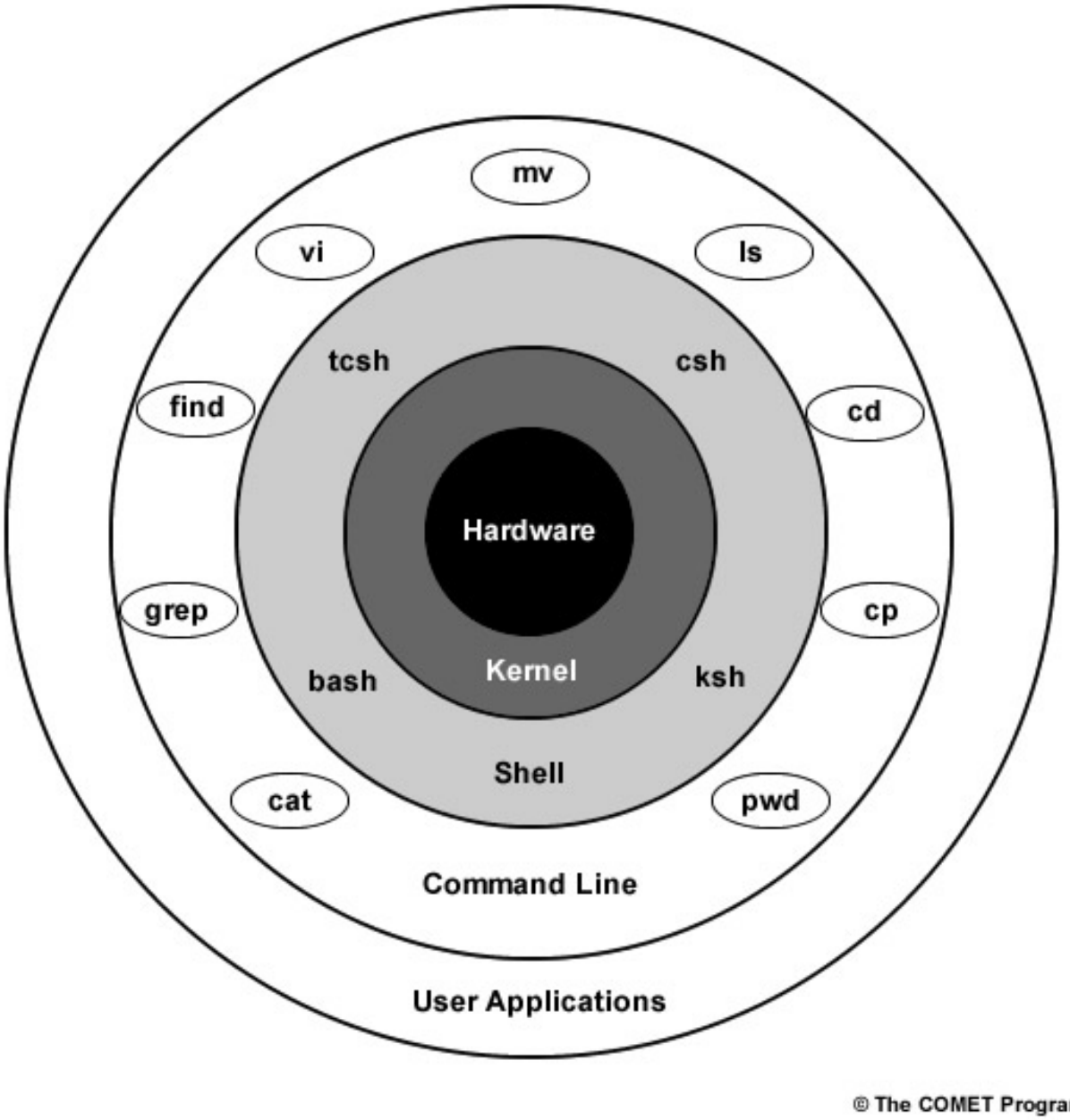
Lss (CIE 1)

UNIT 1
Unix Architecture



The architecture of UNIX is basically divided into four main layers-

1. Kernel
2. Hardware
3. Shell
4. Files and directories

The first layer kernel deals with all the hardware. Hardware is the second layer. The third layer known as shell acts as the bridge between the user, user commands

and predefined UNIX commands. Last but not the least the final layer is the user. This means the whole operating system is visible to the user from the shell itself.

1. Kernel

Amongst the four layer's kernel is the most powerful one.
Kernel program has the power to start or stop a program.
It also suggests which program to be selected, when two program demand for the same resource at the same time.
Kernel is divided into two categories and listed below

1. Process management.
2. File management.

- **Process Management**

The resource allocation of CPU, memory, and services are few things which will be handled under process management.

- **File Management**

File management deals with managing all the data in files needed by the process while communicating with devices.

The main operations done by the kernel are

1. Kernel ensures the running of user-given programs is done on time.
2. Plays a role in memory(resource) allocation.
3. Manages the swapping of programs between primary and secondary memory.

2. Hardware

Hardware can be defined as the system components which are seen through the human eye and be touched like keyboard, monitors, etc.

3. Shell

The shell can easily be defined as the software program which acts as a communication bridge between kernel and user.
When the user gives the commands, the shell reads the commands, understands them and then sends a request to execute the program.
Then when the program is executed, it again sends the request to display the program to the user on-screen.
The shell can also be called a command interpreter.
Various tasks which shell ask the kernel to do are

1. File opening.
2. File writing.
3. Executing programs.
4. Obtaining detailed information about the program.
5. Termination of the process.
6. Getting information about time and date.

4. Unix Files And Directories

Directories
Directories in Unix have name, path, files, and folder.
These are stored in the up-side-down hierarchical tree structure.

Files
Content of data, text and program instructions are stored here. The main workflow of files is-

- Store user information like an image or some content.
- Mostly located under a directory.
- It does not store the data of other files.

Features of UNIX

1. Multiuser System :
Unix provides multiple programs to run and compete for the attention of the CPU. In UNIX, resources are actually shared between all the users, so-called a multi-user system.
For doing so, computer give a time to each user. So, at any instant of time, only one user is served but the switching is so fast that it gives an illusion that all the users are served simultaneously.

2. Multitask System :
A single user may run multiple tasks at the same time. Example : Editing a file, printing another on the printer & sending email to a person, and browsing the net too at the same time.
The Kernel is designed to handle user's multiple needs.
But the catch is only one job can be seen running in the foreground, the rest all seems to run in the background. Users can switch between them.

3. Piping:
The Unix developers thought about keeping small commands for every kind of work. So Unix has so many commands, each of which performs one simple job only. You can use 2 commands by using pipes ('|'). Example : $ ls | wc Here, | (pipe) connects 2 commands to create a pipeline.

4. The UNIX Toolkit :
Unix has a kernel but the kernel alone can't do much that could help the user. So, we need to use the host of applications that usually come along with the UNIX systems. The applications are quite diversified. General-purpose tools, text manipulation utilities (called filters), compilers and interpreters, networked programs, and system administration tools are all included. With every UNIX release, new tools are being added and the older ones are modified/ removed.

5. Pattern Matching :
Unix provides pattern matching features. The '*' is a special character used by the system to match a number of file names.
There are several other meta-char in UNIX.

6. Programming Facility :
Unix provides shell which is also a programming language designed for programmers, not for casual end-users. It has all the loops and variables required for programming purposes.

Basic unix commands

1. who : The '$ who' command displays all the users who have logged into the system currently.
$ who
Output: harssh tty2 2017-07-18 09:32 (:0)

2. pwd : The '$pwd' command stands for 'print working directory' and as the name says,it displays the directory in which we are currently at.
$ pwd
Output: /home/harsh

3. mkdir : The '$ mkdir' stands for 'make directory' and it creates a new directory.
$ mkdir newfolder

4. rmdir : The '$ rmdir' command deletes any directory we want to delete.
$ rmdir newfolder

5. cd : The '$ cd' command stands for 'change directory' and it changes your current directory to the 'newfolder' directory.
$ cd newFolder

6. ls : The 'ls' command simply displays the contents of a directory.
$ ls
Output: Desktop Documents Downloads Music Pictures Public Scratch Templates Videos

7. touch : The '$ touch' command creates a file(not directory) and you can simple add an extension such as .txt after it to make it a Text File.
$ touch example
$ ls
Output: Desktop Documents Downloads Music Pictures Public Scratch Templates Videos example

8. cp : This '$ cp ' command stands for 'copy' and it simply copy/paste the file wherever you want to. In the above example, we are copying a file 'file.txt' from the directory harssh to a new directory new.
$ cp /home/harssh/file.txt /home/harssh/new/

9. mv : The '$ mv' command stands for 'move' and it simply move a file from a directory to anothe directory.In the above example a file named 'file.txt' is being moved into a new directory 'new'

$ mv /home/harssh/file.txt /home/harssh/new

10. rm : The '$ rm ' command for remove and the '-r' simply recursively deletes file. Try '$ rm filename.txt' at your terminal
$ rm file.txt

11. chmod : The '$ chmod' command stands for change mode command. Basically there are 3 modes that we can use with the 'chmod' command
1. +w (stands for write and it changes file permissions to write)
2. +r (stands for read and it changes file permissions to read)
3. +x (generally it is used to make a file executable)
$ chmod +w file.txt
$ chmod +r file.txt
$ chmod +x file.txt

12. cal : The '$ cal' means calendar and it simply display calendar on to your screen.
$ cal
Output : July 2017
Su Mo Tu We Th Fr Sa
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31

13. file : The '$ file' command displays the type of file.
$ ls
Output: Desktop Documents Downloads Music Pictures Public Scratch Templates Videos
$ file Downloads
Output: Downloads: directory

14. grep: The '$ grep' command searches the specified input fully(globally) for a match with the supplied pattern and displays it.
In the example, this would search for the word 'picture' in the file newsfile.
$ grep picture newsfile

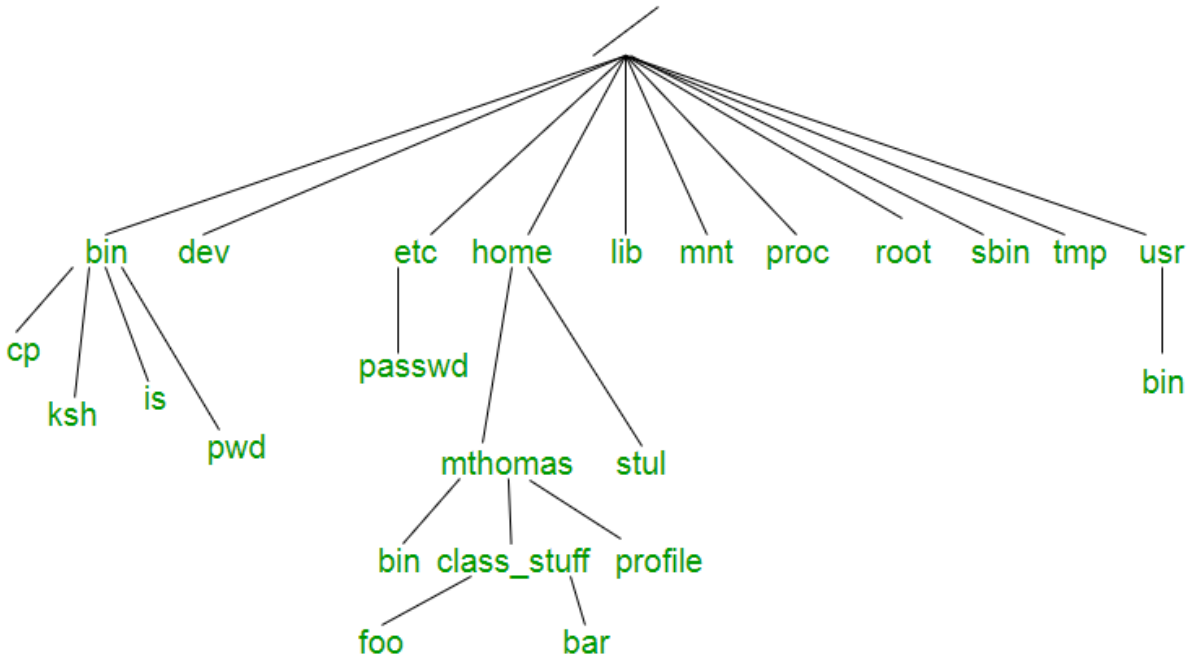
15. man : The '$ man' command stands for 'manual' and it can display the in-built manual for most of the commands that we ever need.In the above example, we can read about the '$ pwd' command.
$ man pwd

18. passwd : The '$ passwd' command simply changes the password of the user. In above case 'harssh' is the user.
$ passwd

Output: Changing password for harssh.
(current) UNIX password:

19. clear **:** The '$ clear' command is used to clean up the terminal so that you can type with more accuracy.

Unix File System



- ✓ It has a root directory (**/**) that contains other files and directories.
- ✓ Each file or directory is uniquely identified by its name, typically called an **inode**.
- ✓ By convention, the root directory has an **inode** number of **2** and the **lost+found** directory has an **inode** number of **3**. Inode numbers **0** and **1** are not used.
- ✓ File inode numbers can be seen by specifying the **-i option** to **ls command**.
- ○ It is self-contained. There are no dependencies between one filesystem and another.

Sr.No.	Directory & Description
1	**/** This is the root directory which should contain only the directories needed at the top level of the file structure
2	**/bin** This is where the executable files are located. These files are available to all users

3	**/dev** These are device drivers
4	**/etc** Supervisor directory commands configuration files, disk configuration files, valid user lists, groups, ethernet hosts, where to send critical messages
5	**/lib** Contains shared library files and sometimes other kernel-related files
6	**/boot** Contains files for booting the system
7	**/home** Contains the home directory for users and other accounts
8	**/mnt** Used to mount other temporary file systems, such as **cdrom** and **floppy** for the **CD-ROM** drive and **floppy diskette drive**, respectively
9	**/proc** Contains all processes marked as a file by **process number** or other information that is dynamic to the system
10	**/tmp** Holds temporary files used between system boots
11	**/usr** Used for miscellaneous purposes, and can be used by many users. Includes administrative commands, shared files, library files, and others
12	**/var** Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data
13	**/sbin** Contains binary (executable) files, usually for system administration. For example ***fdisk*** and ***ifconfig*** utlities
14	**/kernel** Contains kernel files

chmod command can be use to change different permission configurations. chmod takes two lists as its arguments: permission changes and filenames. You can specify the list of permissions in two different ways. One way uses permission symbols and is referred to as the symbolic method. The other uses what is known as a “binary mask” and is referred to as either the absolute or the relative method.

Symbolic Method
The symbolic method of setting permissions uses the characters **r, w, and x** for read, write, and execute, respectively. Any of these permissions can be added or removed. The symbol to add a permission is the **plus sign, +**. The symbol to remove a permission is the **minus sign, -.**
chmod :- File Permissions in Symbolic Method
Description
r: Read
w: Write
x: Execute (also gives permission to change into a directory)
X: Execute only if it is a directory or has execute permission for some user
s: Set user or group ID on execution
t: Sticky bit
u: Permissions granted to user who owns the file
g: Permissions granted to users in the file's group
o: Permissions granted to owner of the group and users in the file's group

r w x permissions
The first three (r, w, x) are clear. Use them to set read, write, and execute permissions.

s permission
The s permission is used on directories to keep the user or group ID for a file created in the directory. To set the user ID for any new files created in the directory to the owner of the directory, use the chmod u+s <directory> command. To set the group ID for any new files created in the directory to the directory's group, use the chmod g+s <directory> command.

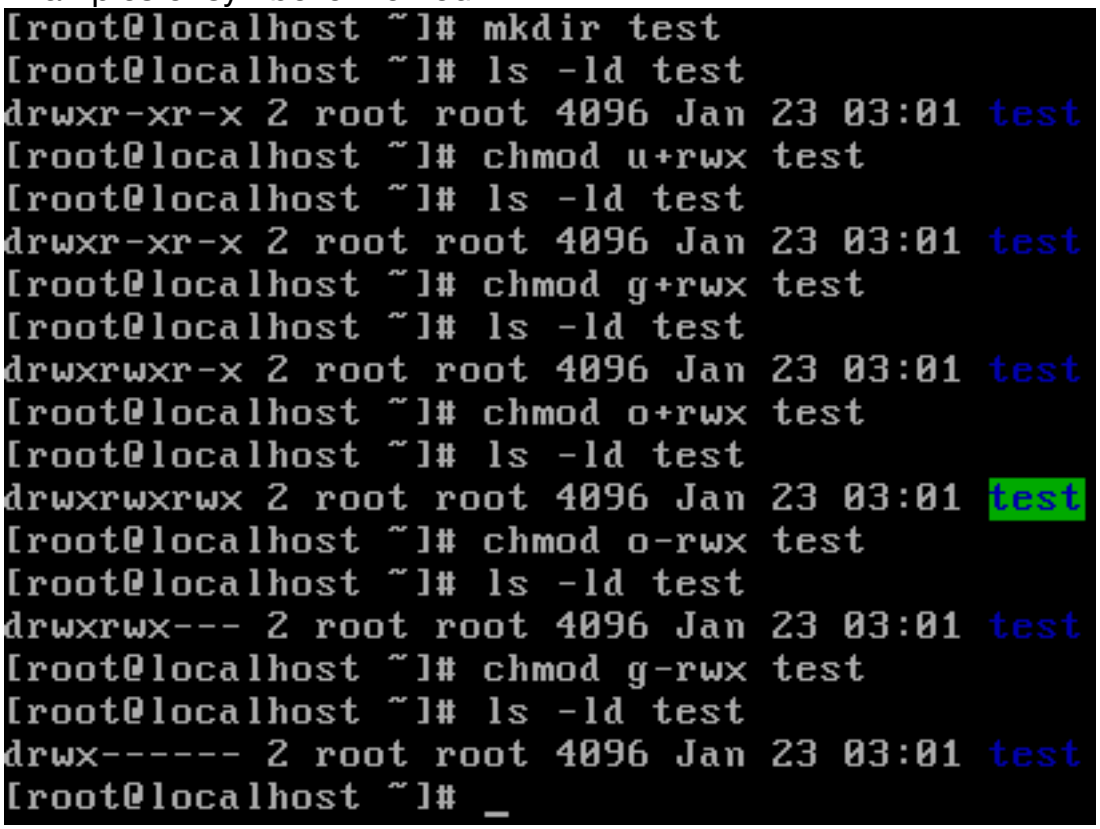
t permission
t is a special permission which provides greater security on directories. Sticky bit is used for directories to protect files within them. Files in a directory with the sticky bit set can only be deleted or renamed by the root user or the owner of the directory.

Sticky Bit Permission Using Symbols
The sticky bit permission symbol is t. The sticky bit shows up as a t in the execute position of the other permissions. A program with read and execute permissions with the sticky bit has its permissions displayed as r-t.

#chmod +t /home/vinita/account_detail

#ls -l /home/vinita/account_detail
-rwxr-xr-t 1 root root 4096 /home/vinita/account_detail

u g o permission
The last three permissions (u, g, o) are only used with the = operator to set permissions for the owner, group, others, or everyone equal to the existing permissions for the owner, group, others, or everyone. For example, chmod g=u [filename] sets the group permissions to the current permissions for the owner of the file.
Examples of symbolic method



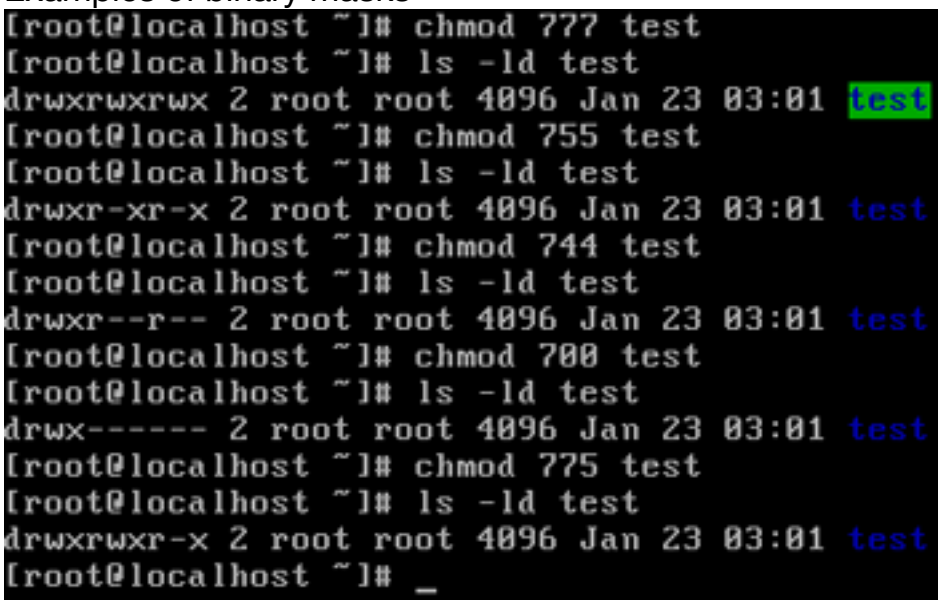
Absolute Permissions: Binary Masks
The absolute method changes all the permissions at once, instead of specifying one or the other. It uses a binary mask that references all the permissions in each category.

Binary Masks
When dealing with a binary mask, you need to specify three digits for all three categories, as well as their permissions. This makes a binary mask less flexible than the permission symbols.

Digits	permission
0	none
1	execute
2	write
4	read
3 (1+2)	write and execute
5 (1+4)	read and execute

7 (1+2+4) read write execute

777
(rwxrwxrwx) No restrictions on permissions. Anybody may do anything. Generally not a desirable setting.
755
(rwxr-xr-x) The file's owner may read, write, and execute the file. All others may read and execute the file. This setting is common for programs that are used by all users.
700
(rwx——) The file's owner may read, write, and execute the file. Nobody else has any rights. This setting is useful for programs that only the owner may use and must be kept private from others.
666
(rw-rw-rw-) All users may read and write the file.
644
(rw-r–r–) The owner may read and write a file, while all others may only read the file. A common setting for data files that everybody may read, but only the owner may change.
600
(rw——-) The owner may read and write a file. All others have no rights. A common setting for data files that the owner wants to keep private.
Examples of binary masks



Permission	Owner	Group	Other	
Read	X	x	x	777
Write	x	x	x	
Execute	x	x	x	
Permission	**Owner**	**Group**	**Other**	
Read	X	x	x	755
Write	x			
Execute	x	x	x	
Permission	**Owner**	**Group**	**Other**	
Read	X	x	x	744
Write	x			
Execute	x			
Permission	**Owner**	**Group**	**Other**	
Read	X			700
Write	x			
Execute	x			
Permission	**Owner**	**Group**	**Other**	
Read	X	x	x	775
Write	x	x		
Execute	x	x	x	

Permission	Owner	Group	Other
Read - 4	X	x	
Write - 2	x	x	x
Execute - 1	x	x	x
	4+2+1=	4+2+1=	2+1=
	7	7	3

Defaults Permission : umask

Whenever you create a file or directory, it is given default permissions. You can display the current defaults or change them with the **umask** command.

The permissions are displayed in binary or symbolic format. Execute permission for a file is turned off by default when you create it because standard data files do not use the executable permissions.

The -S option uses the symbolic format.

- #umask -S u=rwx,g=rx,o=rx

•
This default umask provides rw-r--r-- permission for standard files and adds execute permission for directories, rwxr-xr-x.
You can set a new default by specifying permissions in either symbolic or binary format. To specify the new permissions, use the -S option. The following example denies others read permission, while allowing user and group read access, which results in permissions of rwxr-x---:
• #umask -S u=rwx,g=rx,o=
•
When you use the binary format, the mask is the inverse of the permissions you want to set. To set both the read and execute permission on and the write permission off, you use the octal number 2, a binary 010. To set all permissions on, you use an octal 0, a binary 000.
The following example shows the mask for the permission defaults rwx, rx, and rx (rw, r, and r for files):
• #umask 0022
•
To set the default to only deny all permissions for others, you use 0027, using the binary mask 0111 for the other permissions.
• #umask 0027

Version Control System (VCS)

Version control systems, are specialised software whose primary goal is to manage changes to codebases over time, a process called version control.
The main aim of such a system is to be able to recall specific versions later.

What is Git?

Git is a version control system used for tracking changes in computer files. Git can handle projects of any size.
Git is used to coordinate the workflow among project team members and track their progress over time.
Git allows multiple users to work together without disrupting each other's work.

What is GitHub?

GitHub is a Git repository hosting service that provides a web-based graphical interface. It is the world's largest coding community. Programmers can find source codes in many different languages and use the command-line interface, Git, to make and keep track of any changes.

What are GitHub's Features

1. Easy Project Management

GitHub is a place where project managers and developers come together to coordinate, track, and update their work.

2. Increased Safety With Packages

Packages can be published privately, within the team, or publicly to the open-source community.

3. Effective Team Management
GitHub helps all the team members stay on the same page and organized.

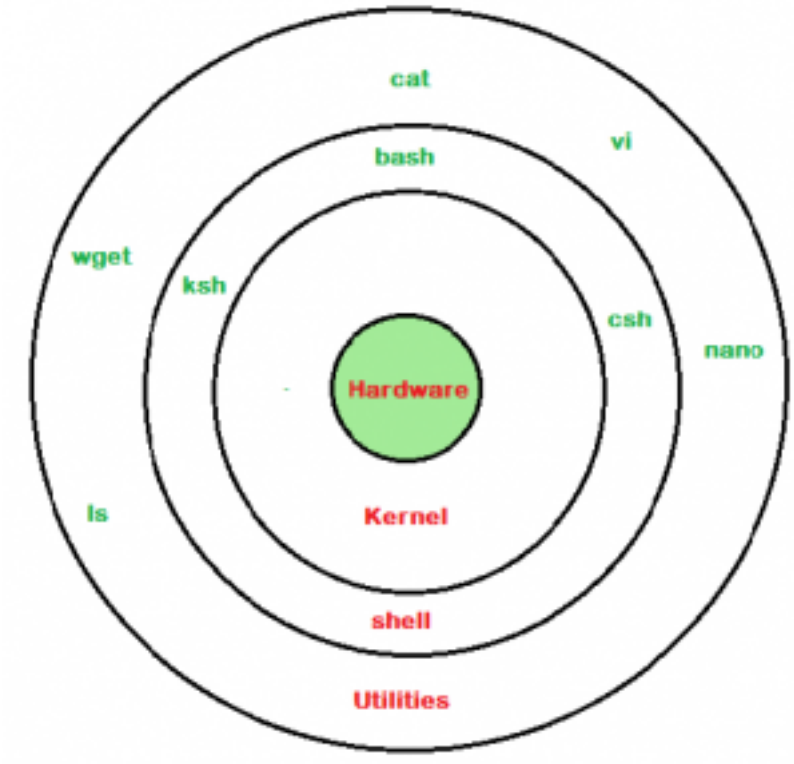
4. Improved Code Writing
Pull requests help the organizations to review, develop, and propose new code.

Important links for Git

- Setting up a repository | Atlassian Git Tutorial
- Git - Getting a Git Repository

Dialog boxes: https://www.geeksforgeeks.org/creating-dialog-boxes-with-the-dialog-tool-in-linux/

What is Shell?



The shell can easily be defined as the software program which acts as a communication bridge between kernel and user.

When the user gives the commands, the shell reads the commands, understands them and then sends a request to execute the program. Then when the program is executed it again sends the request to display the program to the user on-screen. The shell can also be called a command interpreter.
Shell is broadly classified into two categories:

- Command Line Shell
- Graphical shell

Shell Environment

Shell variables are of two types—

- local
- environment

PATH, HOME, and SHELL are **environment variables**. They are so called because they are available to the User.
By convention, environment variable names are defined in uppercase.
env is an external command and runs in a child process. It thus lists only those variables that it has inherited from its parent, the shell.
But **set** is a shell builtin and shows all variables visible in the current shell.

Common Environment Variables

HOME	Home directory—the directory in which a user is placed on logging in
PATH	List of directories searched by shell to locate a command
LOGNAME	Login name of user
USER	As above
MAIL	Absolute pathname of user's mailbox
MAILCHECK	Mail checking interval for incoming mail
MAILPATH	List of mailboxes checked by shell for arrival of mail
TERM	Type of terminal
PWD	Absolute pathname of current directory
CDPATH	List of directories searched by **cd** when used with a non-absolute pathname
PS1	Primary prompt string
PS2	Secondary prompt string
SHELL	User's login shell and one invoked by programs having shell escapes

———————————————————————————

How do I create a sub shell in Unix?
Whenever you run a shell script, it creates a new process called subshell and your script will get

executed using a subshell.
A Subshell can be used to do parallel processing.
https://linuxhandbook.com/subshell/
Just put the commands to be run in the sub-shell inside parentheses.
This causes bash to start the commands as a separate process.

OpenSSH :

It is the connectivity tool for remote login with the SSH protocol.
It encrypts all traffic to eliminate connection hijacking, and other attacks.
In addition, OpenSSH provides a large suite of secure tunneling capabilities, several authentication
methods, and sophisticated configuration options.

expr command

The expr command in Unix evaluates a given expression and displays its corresponding output.
It is used for:
-> Basic operations like addition, subtraction, multiplication, division, and modulus on integers.
-> Evaluating regular expressions, string operations like substring, length of strings etc.
$ c = `expr $a + $b`

Set Command

It is used to set or unset specific flags and settings inside the shell environment. (determines the behavior of the script and helps in executing the tasks without any issue)
It can be used to change or display the shell attributes and parameters.
https://www.geeksforgeeks.org/shell-scripting-set-command/

bc command

bc command is used for command line calculator.
It is similar to basic calculator
by using which we can do basic mathematical calculations.
bc command and expr command for doing arithmetic calculations.
https://www.geeksforgeeks.org/bc-command-linux-examples/

test command

The test command compares one element against another and returns true or false.
The test command takes an EXPRESSION as an argument.
After calculating the EXPRESSION, the test returns a value to the bash variable "$?".
If the value is 0, then the expression evaluation was true.
If the value is 1, then the expression evaluation was false.
https://linuxhint.com/bash-test-command/

Linux Utility commands

Linux utility commands

From sources across the web

- cat
- mkdir
- pwd
- chmod
- rmdir
- chown
- head
- touch
- tail
- chgrp
- find
- file
- grep
- uname
- comm
- cut

LSS (CIE 2)

Version Control System (VCS)

Version control systems, are specialised software whose primary goal is to manage changes to codebases over time, a process called version control.
The main aim of such a system is to be able to recall specific versions later.

What is Git?

Git is a version control system used for tracking changes in computer files. Git can handle projects of any size.
Git is used to coordinate the workflow among project team members and track their progress over time.
Git allows multiple users to work together without disrupting each other's work.

What is GitHub?

GitHub is a Git repository hosting service that provides a web-based graphical interface. It is the world's largest coding community. Programmers can find source codes in many different languages and use the command-line interface, Git, to make and keep track of any changes.

What are GitHub's Features

1. Easy Project Management
GitHub is a place where project managers and developers come together to coordinate, track, and update their work.

2. Increased Safety With Packages
Packages can be published privately, within the team, or publicly to the open-source community.

3. Effective Team Management
GitHub helps all the team members stay on the same page and organised.

4. Improved Code Writing
Pull requests help the organisations to review, develop, and propose new code.

Important links for Git

- Setting up a repository | Atlassian Git Tutorial
- Git - Getting a Git Repository

SetUp git

- ssh-keygen -t ed25519 -C kaustav@gmail.com

ed25519 is a public-key signature system with several attractive features.

- ls -al
- Cd .ssh
- eval "$(ssh-agent -s)"
- ~/.ssh$ sudo nano id_ed25519.pub
- ssh-add ~/.ssh/id_ed25519.pub

@@@@@@@@@@@@@@@@@@@@@@@@@@@
WARNING: UNPROTECTED PRIVATE KEY FILE!
@@@@@@@@@@@@@@@@@@@@@@@@@@@

- cd ~
- ssh -T git@github.com

Hi Kaustav! You've successfully authenticated, but GitHub does not provide shell access.

Clone/ Download

- git clone git@github.com: Kaustav1999paul/demo.git

Upload to GitHub

- git init
- git add .
- git commit -am "Hello"
- git remote add origin git@github.com:Kaustav1999paul/Test.git
- git push origin

Read

$ echo "what is your name..?";read name;echo "hello $name"

Variables

https://www.tutorialspoint.com/unix/unix-using-variables.htm

WildCard | Pattern Matching

https://geek-university.com/wildcard/
https://www.linuxjournal.com/content/pattern-matching-bash

- **?** – matches a single character. For example, **O??d** matches anything that begins with **O**, ends with **d** and has two characters in between (like **Oind**, **Okhd**, **Oerd**, but not **Oereed**, **Oad**, **Oerererd**.)

- ***** – matches any character or set of characters, including no character. For example, **O*d** matches anything that begins with **O** and ends with **d** (like **Oind**, **Okhd**, **Oerd**, **Oereed**, **Oad**, **Oerererd**, **Od**, **Oarmeerrd**). The number of characters in between **O** and **d** is not important.
- **Bracketed values** – match characters enclosed in square brackets. For example, **O[ac]d** matches only **Oad** and **Ocd**. You can also specify a range of values: **O[a-e]d** matches **Oad**, **Obd**, **Ocd**, **Odd** and **Oed**.

Pattern	Description
*	Match zero or more characters
?	Match any single character
[...]	Match any of the characters in a set
?(patterns)	Match zero or one occurrences of the patterns (extglob)
*(patterns)	Match zero or more occurrences of the patterns (extglob)
+(patterns)	Match one or more occurrences of the patterns (extglob)
@(patterns)	Match one occurrence of the patterns (extglob)
!(patterns)	Match anything that doesn't match one of the patterns (extglob)

```
 $ ls *.pdf
ee.pdf  e.pdf  .pdf

$ ls ?(e).pdf    # zero or one "e" allowed
e.pdf  .pdf

$ ls *(e).pdf    # zero or more "e"s allowed
ee.pdf  e.pdf  .pdf

$ ls +(e).pdf    # one or more "e"s allowed
ee.pdf  e.pdf

$ ls @(e).pdf    # only one e allowed
e.pdf
```

Escape Characters (\ )

https://www.shellscript.sh/escape.html

Logical Operator (-a && | -o ||)

https://dyclassroom.com/unix/shell-programming-logical-operators
https://tecadmin.net/use-logical-or-and-in-shell-script/

```
if [ `expr $a % 2` != 0 -o $a -lt 10 ]
then
  echo "$a is either odd or less than 10."
else
  echo "$a failed the test."
fi
```

Conditional Operators

https://linuxhint.com/bash_conditional_statement/

Function | Scope of Variable | parameters

https://linuxize.com/post/bash-functions/
https://www.tutorialspoint.com/unix/unix-shell-functions.htm

User Management

https://www.redhat.com/sysadmin/linux-commands-manage-users

- Adding group
- Add user
- Modifying and removing user

Process Management

https://www.journaldev.com/43930/process-management-in-linux
https://www.tutorialspoint.com/unix/unix-processes.htm
https://www.geeksforgeeks.org/process-management-in-linux/

UNIT 3

Filters (head, tail, sort, cat, uniq)

https://www.geeksforgeeks.org/filters-in-linux/#

Paste

https://www.geeksforgeeks.org/paste-command-in-linux-with-examples/

Let us consider three files having name **state**, **capital** and **number**
$ paste number state capital **=>** writes corresponding lines from the files with tab as a deliminator on the terminal.

The delimiter can be changed to any other character by using the **-d** option.
$ paste -d "|" number state capital
1|Arunachal Pradesh|Itanagar
2|Assam|Dispur
3|Andhra Pradesh|Hyderabad

It reads all the lines from a single file and merges all these lines into a single line with each line separated by tab. And these single lines are separated by newline.
$ paste -s number state capital
1 2 3 4 5
Arunachal Pradesh Assam Andhra Pradesh Bihar Chhattisgrah
Itanagar Dispur Hyderabad Patna Raipur

pr

https://www.geeksforgeeks.org/pr-command-in-linux/
pr -3 abc.txt -> print this content in 3 columns we will use the following command.
pr -t abc.txt -> suppress the headers and footers the -t option is used.
pr -n abc.txt -> To provide line numbers.

tr

https://www.geeksforgeeks.org/tr-command-in-unix-linux-with-examples/
$cat greekfile | tr “[a-z]” “[A-Z]” => convert from lower case to upper case
$ echo "Welcome To GeeksforGeeks" | tr [:space:] '\t' => translate all the white-space to tabs
$ tr '{}' '()' newfile.txt => translate braces in a file with parenthesis.
$ echo "Welcome To GeeksforGeeks" | tr -s [:space:] ' ' => convert multiple continuous spaces with a single space
$ echo "Welcome To GeeksforGeeks" | tr -d 'w' => deletes characters in the first set specified.

SED & AWK

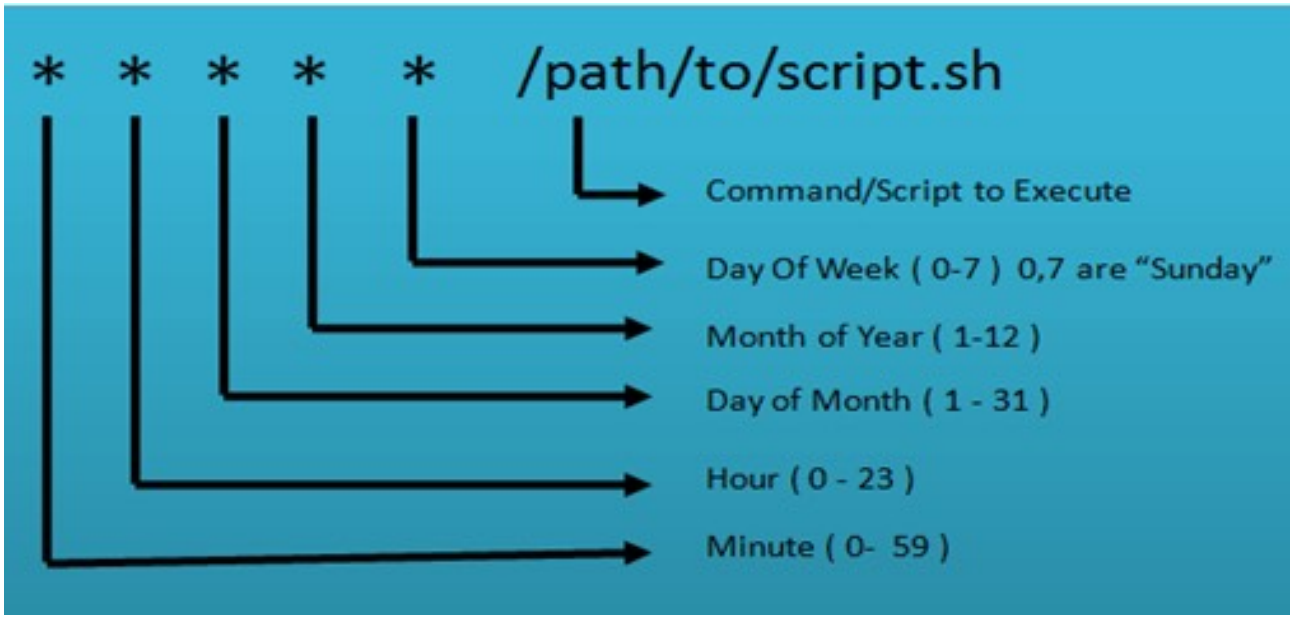
https://www.tim-dennis.com/data/tech/2016/08/09/using-awk-filter-rows.html
(Notes)

Grep

(Notes)

At

https://www.geeksforgeeks.org/at-command-in-linux-with-examples/amp/



Description	Command
Cron command to do the various scheduling jobs. Below given command execute at 7 AM and 5 PM daily.	07,17 * * * /scripts/script.sh
Command to execute a cron after every 5 minutes.	*/5* * * * * /scripts/script.sh
Cron scheduler command helps you to execute the task on every Monday at 5 AM. This command is helpful for doing weekly tasks like system clean-up.	0 5 * * mon /scripts/script.sh
Command run your script on 3 minutes interval.	*/3 * * * * /scripts/monitor.sh
Command to schedule a cron to which executes for a specific month. This command to run tasks run in Feb, June and September months. Sometimes we need to schedule a task to execute a select monthly task.	* * * feb,jun,sep * /script/script.sh
Command to execute on selected days. This example will run each Monday and Wednesday at 5 PM.	0 17 * * mon,wed /script/script.sh
This command allows cron to execute on first Saturday of every month.	0 2 * * sat [$(date +%d) -le 06] && / script/script.sh
Command to run a script for 6 hours interval so it can be configured like below.	0 */6 * * * /scripts/script.sh
This command schedule a task to execute twice on Monday and Tuesday. Use the following settings to do it.	0 4,17 * * mon,tue /scripts/script.sh

Command schedule a cron to execute after every 15 Seconds.	* * * * * /scripts/script.sh * * * * * sleep 15; /scripts/script.sh
Command to schedule tasks on a yearly basis. @yearly timestamp is= to “0 0 5 1 *”. This executes the task in the fifth minute of every year. You can use it to send for new year greetings.	@yearly /scripts/script.sh
Command tasks to execute on a monthly basis. @monthly timestamp is similar to “0 0 1 * *”. This command expression allows the execution of a task in the first minute of the month.	@monthly /scripts/script.sh
Command to execute multiple tasks using a single cron.	* * * * * /scripts/script.sh; /scripts/scrit2.sh
Command to schedule tasks to execute on a weekly basis. @weekly timestamp is similar to “0 0 4 * sun”. This is used to perform the weekly tasks like the system cleanup	@weekly /bin/script.sh
Task will be scheduled to execute on a daily basis. @daily timestamp is similar to “0 2 * * *”. It executes the task in the second minute of every day.	@daily /scripts/script.sh
Allows tasks to execute on an hourly. @hourly timestamp is similar to “0 * * * *”. This command executes a task in the first minute of every hour.	@hourly /scripts/script.sh
Allows tasks to execute on system reboot. @reboot expression is useful for those tasks that the system wants to run on your system startup. This is helpful to begin tasks background automatically.	@reboot /scripts/script.s