Unix

Unix Shell is both a **CLI** (Command Line Interface) and a Scripting Language.

- Bourne Again SHell (BASH) s the most popular Unix shell
- **$** is the prompt -> Indicates that unix is waiting for an input
- ls - List directory (all contents in the current directory)
- Shell is a program used to invoke other programs
- File system manages and organizes data into files and directories which hold files and other directories (sub-directories)
  - **pwd**- Print Working Directory (path)
  - At top of the file system is the root directory (/)
  - bin stores miscellaneous data
  - Users store personal user directories
  - Tmp store temporary files
- **ls -F**: -F is a switch/flag which adds a master to files and directories listed by ls to identify their types
  - / : directories
  - @: a link
  - * : executable
- **ls -F /** : ls command with -F flag and / is the argument which tells the command what to operate on (/ is the directory to operate on here)
- **ls -s** : Displays the size of files and directories with the name list
- **ls -S** : Sorts the files and directories by their sizes
- **ls --help**: For help options of ls command
- **man ls** : For a manual on ls. **man** command navigation uses:
  - scroll
  - B+spacebar for page down
  - /word for searching for a word
  - N for moving between multiple hits for a search
  - Shift + N to move backward between multiple hits for a search
  - q to quit
- **ls -lt**r : List all the files and directories in long form, sorted by time in reverse order
- **cd directory-name** : Change Directory
- **ls lSrh**: List all Directories and files in long form, sorted by size in reverse order and in human readable format
- **cd directory-name1/directory-name2** : Only while moving down the file system hierarchy
- **cd** .. :To go up the file system hierarchy by on**e directory** (Immediate Parent Directory)
- **ls -F -a or ls -Fa**: Shows the parent directory (../) and current working directory(./)
- **.bash profile** and similar files ar shell configuration files
- **cd** : without any argument points to the home directory (of main user)
- **cd** ~ : Returns us back to to the home directory (of the current user)

- **cd** - : Returns us back to our last working directory
- **mkdir directory-name** : To create/make directories. They are created in the present working directory or the given absolute path (if given)
- Directory names should ideally not use spaces or special character otherwise we need to us “ “ to refer to them
- **nano filename.txt** : is a text editor command used to edit a file. If the file is not already present, nano creates the file first. It can only work with plain character data. Other examples of text editors are emacs/Vim/Gefit (GUI based)
  - We can use ctrl+o to save our data in the file to disk
  - ctrl+x to exit the file after saving
- **touch filename.txt** : Creates an empty file (0 bytes). It is used to create a blank tet file before populating it with a text in a script of a program
- **mv source target** : Where source and target can be absolute paths or filenames. It can also be used for renaming files.
- **mv -i** : For interactive move which asks for confirmation before overwrtiting an already existing file with the target’s name at the destination (normal mv commands oerwrites the present file silently)
- **ls filename** : It is used to search only the filename (argument) int the current directory
- **ls -1:** Shows only names of files in current working directory in a list form
- **cp source target** : Copies a file
- **cp -r source target** : Copy all files in the source directory recursively to the target directory
- **rm filename** : Used to remove a file
- **rm -i filename**: Remove interactively (asks for confirmation before deleting)
- **rm -r directory_name:** rm can not remove directories directrly. Hence, we have to use -r to recursively remove all contents of a directory and the directory itself. This cannot be used with -i (no confirmation)
- **mv filename1 filename 2 target**: In case more than 2 arguments are given to mv, it assumes that last argument is the target directory (cannot be a file) and all previous arguments as the source files. Same can be used with cp
- *** is a wildcard** : It matches 0 or more characters.
  - **? is also a wildcard** : It matches only 1 character
  - **Example**: We have 3 files - Methane.txt, Ethane.txt and Propane.txt
    - **ls *ane.txt** : Selects all 3 files
    - **ls ?ane.txt** : Selects no file
    - **ls ???ane.txt** : Selects only e t h ane.txt

Pipes and Filters

- .pdb extension is Protein Data Bank Extension, which is effectively a text format that specifies the type and position of each atom in a molecule
- **wc** : Word count command

- **wc filename** : Gives the word count of the argument file in the format - number of lines number of words number of characters filename
    - **wc -l** : for only lines and filename
    - **wc -w** : Number of words and filename
    - **wc -m** : number of characters and filename
- **wc -l *.pdb > lengths.txt** : Here > directs the output of the earlier command to a text file lengths,txt and saves the output in that file. It creates the file if not already present and overwrites a file if already present
- **>>** : Operator is used to append the new contents in a file if the file is already present, instead of overwriting it like >
- **cat filename.txt** : Prints the contents of a file. It is used to concatenate and print the contents of one file after the other and so on in case there are more than one files as **arguments**
- **less filename.txt**: Does similar work as cat but displays only one screen full of content at a time. We can press **b** to show another page or **q** to quit
- **sort filename**: Sorts and prints the contents of a file in alphanumeric order
- **sort -n filename**: Sorts and prints the contents of a file in numeric order
- **sort -n filename > filename-sorted** : Stores the sorted output in a file. We ideally should use a new file and not overwrite the same file as it produces wrong output
- **head -n 1 filename**: To get the first line of a file. **-n 20** is used to get the first 20 lines of the file
- **echo** : Command is used to print text given to it as argument
- **tail -n 2 filename** : Prints the last 2 lines of a file
- **|** : Pipe operator.
    - **command 1 | command 2**: The output of command 1 is passed as the input to command 2
    - e.g: **sort -n lengths.txt | head -n 1**: First sorts the file lengths,txt and fives the sorted list to the head command which prints the first line of sorted output
    - **wc -l *.pdb | sort -n | head -n 1 :** Line counts of all pdb files are passed to sort command which sorts them numerically. This output is passed to head command which finally prints the first line of the sorted line lengths
    - Pipe is used for nesting
- A filter is a program that transforms a stream of input into a stream of outputs
- **cut** : Is a command used to extract certain sections of each line in a file. It by default expects a tab delimiter
    - -d is used for specifying the delinter. Only one character can be used as a delimiter at a time
    - -f is used for specifying the files
    - **cut -d , -f 2 animals.txt**: Cut out second field/column of animals.txt which is comma (,) delimited
- **uniq**: Filters out adjacent matching lines in a file (It removes any number of adjacent duplicates). For example : **cut -d , -f 2 animals.txt | sort | uniq**

- **uniq -c**: Gives the count of number of time a line (adjacent duplicates) occurs in the input
- **Wildcard expression [ ]** : **ls *[AB].txt** : Matches the files with any name but ending with either A or B before .txt. Alternatively, we can also write **ls *A.txt *B.txt**

Loops

- **for thing in list-of-things**
  do
  operation using $thing
  done
  - Indentation is not compulsory here
  - $thing returns the value of variable thing
  - For example
    ```
    for filename in *
     do
            head -n 2 $filename | tail -n 1
     done
    ```
    Prints the second line of each file present in the current working directory
- **>** : is also a prompt sed in loops when the unix is awaiting input from the user
- To copy a number of files with new names:
  - for filename in *.dat
  - do
  - cp $filename duplicate-$filename
  - done
- **bash script-name [arguments]** : Command is used to run a script
- **history | tail -n 5** : Gives the last 5 commands run. We can then use !command-number to run a particular command from the history
  - Last run command is printed by **history | tail -n 2 | head -n 1** . This is because the history command itself is registered as the last run command in the history
  - Alternatively, we can use **ctrl+r** to reverse i history (history search mode). It finds the most recent command in history searched by the text we enter
  - **!!** : Gives the last run command
  - **!$** : gives the last word of the last command
- Nested loops are for loops within loops

Scripts

- Use **#** in scripts to add comments. (only single line comments are possible)
- Use **$1 , $2**, … to pass arguments/parameters to a script
- Run script using **bash script-name.sh param1 param2**

- If we don't know the number of arguments beforehand, we can use **$@** which signifies *all of the command line arguments given to the script*
- **bash -x** : Can be used to run a script in debug mode which prints out each command as it runs, to make it easier to locate an error

Finding Files

- **grep** : global/regular expression/print - It finds and prints lines in a file that matches a pattern given to it as an argument
  - **grep word(s)-to-find filename** : Here word-to-find is case sensitive
  - **grep -w "word to find" filename** : -w is used for word wrap
  - **-n** : gives the number of lines we found the word in
  - -**v** : inverts the grep search and searches for lines not containing the specified word
  - **-c** : Selects only the words that are required
  - **-i** : For case insensitive
  - **grep - help**; for all options
  - grep can have patterns to indicate wildcards/regular expressions eg. - **grep - E "^.o" haiku.txt** - to find all lines having o in the second character of line. **-E** is used for extended regular expression usage. **^** anchors the match to the start of line and **.** Matches a single character (similar to ? in shell)
  - **- o**: finds only part of matched expression
  - To find the no. of times a name occurred in a file : grep - ow name filename
- **find .**: To find all the names of file and directing under **(.)** - current directory (also under its child directories)
  - **find . -type d** -find only types - directories
  - **find . -type f** - find only types - files
  - **find . name "*.txt**" - finds by matching name (here only .txt files). However, Shell runs wildcard commands before the actual command. Hence, we should put it "*.txt" to avoid this
  - Find returns the paths, we can have it as **$(find . -name "*.txt")**
- To find word counts of all text files: **wc -l $(find . -name "*.txt")** : When Shell runs, it first runs what is inside ( ) & replaces it with the command's output.
- Using grep & find together - grep "FE" (find .. - name "*.pdb") : finds all files with pdb extension in current folder's parent folder and grep then finds all lines having "FE" (iron) in those .pdb files