Extended RegExp (ERE): `-E` will turn on extended

3. Perl compatible RegExp (PCRE): `-P` will turn on perl compatibility

Here's the key paragraph in `grep` manual at
https://www.gnu.org/software/grep/manual/grep.html#Basic-vs-Extended

In basic regular expressions the characters '?', '+', '{', '|', '(', and ')' lose their special meaning; instead use the backslashed versions '\?', '\+', '\{', '\|', '\(', and '\)'. Also, a backslash is needed before an interval expression's closing '}', and an unmatched \) is invalid.

Session 7: Text processing contd. + Building & installing

- In both sed and awk, it is possible to specify restricted scope as shown below
  - /-/ {print $2} => Apply only on those lines where there is - ; 2nd column
  - /-/,/_/ {print $1} =>From the line that matches - to line that matches _ ; First column
  - Structure of an awk program
    - A begin block is used to initialise the variables
      - FS holds the field separator. This can be used instead of -F to change field separator from default
      - RS is record separator i.e, the line separator
    - From and to conditions can be specified by / /,/ /
    - Program is inside the {}
    - End block can be mentioned
- Installing programs downloaded from git repositories:
  - Avoid to get rid of dependencies & Leverage best practices
  - So, prefer pre-packaged. But do evaluate the sources
- If package manager is available
  - Not an installer. Maintains records of which file came from which package
  - Update, Upgrade functionality is also provided
  - Brew is the package manager on Mac
  - Apt-get is the package manager for Ubuntu
    - Apt: Wrapped version of Dpkg
    - Dpkg : Original
- If package manager is not available, well built C & C++ programs have the following standard mechanisms for building
  - configure
    - checkout & clone repository
    - get into directory
    - ./ configure automatically figures out the environment and produces the make file required for building(Autoconf)
  - make
    - compile packages and build executables
    - cmake etc. exist
  - make install
    - Copy build artefacts to appropriate folders
- `csplit` splits a file into many parts based on context provided by a regular expression. It is coming from GNU, the organization that distributes most of the popular text utilities. Given a text pattern, it can split a files into multiple files

Session 8: GNU parallel, Building & Installing (C based)

- If we have a program that is not parallelized (so that multiple cores can be utilised), GNU parallel can help. It also keeps the output of the order the same as the input. It can work on separate machines as well.
- We either download source files as .tgz (tar gz files) or zip files.
  - 2 kind of tar programs: gzip, bzip
- C program structure
  - Main function that calls other functions in the same or other file (
    - #include
      - <> : Header file in standard locations
      - " " : Header file in any place
        - Header file: Input, Output //Signature of the function
        - Source file: Contains the logic of the function
  - Compiling
    - Can compile to executable of shared objects (.so) using G++ compilers
    - Compiling single file (Executable ./a.out) vs. multiple files into a single executable file
    - If .h file is coming from a non-standard location, we can append to the list of /include directories using -I (Multiple -I allowed)
    - -l : link with library
  - Running
    - Dynamic loading
    - LD (Dynamic linker/Load dynamic) looks in /lib, /usr/lib, /usr/local/lib. Additional directories can be added
    - -L
- Common troubleshooting aspects (especially with RStudio)
  - Header file not found (e.g., bzlib.h is required for packages that require compression)
    - File not on the box :
      - Requires installation of source package using Apt-get.Generally, all such installations go to /usr/include.
        - /include, /usr/include, local/usr/include are the standard directories
      - If not a standard header file and not Apt-get package is available and a .tar file is downloaded and to be pointed to, use -I to point it to.
        - While running this, If symbol not found error crops up=>dynamic libraries are missing and ldd command is used to identify the missing library. Either add to the /lib or identify using .so
        - -
    - File available but in non-standard location
- Make files (Refer Dev Tools 101 classroom)
  - 2 types of execution strategies
    - Procedural/Imperative vs.
    - Dependency graph/Acyclic directed Graph (ADG): Efficient
  - Recompiling is made efficient by studying timestamps of the dependencies for changes
  - Including the -I can be automated:
    - Environmental variables
    - CFLAGS / CPATH: Header files can be specified using this

            - @ suppresses the output and does it silently
        - clean
            - .PHONY

Session 9: Building & Installing (Python based); Virtual env.

- Refer the exercise in Python 101:Creating Python application, packaging, pushing to server, downloading from server & install
    - Assignment 1: Creating project folder structure for organisation of source code repositories
        - atk-gen to instantiate templates of directory structures (Refer this link)
            - atk-gen.sh creates new projects/folders/modules with best practices template
        - Code organisation:
            - Code development a team member should be independent of the team member's code development. So, only the 'header' file should be a formal contract across the team.
            - Packaging units e.g., Libraries, modules that are deployable by themselves. These are brought together/stitch together libraries in the 'main' function.
        - Creating package from python modules that contain functions
            - setup.py (uses setuptools library, calls setup function) is used to create package
            - Used to package modules
            - make bundle (within Makefile) calls setup.py and creates packages (../wheel/dist)
                - sdist: source distribution where all source packages are available(tar.gz file)
                - bdist_wheel: binary distribution in the form of a wheel (.whl is a packaging mechanism for python)
            - pip-push to push the package built to the Pypi server
            - pip install first checks in in-house Aganitha Pypi server (pypi.own3.aganitha.ai) protected by credentials which are put in .netrc
    - Assignment 2: Virtual environments
        - Python modules usually are not independent. So, it is important to manage dependencies
            - requirements.txt: Mention all python modules required for the code to run
            - These need to be installed using pip install -r requirements.txt. However, it is installed into base python environment where the user may not have access to. Also, multiple projects may have conflicting or specific dependencies restraining a global installation. So, isolation of these project-specific dependencies is achieved through a virtual environment.
                - virtualenvwrapper is used at Aganitha
                - .virtualenvs contains all the virtual environments
                - Dependencies are installed in /lib folder of the virtual environment
            - Configuring PyCharm to use virtual environment

            - Open the folder containing code
            - Cancel the option to create virtual environment using PyCharm
            - Preferences>Python Interpreter> +> Select the python from /bin folder of the virtual environment
        - Python can have dependencies on C code. If .h files are not found, CFLAGS can be used to address them.
            - CFLAGS=<> pip install
    - Cython converts Python to C and then compiles in C.
        - Helps in performance improvement and bug identification

Some commands

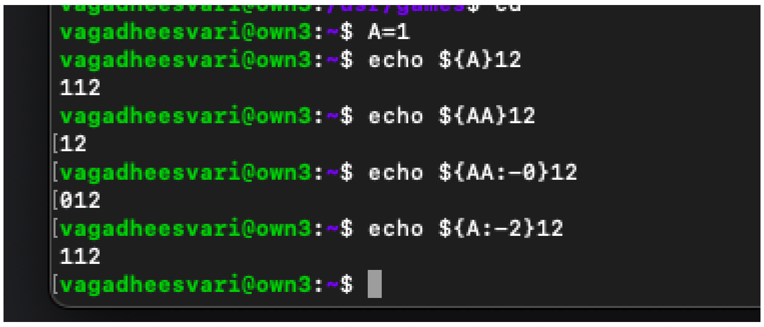
Session 1

- ssh
    - Means secure shell
    - OpenSSH is one kind of its implementation generally used in Linux
- ssh-keygen - t rsa
    - To generate a public<->private key pair using an rsa encryption algorithm. There are others as well such as dsa, ecdsa, ed25519 etc.
    - The generated key pair is stored in a hidden folder that can be accessed by using the commands :
        - cd .ssh
        - ls -l
- ssh - vv chap@own3.aganitha.ai
    - To login into a domain e.g., own3.aganitha.ai
    - Then # of 'v's can be 1 or 2 or 3. More the 'v's, more verbose you are asking the command to be in its output
    - If you give only the domain e.g., own3@aganitha.ai, the command will consider the user name on the local machine
- grep
    - grep - -help
      Usage: grep [OPTION]... PATTERN [FILE]...
      grep keyword file1.txt file2.txt >out.txt
      GREP OPTIONS
        - c : Displays count of number of lines where the pattern occurs
        - n: Displays line numbers along with the lines
        - v: Displays all lines except line matching patter
        - i : ignores case for matching
- jobs
    - To see the current jobs and their status
- Ctrl + Z
    - To suspend
    - sleep 5 &
- bg
    - To send the job to background
- kill %i

    - To kill job i
- Command ending with & runs in the background
    - If the network connection times out, shell gets killed and that in turn kills all the programs it started, including those running in the background
- Nohup
    - Means ‘No hung up’
    - If nohup is added before the command, shell death will not result in killing of the job started
- Screen
- Error and Output redirections
    - 1>out.txt is to redirect into out.txt (1 is implicit). Better to redirect O/P into files
    - 2>/dev/null is used for redirecting standard error
    - 2>&1 is used to redirect error into the same file as output
    - If no redirections are given, console mixes them up
    - > is only for the output
    - tee is the command to show in console and also file

Session 2

- env
    - env | grep “^A” —> Show’s variable value
- echo
    - echo $<variable>
    - echo $<variable><variable>



```
vagadheesvari@own3:~$ A=1
vagadheesvari@own3:~$ echo ${A}12
112
vagadheesvari@own3:~$ echo ${AA}12
[12
[vagadheesvari@own3:~$ echo ${AA:-0}12
[012
[vagadheesvari@own3:~$ echo ${A:-2}12
112
[vagadheesvari@own3:~$
```

echo $(AA:=2)12
212

echo $SHELL

- export
- alias
- source
- unset

Session 3

- ‘Which’
    - First possible location where the executable command is found
- id
    - List of all IDs
- getent

- Means ‘Get entity’
    - getent group <groupname> outputs list of members
- adduser
    - Add users to group
- usermod
    - Modifying users to group
- touch
    - Updates modification time if the file exists or creates if file doesn't exist
- umask
    - Default permissions set in numbering system
- umask -$
    - Default permissions set in readable format
- mkdir
    - Make directory
- chmod
    - o-w : Take away write permissions to others
    - o+w: Give write permissions to others
    - ugo+w: Give write permissions to user, group, others
    - ‘a’ is all of ‘ugo’
    - g+s: Files in that folder from that point onwards will be owned by GID of the parent folder
    - u+x
- man
    - Details of a command e.g., man chmod gives details of chmod
- chgrp
    - Change group
- chown
    - Change owner
    - <user>:<group>
- sudo
    - To change your status from user to super user
- -R
    - To make the command recursive
- . indicates current directory
    - ./ is a shortcut to indicate file in ‘this’ folder path/
- less
    - Read contents of text file one page (one screen) at a time
- file
    - Will tell what is the type of file e.g., file ‘which grep’, file a.txt
    - Looks at the first few lines of the file and not just the file extension
- vi
    - editor to edit an existing file or to create a new file from scratch

Session 4

- tail -f
    - tail: last few lines of a file
    - -f :follow
    - equivalent is *docker logs -f <container name>* in docker
- head
    - head: first few lines of a file

- -n : Last n lines
    - +n : From the nth line. + indicates counting from top
    - n: From the nth line. counting from bottom
- ps
    - shows only the processes started in current shell
- ps aux
    - all processes that are running
    - grep <process> will help us get the PID details
    - Column ‘PID‘ indicates Process Identifier
    - ps aux | grep <username> shows processes started by a particular user

      Eg: ps aux | cut -f 1 -d ' ' | grep chap

- /proc <PID>
    - shows the dynamic specific ‘folder’ (or metadata) of a process
    - e.g., cwd shows the directory where the process was started
- cut
    - ps aux | cut -f 1 *(Field 1)* -d ‘ ‘*(Separate by ‘ ‘)* | grep <username>
    - To separate a particular column from a table
    - you can ‘grep’ on this particular column
- top
    - It is an interactive command. e.g., 1=> core-wise performance, P/M to sort the table by process, memory etc.
    - Shows overall system performance, like Activity monitor of Mac or Task Mgr. of Windows
    - By each core, we get to see the utilisation
- htop
    - Same as ‘top’ but output is more ‘human’ readable
- lsblk
    - Shows all the bulk storage devices
- iostat -d 2
    - Shows performance of disk usage
- df -h
    - Checking available disk usage
- ping
    - Shows network speed (round trip time), data packet loss (%)

Session 5

- cat
    - create single or multiple files, view content of a file, concatenate files and redirect output in terminal or files
- pwd
    - shows present working directory
- set - o vi
    - changes editor to vi from emacs(default in bash)
- set -o emacs
    - changes editor to emacs
- : + up arrow
    - Previous command
- man <command>

- For help on a particular command w.r.t. syntax & options
- Ctrl + c to stop download
- rm <file>
  - Remove <file>
- curl
  - c url
  - Get all the files based on URL patterns of the file links
  - By default displays
  - CLI utility to interact with http pages
  - For hacking http
  - curl <URL> will print all the HTML returned by the URL
    - To save curl -o <URL> a.html has all that output stored in a.html OR
    - curl <URL> > b.html (using redirection)
- wget
  - Similar to curl but to get all the pages by crawling the links on a page. So, the danger of downloading the entire internet exists :-) using '-r'
  - Watch out for 'robots.txt' settings
  - But it does not have 'ls' (listing) capabilities. It uses the links on a page.
  - -d to limit the range of download (by domain or # pages)

Session 6

- grep - e "<regular expression>"
  - grep -e "[-_]\+" <filename>
- grep "[-_]" <filename>
  - - or _
- grep "[-_]*" <filename>
  - - or _
- `grep '[-_]+' <filename>`
  - `# matches nothing`
  - `Does not work because in basic regular expressions the characters '?', '+', '{', '|', '(', and ')' lose their special meaning`
  - Instead use backslashed versions as shown below
- `grep '[-_]\+' requirements.txt`
  - `escaping + works!`
  - `OR…the following`
- `grep -E '[-_]+' requirements.txt`
  - `Using -E will remove the need for escaping +`
- sed -e 's /- /_ /g' <filename>
  - stream editor => sed
  - / can be : or or ~ anything else but should not escape the pattern
  - Replace - with _ globally (g)
- sed -f <filename w patterns> <filename we should apply on>
  - filename contains all the patterns in the form of 's///'
- awk '{print $2}' <filename>
  - 2nd column of <filename> will be printed
  - Content in the {} is a program
- awk '{print $2, $3}' <filename>
  - 2nd & 3rd column of <filename> will be printed
- awk -F, '{print $2}' <filename>
  - To change the default field separator ' ' to ,

- awk -f <filename w columns> <filename we should apply on>
  - filename contains all the columns to be selected
- cut -d, -f2 <filename>
  - -d: delimiter
  - f2: 2nd field

Session 7

- echo hello | awk -F, -f {print FS}
  - , is the output
- echo hello | awk -F/ -f {print FS}
  - / is the output
- brew list
  - list of packages installed
- docker images | grep ubuntu
- docker run --rm -it ubuntu bash
- apt-get update
  - Ensures latest set of trusted sources
- apt-get install <package name e.g., python3>
  - Additional packages
  - Suggested packages
- man dpkg
- dpkg –help
- dpkg -L <package name>
  - Files from package installation
- dpkg -S <folder name>
  - Outputs the package that brought in the file
- dpkg -l
  - All packages installed
- For this test file

```
$ cat test.xml

<DocumentSummarySet>

    <a>

        <b>1</b>

    </a>

</DocumentSummarySet>

<DocumentSummarySet>

    <a>

        <b>2</b>

    </a>

</DocumentSummarySet>
```

the following awk script identifies which document each line is in

```
BEGIN { i = 1 }

/<Document/, /<\/Document/ { ///<\/Document/ is really not necessary

      print "Line " FNR " is in document " i

      if ($0 ~ /<\/Document/) { i++ } // Checks for the end of the document
with $0. In Reg Ex $ represents last character in a line, But in awk $n
represents nth column and $0 represents full last line

}
```

$ awk -f awk_simple.txt test.xml gives the following output

```
Line 1 is in document 1

Line 2 is in document 1

Line 3 is in document 1

Line 4 is in document 1

Line 5 is in document 1

Line 6 is in document 2

Line 7 is in document 2

Line 8 is in document 2

Line 9 is in document 2

Line 10 is in document 2
```

- `csplit -z -q -n 4 <fasta-file> '/>/' '{*}' //fasta file; Pattern is >. * is to repeat for as many sequences as there are instead of limiting it`
    - `-z is to suppress any zero-length files from being created.`
    - `-q is to suppress printing of resulting file sizes.`
    - `- 4 is to say that the resulting files should use a 4 digit sequence number in their names`

```
$ cat a.fasta

>seq0

FQTWEEFSRAAEKLYLADPMKVRVVLKYRHVDGNLCIKVTDDLVCLVYRTDQAQDVKKIEKF

>seq1

KYRTWEEFTRAAEKLYQADPMKVRVVLKYRHCDGNLCIKVTDDVVCLLYRTDQAQDVKKIEKFHSQLMRLME
LKVTDNKECLKFKTDQAQEAKKMEKLNNIFFTLM

>seq2

EEYQTWEEFARAAEKLYLTDPMKVRVVLKYRHCDGNLCMKVTDDAVCLQYKTDQAQDVKKVEKLHGK

>seq3

MYQVWEEFSRAVEKLYLTDPMKVRVVLKYRHCDGNLCIKVTDNSVCLQYKTDQAQDVK

$ csplit -z -q -n 4 a.fasta '/>/' '{*}'
$ ls xx*

xx0000  xx0001  xx0002  xx0003 //4 fasta files

$ head xx*

==> xx0000 <==

>seq0

FQTWEEFSRAAEKLYLADPMKVRVVLKYRHVDGNLCIKVTDDLVCLVYRTDQAQDVKKIEKF


==> xx0001 <==

>seq1

KYRTWEEFTRAAEKLYQADPMKVRVVLKYRHCDGNLCIKVTDDVVCLLYRTDQAQDVKKIEKFHSQLMRLME
LKVTDNKECLKFKTDQAQEAKKMEKLNNIFFTLM


==> xx0002 <==

>seq2

EEYQTWEEFARAAEKLYLTDPMKVRVVLKYRHCDGNLCMKVTDDAVCLQYKTDQAQDVKKVEKLHGK


==> xx0003 <==

>seq3

MYQVWEEFSRAVEKLYLTDPMKVRVVLKYRHCDGNLCIKVTDNSVCLQYKTDQAQDVK
```

Session 8

- for i in `seq 1..100`;do echo $i; done
    - Will do in sequence
- seq 10 | parallel echo {} +1 is {= ‘$_++’ =}
    - 1+1 is 2
    - 2+1 is 3
    - 3+1 is 4
    - It also keeps the output of the order same as the input
- tar cvzf <compressed filename> <directory>
    - create, verbose, (z)compress, filename
    - All contents of the directory go into the compressed file
- tar tvzf <compressed filename>
    - t: table of contents
- rm -fr split
    - removes folder
- tar xvzf <compressed filename>
    - Unzips

- tar xvjf <compressed filename>
  - Unzips
- ldd <program>
  - List of dependencies (such as dynamic libraries) required for a program to run
- echo $LD_LBIRARY_PATH

Session 9

- atk-list-commands.sh
  - Lists the installed atk programs
  - Works only if atk is installed
- atk-gen.sh
  - list of cookies available e.g., project-basic
- atk-gen.sh project-basic
  - Destination folder name, Project name & description of project to be entered after which folders are created
- atk-gen.sh python-module
  - Destination folder name, Module name, Author, Short description to be entered after which folders are created
- mv <filename1> <filename2>
  - Renames filename 1 to filename2
- make bundle
  - Produces ‘build’ directory and ‘test’ directory
- twine upload
  - To upload the created source distribution on PyPI.
- pip install -r requirements.txt
  - installs all the dependencies listed in requirements.txt
- python3 -m venv tutorial-env
  - Creates a ‘tutorial-env’ environment using venv
- mkvirtualenv <environment name>
  - Making virtual environment
- activate
  - Accessing the environment
- deactivate
  - Exiting the environment
- workon <environment name>
  - Accessing the environment
- lsvirtualenv
  - Lists names of all virtual environments
- rmvirtualenv
  - Remove a virtual environment

Some trivia

- sh (Bourne shell) ->ksh (Korn shell)->csh got killed ->GNU modernised UNIX->bash: Bourne again shell->Mac has the zshell to avoid GPL issues.
- UNIX was originally available as a spec of which there were many implementations
- Some docker containers don't have bash but the original sh to restrict the image size
- Extensions have no meaning in UNIX. They are DOS, Windows concept
- What is regular about regular expressions?

    - Difficulty is bounded
    - Tracking vs. not tracking state
        - Tracking state=>
            - Context specific
            - Push/Pop operations on stack
        - Not tracking state=>
            - .cfg (context free grammar)
            - Executed using FSM (Finite State Machine)

- Miniconda compiles libraries such as Numpy with Intel's proprietary optimizations (invovling eigen vector's computations) whose licence is available only through conda

References

https://tldp.org/LDP/GNU-Linux-Tools-Summary/GNU-Linux-Tools-Summary.pdf

https://docs.google.com/document/d/1ZGjbPHh5f8CUuMEcec3qkIiEjDKDTMBNLMTwqsF1At4/edit

https://www.gnu.org/software/bash/manual/html_node/Shell-Parameter-Expansion.html

https://github.com/whitead/nlcc

Basic intro to CUDA: helps you see how GPU code is written
https://developer.nvidia.com/blog/even-easier-introduction-cuda/