Week 2 Notes

- Multiple uses of / is as good as one
  - ie : `cd usr//////bin` will take you to `usr/bin`
- The root folder / is its own parent
  - ie : if you do `cd ..` within the root directory you stay in the same directory.
- Options / Flags can be written in multiple combinations
  - `ls -l level1 -di`
  - `ls -d level1 -il`
  - `ls level1 -ldi`
  - `ls -ldi level1`
- long formats for options are also available
- `ls -a` is equivalent to `ls --all`

Commands

- `ls`
  - R flag lists all subdirectories recursively
  - Passing directory name to ls shows what is within that directory. ie : ls -l level1
  - d flag displays details of a folder without traversing inside it. it : ls -ld level1
  -
- `ll`
  - a shortcut for the ls -la command
- `which`
  - which *command* will show the location of the command
  - `which less` will show usr/bin/less
- `whatis`
  - gives a brief description of the command
- `alias`
  - give a nickname to a frequently used command
  - usage : `alias ll = 'ls -l'`
  - Just typing alias will show a list of aliases
  - `alias date = 'date -R'`
  - If the command is executed by typing the whole path eg : `/usr/bin/date` the alias is not invoked. (`cd /usr/bin` and `./date`)
  - An alias can be escaped by prefixing a \ ie: `\date`
- `unalias`
  - used to remove an alias
- `rmdir`
  - removes an empty directory
- `ps`
  - displays current processes
  - `ps --forest` - which process has launched which child process.

    - `ps -f` - displays parent process id
    - `ps -ef` - all the processes running in the operating system now
    - PID is the process ID , PPID is the parent process ID.
    - PID 1 is `/sbin/init`
- `bc` - bench calculator
    - exit using `Ctrl + D`

Commands to know contents of a text file

- `less`
    - displays the content in one screen
    - ls -l /usr/bin/less shows that the command takes 180KB
- `wc`
    - prints newline,word and byte counts for the file
    - the -l flag shows just the number of lines
- `head`
    - head profile displays the first ten lines
    - use -n flag to specify the number of lines
- `tail`
    - tail profile displays the last ten lines
    - use -n flag to specify number of lines to be displayed
- `cat`
    - in /etc , cat profile would just dump contents on the screen without any further prompts.
    - disadvantages : cant move back and forth to view page by page, can't come out half way through.
    - if the file is very long cat is not the best way to look at the content.
- `more`
    - similar to less. Allows page by page viewing
    - ls -l /usr/bin/more shows that the command takes 43KB

Knowing more commands

- `man`
- `which`
- `apropos`
    - For a keyword it shows you all the commands which have that keyword in the description
    - Used to discover new commands
    - If you type `ls -l /usr/bin/apropos` you see that it is a symbolic link to whatis, but the outputs are different : Why?
    - Reason : In Linux every executable will know in what name it has been invoked - can have different behaviour depending on the name that invoked it.
    - It also has the same output as `man -k` : Searching for a keyword
- `info`
    - Allows browsing through commands using the cursor
    - Can go back using < or 'shift'+','

- `whatis`
- `help`
  - displays keywords reserved for the shell being run
- `type`
  - displays what type of command it is
  - type type shows that it is a 'shell built in' being offered from the shell and not the os
  - type ls shows that it is aliased with some option. which ls shows that it is coming from os because there is an executable available.

Multiple Arguments

🔗 **Recap : Arguments and Options**

- Options are enhanced features of the command
- Arguments are specific names of files or directories

- Second arrument behaviour and interpratation of last argument should be seen in the man pages
- Recursion is assumed for `mv` and not `cp`

- recursion is assumed for some commands and should be explicitly stated in others
- For copy command recurssion is not assumed
- `cp dir1 dir2` need not work. dir1 has 2 files in it.
- `cp -r dir1 dir2` works - recurssion is specified explicitly.
- `mv dir1 dir3` works - it just renames the directory.

- `touch file1 file2 file3` creates all 3 files in one go with identical timestamp.

Links (Hard Links and Soft Links)

- Can determine whether a link is HL or SL by looking at the Inode numbers
  - Hard links will have the same inode numbers
  - Soft Link will have different inode numbers
  - If you delete a certain file using the `rm` command (`rm` unlinks the file from the filesystem. the data is still at the memory location. `shred` for permanant deletion)
    - Its hard link will still give you access to the original file data.
    - Its soft link will not work
- `ln -s source destination` to create symbolic link. `ln -s file1 file2`
  - file2 is a separate inode entry but it is just a shortcut to file1
  - file2 has only 1 hardlink.
- `ln source destionation` to create a hard link . `ln file1 file3`
  - file1 and file 3 have the same inode number - They are basically the same file.
  - file1 and file3 have 2 hard links when we do `ls -li`
- You can create a Soft Link `ln -s ../dir/filex fileSL` but creating a hard link using `ln ../dir/filex fileHL` will not work.

    - the first/source-file parameter is interpreted in the case of hard link creation and not in soft link creation
    - In the above example, assume that `../dir/filex` does not exist.
    - soft links useful in version control systems

File Sizes

- `ls -s`
    - file size appears in the first column
- `stat`
    - in `/usr/bin` we look at `stat znew`
    - Gives information about the size, how many blocks are being occupied
    - Here the size is little more than 4kb
    - `stat zmore` shows that it takes less than one block
- `du`
    - in `/usr/bin` we look at `du znew` or `du -h znew`
    - Gives information about the size
    - Here the size is displayed as 8.0KB since there is a block overflow.
    - This means that files that are smaller than the block size will actually take up a whole block
    - `du -h zmore` shows that it occupies one block - around 4.0K
- Role of block size
    - explained in stat and du

In-Memory File Systems

- `/proc`
    - Is an older system
    - `ls -l` will display several zero-size files, even though we can read content from them.
    - These are only a representation and not real files on the HDD.
    - `less cpuinfo` - information about the cpu
    - `cat version` - information about the OS. Also accessible using `uname -a`
    - `cat meminfo` - information about the memory - also `free -h`
    - `cat partitions` - information about the partitions - also `df -h`
    - The `kcore` file appears to take huge space - Shows maximum virtual memory that the current linux os is able to handle. 2^47 or 140 TB
- `/sys`
    - Used from Kernel v2.6 onwards, however information about various processes that are running are still stored in the /proc directory itself.
    - Much more well organised than /proc
    - eg : `sys/bus/usb/devices/1-1` points to a specific usb device.
- These are directories that are visible in the root folder. They are not on the disk but only in the memory.
- Important system information can be viewed from these directories in a read-only manner.

Shell Variables

- Makes it possible to communicate between 2 processes very efficiently. Need not write and read the filesystem.
- Security Concern : Some information that you write to the filesystem may be visible to other processes.
- Shell variables are available only within the shell or its child processes.
- `echo` prints strings to screen
  - uses space as a delimiter so multiple spaces between words are ignored. For multiple spaces, enclose the string in quotes.
  - can print a multi-line string by using double quotes and not closing it
  - ** Difference between ' and " **
  - `echo $USERNAME` and `echo "$USERNAME"` give the same result but `echo '$USERNAME'` is not interpreted to give the value of the shell variable.
  - ** Escaping to prevent interpretation **
  - `echo "username is $USERNAME and host name is \$HOSTNAME"`
  - Escaping is usefule when you want to pass on the information to a child shell, without it being interpreted by the shell launching it.
- `echo $HOME` prints values of variables
  - By convention every shell variable starts with a Dollar
- **Commonly used shell variables**
  - `$USERNAME` eg : `echo "User logged into system now is : $USERNAME"`
  - `$HOME`
  - `$HOSTNAME`
  - `$PWD`
  - `$PATH` - variable contains a list of directories which will be searched when you type a command. When ever you type a command the system scans these paths from left to right to see if the command is in the directory.
- Commands like `printenv , env , set` to see variables that are already defined
  - `printenv` displays all the shell variables defined in the shell that you are running.
  - `env` gives the same output
  - `set` displays some functions defined to interpret what you are typing on the command line.
- **Special Shell Variables**
  - `$0` : name of the shell eg `bash` or `ksh`
  - `$$` : process ID of the shell
  - `$?` : return code of previously run program
  - `$-` : flags set in the bash shell . The man page for bash shows the meaning of the flags.
- **Process Control** `echo $$`
  - use of `&` to run a job in the background
  - `fg` - bring process to foreground
  - `coproc` - run a command while also being able to use the shell
  - `jobs` - list programs running in the background
  - `top` - See programs that are hogging the CPU or memory (refreshed every second)

    - `kill` - kill process owned by you
- **Program Exit Codes** `echo $?`
    - exit code always has a value between *0 and 255*
    - 0 : Success
    - 1 : Failure
    - 2 : Misuse (insufficient permissions)
    - 126 : command cannot be executed (usually due to insufficient permissions to execute a file)
    - 127 : command not found (usually due to command typos)
    - 130 : processes killed using control+c
    - 137 : processes killed using `kill -9 <pid>`
    - If the exit code is more than 256 then the exitcode%256 will be reported as the exit code
    - `exit 0` or `exit 1` or `exit <n>` exits with exit code n
    - Used when there are command dependencies (ie: run second command only if first command completes successfully)
- **Flags set in bash** `echo $-`
    - h : locate hash commands
    - B : braceexpansion enabled
    - i : interactive mode
    - m : job control enabled (can be taken to bg or fg)
    - H : !style history substitution enabled
    - s : commands are read from stdin
    - c : commands are read from arguments

Linux Process Management

- `sleep` command to create processes
    - usage : `sleep 3` for 3 seconds
- If you have a command running in the Foreground for a long time but you need to write something else on the command line :
    - kill the process
    - suspend the process
    - run it in the background `coproc sleep 10` - When complete it gives a message.
- `coproc` is a shell keyword. No manual entry for it.
    - To learn more about a shell key word use `help coproc`
    - a running background process can be killed by process id (use : `ps --forest` to find PID and `kill -9 <pid>`)
- A command followed by an `&` means that it is being assigned to the background
    - Executing the command `fg` will bring it back to foreground
- `jobs` is a shell builtin - it lists active jobs in the current shell
- `top` shows processes taking up maximum cpu and memory. Exit gracefully by pressing Q
- `Ctrl + z` suspends a process.
    - Suspended processes can be seen with `jobs`
    - Can be brought back to foreground using `fg` command

- `Ctrl + c` kills a process
- `fg` is a shell builtin
- `bash -c "echo \$-"` creates a child shell, gets the value of `echo $-` , gives the output to the parent shell
  - `bash -c "echo \$-; ps --forest;"` - multiple commands separated by ;
  - `bash -c "echo \$$ ; ps --forest ; exit 300"` : custom error code mod 256 = 44
- `history` displays a list of commands that have been run on that computer
  - `!n` executes command line no n displayed by `history`
  - useful for repeating long commands
  - The `H` flag in bash means the history is being recorded
- Brace expansion option `B`
  - if you type `echo {a..z}` character in the ASCII sequence will be expanded.
  - In combination `echo {a..d}{a..d}` will display all possible combinations of the 2 alphabets.
  - `*` exapnds to all the files in the current directory
  - `echo D*` lists all the files begining with D.
  - Examples :
    - `mkdir {1..12}{A..E}` or `rmdir {1..12}{A..E}` or `touch {1..12}{A..E}/{1..40}`
- `;` acts as a separator between individual commands eg : `echo hello ; ls`

REPLIT CODE WITH US

Link to Replit

- `date -d "2024-04-01" +%A` - Day of the week for given date
- `file --mime-type somefile` - mime type of a given file
- `mkdir {1..12}{A..E}`
- `rmdir {1..12}{A..E}`
- `touch {1..12}{A..E}/{1..40}`
- `lscpu | grep -i "model name"| cut -d ":" -f "2"`