**Assignments**:-

**1.List the contents of a directory and their attributes.**

Answer: ls or -d

**2.Long list the content with file, directory ownership, permissions,sizes, etc…**

Answer: ls -l

**Display the size of the file in human readable format**

Answer: ls -s

**Show all files and folders including hidden one**

ls -a

**list directories recursively**

ls -R

**Sort the files by size with largest at the top**

ls -S

**Sort the files by last time modified displaying the newest first.**

ls -ltr

**Diplay the location of a program/command, where it is installed.**

whereis

**Which command is used to switch directory from one to another**

cd

**List all the environment variables set for the current shell environment**

env

**Did you notice something in the output of "env" command?**

It can print a list of current environment variables,or to run another program in a custom environment

without modifying the current one.

**Which command is used to print the text or any variables value in the Console/Terminal?**

echo

**Print the value of the env variable "PATH" on the console**

echo $env:PATH

**Is linux a case-sensitive operating system?**

Yes.

**Is, "Ls" same as "ls"?**

No.

**Display your currently logged in user**

w command

**how do you change the currrently logged in user to another user?**

su command (switch user)

**Which command is used to leave a shell environment that you are currently logged in to?**

exit

**How do you reboot the system?**

To reboot the Linux system from a terminal session, sign in or “su”/”sudo” to the “root” account.

Then type “ sudo reboot ” to reboot the box. Wait for some time and the Linux server will reboot itself

**How do you shutdown the system?**

log in as root and type one of the commands poweroff, halt or shutdown -h

**Display all the major running processes in the system**

ps aux

sudo ps -a

**Understand the output of command used in above question of displaying processes, explain the** **meaning of each column and what data it displays?**

It describes column by column CPU time used by the processes. Physical memory used by this process, Virtual memory used by this process and

terminal from which this process started.

**Display the name of the system kernel**

uname -s

**display the kernel release number**

uname -r

**display teh machine type of the current kernel**

uname -m

**Display the name of the operating system that the kernel is running on**

uname -o

**Display all info that uname command can show.**

uname -a

**Display the name of directory that you are currently pointing to**

pwd

**change the current directory to another directory that you have in your system.**

cd photos

**Go up one directory**

cd ..

**Return to last directory**

cd-

**change the current directory to home(logged in user's) directory**

cd command

**How to check all the command used from the prompt (Command History)**

history

**In which file the history of commands are stored in?**

history file

**How many lines of history does the system keep and from where you can change it?**

By default 1000 lines of history will be stored as per the values stored in $HISTSIZE and $HISTFILESIZE

The default for all users is stored in /etc/profile file,this can be modified.

**How can you modify bash's history behaviour**

PROMPT\_COMMAND

**Display all the commands entered so far, now, try to run a particular command from the history list** **without typing that command.**

history

Control+R and type the keyword

NOTE:- TAB key is your friend when it comes to command completion and having long file and directory names autocompleted at the bash prompt for you. JUST BE LAZY AND USE TAB KEY FOR AUTO COMPLETION ;-)

**What are the different types of shell and where are they used and how do we use them?**

Each shells has properties that make them highly efficient for a specific type of use over other shells. So Following are the different types of shells in Linux along with their properties and features.

1. The Bourne Shell (sh)

Developed at AT&T Bell Labs by Steve Bourne, the Bourne shell is regarded as the first UNIX shell ever. It is denoted as sh. It gained popularity due to its compact nature and high speeds of operation.

This is what made it the default shell for Solaris OS. It is also used as the default shell for all Solaris system administration scripts. Start reading about shell scripting here.

However, the Bourne shell has some major drawbacks.

It doesn’t have in-built functionality to handle logical and arithmetic operations.

Also, unlike most different types of shells in Linux, the Bourne shell cannot recall previously used commands.

It also lacks comprehensive features to offer a proper interactive use.

The complete path-name for the Bourne shell is /bin/sh and /sbin/sh. By default, it uses the prompt # for the root user and $ for the non-root users.

2. The GNU Bourne-Again Shell (bash)

More popularly known as the Bash shell, the GNU Bourne-Again shell was designed to be compatible with the Bourne shell. It incorporates useful features from different types of shells in Linux such as Korn shell and C shell.

It allows us to automatically recall previously used commands and edit them with help of arrow keys, unlike the Bourne shell.

The complete path-name for the GNU Bourne-Again shell is /bin/bash. By default, it uses the prompt bash-VersionNumber# for the root user and bash-VersionNumber$ for the non-root users.

3. The C Shell (csh)

The C shell was created at the University of California by Bill Joy. It is denoted as csh. It was developed to include useful programming features like in-built support for arithmetic operations and a syntax similar to the C programming language.

Further, it incorporated command history which was missing in different types of shells in Linux like the Bourne shell. Another prominent feature of a C shell is “aliases”.

The complete path-name for the C shell is /bin/csh. By default, it uses the prompt hostname# for the root user and hostname% for the non-root users.

4. The Korn Shell (ksh)

The Korn shell was developed at AT&T Bell Labs by David Korn, to improve the Bourne shell. It is denoted as ksh. The Korn shell is essentially a superset of the Bourne shell.

Besides supporting everything that would be supported by the Bourne shell, it provides users with new functionalities. It allows in-built support for arithmetic operations while offereing interactive features which are similar to the C shell.

The Korn shell runs scripts made for the Bourne shell, while offering string, array and function manipulation similar to the C programming language. It also supports scripts which were written for the C shell. Further, it is faster than most different types of shells in Linux, including the C shell.

The complete path-name for the Korn shell is /bin/ksh. By default, it uses the prompt # for the root user and $ for the non-root users.

5. The Z Shell (zsh)

The Z Shell or zsh is a sh shell extension with tons of improvements for customization. If you want a modern shell that has all the features a much more, the zsh shell is what you’re looking for.

Some noteworthy features of the z shell include:

Generate filenames based on given conditions

Plugins and theming support

Index of built-in functions

Command completion

**What is the difference between login shell and non-login shell?**

**What happens when you start a login shell (which files are read and used and Why)?**

**What happens when you start a non-login shell (Which files are read and used and Why)?**

) Login shells

A Login shell is started after a successful login, using /bin/login, by reading the /etc/passwd file. Login shell is the first process that executes under our user ID when we log in to a session. The login process tells the shell to behave as a login shell with a convention: passing argument 0, which is normally the name of the shell executable, with a “-” character prepended. For example, for Bash shell it will be -bash.

When Bash is invoked as a Login shell;

→Login process calls /etc/profile

→/etc/profile calls the scripts in /etc/profile.d/

→Login process calls ~/.bash\_profile

→~/.bash\_profile calls ~/.bashrc

→~/.bashrc calls /etc/bashrc

Login shells include the following.

• Shells created by explicitly telling to login.

examples: # su - | # su -l | # su --login | # su USERNAME - | # su -l USERNAME | # su --login USERNAME | # sudo -i

• Shells created at login, including X login.

A Login shell can be recognized by the following procedure.

Execute the below command in shell.

# echo $0

If the output is the name of our shell, prepended by a dash, then it is a login shell.

For example -bash, -su etc.

2) Non login shells

A Non login shell is started by a program without a login. In this case, the program just passes the name of the shell executable. For example, for a Bash shell it will be simply bash.

When bash is invoked as a Non login shell;

→Non-login process(shell) calls ~/.bashrc

→~/.bashrc calls /etc/bashrc

→/etc/bashrc calls the scripts in /etc/profile.d/

Non login shells include the following.

• Shells created using the below command syntax.

examples: # su | # su USERNAME

• Graphical terminals

• Executed scripts

• Any other bash instances

A Non login shell can be recognized by the following procedure.

Execute the below command in shell.

# echo $0

If the output is the name of our shell, does not prepend by a dash, then it is a Non login shell.

For example bash, su etc.

**How do we start login shell and non-login shell?**

You can allow Terminal to start a login shell. Your default shell will be started with a dash character prepended to its name.

Press the menu button in the top-right corner of the window and select Preferences.

In the sidebar, select your current profile in the Profiles section.

Select Command.

Under the Command label, select Run command as a login shell.

→Non-login process(shell) calls ~/.bashrc

→~/.bashrc calls /etc/bashrc

→/etc/bashrc calls the scripts in /etc/profile.d/

**What are Shell Configuration Files, why do we need it?**

Shell configuration files are executed automatically when you log in and

out of a shell. They initialize and configure a shell upon login and

perform cleanup operations upon logout.

BASH

BASH defines three configurations files: .bash\_profile, .bashrc, and

.bash\_logout. .bash\_profile contains initialization commands that set

environment variables, a shell's prompt and so on. The .bashrc contains

commands that configure the shell, define command aliases and set

command shell options. .bash\_profile is executed automatically when a

user logs into a shell, along with .bashrc. The .bash\_logout file is

executed when the user logs out of a shell. .bash\_logout contains

cleanup operations and other commands that you want the shell to execute

whenever a user logs out of a shell. For example, it can include

commands that clear the screen and print a farewell message. Unlike the

.bash\_profile file, which is created automatically when you open a new

account, you have to create the .bash\_logout by yourself using a text

editor such as vi, emacs, etc.

In the following example, the .bash\_logout file contains commands that

clear the screen and print a reminder to the user to take her diskette

from the disk drive:

clear

echo "don't forget to remove your homework exercises diskette from the

diskette drive!"

Other Shells

Other shells have similar configuration files, albeit with different

names. Thus, instead of the BASH .bash\_profile, .bashrc, and

.bash\_logout configuration files, TCSH uses the .login, .tcshrc and

.logout configuration files, respectively. Likewise, Z-shell uses the

files .zshenv, .zprofile, and .zlogin for initialization, .zshrc as a

configuration file equivalent to .bashrc, and .zlogout as the equivalent

of .bash\_logout. The PDKSH shell uses the .profile and .kshrc files for

initialization and shell configuration, respectively.

**Explain the Order of file usage from the system/user's home directory when user logs in to the System.**

$ **cd**

$ **pwd**

/export/home/*username*

$ **touch tempfile (Creating a test file)**

$ **ls tempfile**

tempfile (Listing files)

$ **cp tempfile copyfile** (Copying files)

$ **ls \*file**

copyfile tempfile (Copying files)

$ **mv tempfile emptyfile** (Moving and renaming file)

$ **ls \*file**

copyfile emptyfile

$ **rm copyfile**

$ **ls \*file**

Emptyfile (Deleting files)

$ **cat file1 file2 file3 > bigfile**

$ **ls \*file**

bigfile

file1

file2

file3

$ (Displaying file contents)

$ **file copyfile**

copyfile: ascii text (Displaying file type)

**What are Shell Variables, list major shell variables and what do they represent?**

A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables.

Major shell variables:  
Local,environment and shell variables.

**How we see all our env variables?**

$printenv

**How we see all env variables in alphabical order?**

Get-children env:\* |

Sort-object Key |

**What Format does the env var and its values are stored?**

each environment variable's default value is stored in the Windows registry or set in the AUTOEXEC.BAT file.

**How do you create your own varible?**

A variable is defined by simply assigning a value to a name using the '=' operator. Avariable name is a series of alphanumeric characters starting with a letter or '\_'. Variables are all treated as text strings unless the context requires them to be treated as a numeric value.

**How do you start a new bash shell?**

konsole

**Difference between Local/Shell variables to Global Variable**

shell variable is local to a particular instance of the shell (such as a shell script), while environment variables are "inherited" by any program you start, including another shell

**Making a variable accessible from other the shell in the system.**

f you want to make avariable available to every new process, then you can put it in /etc/environment (requires root privileges) or ~/. profile