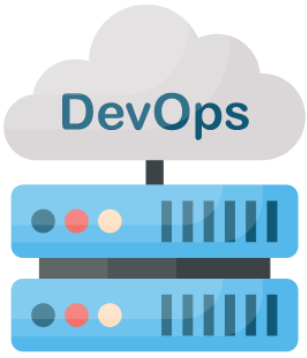
iNeuron

DevOps
Interview Questions

PART -1: Linux Basics



Question 01: Shells

What is a shell?

A: A shell can best be described as a language to communicate with a computer. Most users operate with the point-and-click language of the desktop, but the computer is leading the conversation in this kind of language. The user has the passive role of picking tasks from the choice presented by the computer. It is very difficult for a programmer to include all options and possible uses of a command in the Graphical User Interface (GUI) format. Thus, GUIs are almost always less capable than the command or commands that form the backend.

Shell is an advanced way of communicating with the system because it allows for two-way conversation. Interactions in the communication

are equal so that new ideas can be tested. The shell allows the user to handle a system in a very flexible way and allows for task automation.

When you mess around in the Linux command line, there's always a program called shell running. When you type commands, it's the shell that reads the commands from your keyboard, processes them and finally gives them to the operating system. There are many different shell programs available but on most Linux systems, bash (Bourne Again Shell) is the default shell. When you start a shell on your Linux system, it's most likely bash.

Question 02: Virtual Terminal and Terminal Emulator

I’m in the X Window System (the GUI) right now. How do I get to the Command Line Interfaces (CLI)?

A: You can start playing with the CLI by switching to another virtual terminal or you can start a terminal emulator.

A terminal emulator is a program that opens up a window and then runs a shell in that window. Start a terminal emulator like any other X program, so it means that you can have a command line while you're still safely in the GUI. Now browse the programs menu that you normally use for launching programs and look for applications that seem like a terminal emulator (xterm, rxvt, gnome- terminal, konsole, kvt, eterm, etc…). You can launch as many of them as you want and try all of them. They all

basically do the same thing; it lets you use the shell. The different terminal emulators run the same shell program bash, in each of them.

Another way of getting to the CLI is to switch to another virtual terminal while the X Window System is still running. By default, Linux usually has six virtual terminals and the seventh one is where X is running. You can switch the virtual terminals by pressing Ctrl+Alt and the function key with the number of the desired terminal. Pressing Ctrl+Alt+F1 takes you to the first virtual terminal, Ctrl+Alt+F2 takes you to the second virtual terminal and so on.

When you're in the virtual terminals, you're not in the GUI (X Window System) anymore but in the CLI. The GUI is still running in the seventh virtual terminal so you can just switch back there normally with Ctrl+Alt+F7.

When the terminal displays the login: prompt, you can now type in your user name and log in

normally in the terminal. Maybe doing things in X with a terminal emulator would be easier at first but it doesn't matter because the result is the same: you have a bash prompt in front of you. The prompt will wait for you to type a command or a name of a program. For example, try typing “ls”. As you can see from the output, the command “ls” list the contents of your current

directory. All the commands in Linux are little programs themselves and you can run the programs by typing their name.

Question 03: Linux directory structure

How is the Linux file system structured?

A: The following are some list of directories that will show you the most interesting places in your file system.

< / >: The root directory is the starting point of your directory structure and it is where the Linux system begins. Every other file and directory on your system is under the root directory. Usually the root directory contains only subdirectories and it is not advisable to store single files directly under root.

< /boot >: This is where Linux keeps information that it needs when booting up. If you list the contents of /boot, you'll see a file called vmlinuz - that's the kernel.

< /etc >: This is the configuration files for the

Linux system. Most of these files are text files and can be edited manually. Here are some interesting contents of this directory:

*etc*inittab: A text file that describes what processes are started at system bootup and during normal operation. Here you can determine if you want the X Window System to start automatically at bootup and configure what happens when a user presses Ctrl+Alt+Del.

*etc*fstab: This file contains descriptive information about the various file systems and their mount points, like floppies, cdroms, *etc*.

*etc*passwd: A file that contains various pieces of information for each user account. This is where the users are defined.

< *bin,* usr/bin >: These two directories contain a lot of programs for the system. The *bin directory contains the most important programs that the*

system needs to operate, such as the shells, ls, grep, and other essential things. usr/bin in turn contains applications for the system's users.

< *sbin*, usr/sbin >: Most system administration programs are stored in these directories. In many cases you must run these programs as the root user.

< usr >: This directory contains user applications and a variety of other things for them, like their source codes, pictures, docs, or config files they use. usr is the largest directory on a Linux system and some people like to have it on a separate partition. Here are some contents of the /usr:

*usr*doc: Documentation for the user apps in many files formats.
*usr*share: Config files and graphics for many user apps.
*usr*src: Source code files for the system's software including the Linux kernel. *usr*include: Header files for the C compiler. The header files define structures and constants that are needed for building most

standard programs. A subdirectory under *usr*include contains headers for the C++ compiler.

*usr*X11R6: The subdirectories under *usr*X11R6 may contain some X binaries themselves as well as documentation, header files, config files, icons, sounds and things related to the graphical programs.

< *usr*local >: This is where you install apps and other files for use on the local machine. If you find interesting apps that aren't officially a part of your distro, you should install them in *usr*local. For example, if the app would normally go to *usr*bin but it isn't a part of your distro, you should install it in *usr*local/bin instead. Keep your own programs away from the programs that are included in your distro, to confusion and keep things in order.

< /lib >: The shared libraries for programs that are dynamically linked.

< /home >: This is where users keep their personal files. Every user has their own directory under /home, and usually it's the only place where normal users can write files. You can configure a Linux system so that normal users can't even list the contents of other users' home directories.

< /root >: The superuser's (root's) home directory. You should not confuse this with the root directory (/) of a Linux system.

< /var >: This directory contains variable data that changes constantly when the system is running. Some interesting subdirectories:

*var*log: A directory that contains system log files. These are updated when the system runs and if something in your system suddenly goes wrong, the log files may contain some info about the situation.
*var*mail: Incoming and outgoing mail is stored in this directory.
*var*spool: This directory holds files that are queued for some process like printing.

< /tmp >: Programs can write their temporary files here.

< /dev >: The devices that are available to a Linux system. In Linux, devices are treated like files and you can read and write devices like they were files.

< mnt >: This directory is used for mount points. The different physical storage devices must be attached to some directory in the file system tree before they can be accessed. This attaching is called mounting and the directory where the device is attached is called the mount point. The mnt directory contains mount points for different devices, like *mnt*floppy for the floppy drive, *mnt*cdrom for the CD-ROM and so on. However, you're not forced to use the *mnt directory for this purpose, you can use whatever directory you wish. Actually, in some distros, like Debian and SUSE, the default is to use* floppy and *cdrom as mount points instead of directories under* mnt.

< proc >: This is a special directory. Actually, proc is just a virtual directory because it doesn't exist at all. It contains some info about the kernel itself. There's a bunch of numbered entries that correspond to all processes running on the

system and there are also named entries that permit access to the current configuration of the system. Many of these entries can be viewed.

< /lost found >: This is where Linux keeps the files that it restores after a system crash or when a partition hasn't been uncounted before a system shutdown. This way you can recover files that would otherwise have been lost.

Question 04: Command-line history

What are functions of the history command?

A: The history command can be used to list Bash's logs of the commands you have typed. This log is called the "history". To access it type:

The “history n” will only list the last n commands. You can type "history" (without options) to see the entire history list.

You can also type “!n” to execute command number n. Use “!!” to execute the last command you typed.

The “!-n” will execute the command n times before (ie. !-1 is equivalent to !!).

The “!string” will execute the last command starting with that string and “!? string?” will execute the last command containing the word "string".

For example:

!cd will re-run the command that you last typed starting with "cd".

" commandName !*" will execute the "commandName" with any arguments you used on your last command. This maybe useful if you make a spelling mistake, for example. If you typed: emasc *home*fred/mywork.java *tmp*testme.java

In an attempt to execute emacs on the above two files this will obviously fail. So what you can do is type:

emacs !* will execute emacs with the arguments that you last typed on the command line. In other words this is equivalent to typing:

emacs *home*fred/mywork.java *tmp*testme.java

To search through the Command History (CTRL-R) : sse the CTRL-R key to perform a "reverse-i-search". For example, if you wanted to use the command you used the last time you used snort, you would type: CTRL-R then type "snort".

What you will see in the console window is: (reverse-i-search)`':

After you have typed what you are looking for, use the CTRL-R key combination to scroll backward through the history.

Use CTRL-R repeatedly to find every reference to the string you have entered. Once you've found the command you're looking for, use [Enter] to execute it.

Alternatively, using the right or left arrow keys will place the command on an actual command line so you can edit it.

Question 05: Linux keyboard shortcuts

In MS Windows, there are a lot of keyboard shortcuts that is used to make life easier for users.

Does Linux have this also?

A: There are several keyboard shortcuts in Linux and here are some of them. Some X Window System shortcuts are also included:

VIRTUAL TERMINALS:

Ctrl + Alt + F1: Switch to the first virtual terminal. In Linux, you can have several virtual terminals at the same time. The default is 6.

Ctrl + Alt + Fn: Switch to the nth virtual terminal. Because the number of virtual terminals is 6 by default, n = 1...6.

tty: Typing the tty command tells you what virtual terminal you're currently working in.

Ctrl + Alt + F7: Switch to the GUI. If you have X Window System running, it runs in the seventh virtual terminal by default. If X isn't running this terminal is empty.

X WINDOW SYSTEM:

Ctrl + Alt + +: Switch to the next resolution in the X Window System. This works if you've configured more than one resolution for your X server. You must use the + in your numpad.

Ctrl + Alt + -: Switch to the previous X resolution. Use the - in your numpad.

MiddleMouseButton: Paste the highlighted text. You can highlight the text with your left mouse button (or with some other highlighting method, depending on

the application you're using) and then press the middle mouse button to paste. This is the traditional way of copying and pasting in the X Window System, but may not work in some X applications.

If you have a two-button mouse, pressing both of the buttons at the same time has the same effect as pressing the middle one. If it doesn't, you must enable 3- mouse-button emulation.
This works also in text terminals if you enable the gpm service.

Ctrl + Alt + Backspace: Kill the X server. Use this if X crashes and you can't exit it normally. If you've configured your X Window System to start automatically at boot up, this restarts the server and throws you back to the graphical login screen.

COMMAND LINE INPUT:

Home or Ctrl + a: Moves the cursor to the

beginning of the current line. End or Ctrl

+ e: Moves the cursor to the end of the

current line.

Alt + b: Moves the cursor to the beginning of the current or previous word. Note that while this works in virtual terminals, it may not work in all graphical terminal emulators because many graphical applications already use this as a menu shortcut by default.

Alt + f: Moves the cursor to the end of the next word. Again, like with all shortcuts that use Alt as the modifier, this may not work in all graphical terminal emulators.

Tab: Autocomplete commands and file names. Type the first letter(s) of a command, directory or file name, press Tab and the rest is completed automatically. If there are more commands

starting with the same letters, the shell completes as much as it can and beeps. If you then press Tab again, it shows you all the alternatives.

This shortcut is really helpful and saves a lot of typing. It even works at the lilo prompt and in some X applications.

Ctrl + u: Erases the current line.

Ctrl + k: Deletes the line from the position of

the cursor to the end of the line. Ctrl + w:

Deletes the word before the cursor.

COMMAND LINE OUTPUT:

Shift + PageUp: Scroll terminal output up.

Shift + PageDown: Scroll terminal output down.

Ctrl + l: The clear command clears all previously executed commands and their output from the current terminal.

reset: If you mess up your terminal, use the reset

command. Note that you may not be able to see the command when you're typing it.

COMMAND LINE HISTORY:

history: This gives you a list of the commands you previously executed when you type this command.

ArrowUp or Ctrl + p: Scroll up in the history and edit the previously executed commands.

ArrowDown or Ctrl + n: Scroll down in the history and edit the next commands.

Ctrl + r: This shortcut finds the last command that contained the letters you're typing. For example, if you want to find out the last parameters you gave to the "cp" command, you'll press Ctrl + r and type in "cp".

MISCELLANEOUS COMMAND LINE:

Ctrl + c: Kills the current process.

Ctrl + z: This sends the current process to background. It is useful if you have a program running and you need the terminal for awhile but don't want to exit the program completely. Type the command fg to get the process back.

Ctrl + d: Log out from the current terminal. If you use this in a terminal emulator under X, this usually shuts down the terminal emulator after logging you out.

Ctrl + Alt + Del: Reboot the system. You can change this behavior by editing
*etc*inittab if you want the system to shut down instead of rebooting.

Question 06: Same path for all the users

Is there an easy procedure to get the same path for all the users?

A: The most important settings are possible to set in the global shell initialization files for login shells: *etc*csh.login for tcsh and *etc*profile for bash.

Exceptions that do not get the right path from these files are rsh commands, ssh commands, menu items from X window manager that do not explicitly start login shell, commands invoked from inittab, cron jobs, daemons jobs like magic filters started from lprd, WWW CGI scripts, and so on.

If the path is set in *etc*csh.cshrc, the path is right even when rsh or ssh execute command in remote machine with account using tcsh/csh. However, it is not possible to set path if account uses

bash/sh.

It is possible to combine the path setting to one file, for example to a file *etc*environment-common. There we write: ${EXPORT}PATH${EQ}/bin:/usr/bin:/sbin:/usr/

sbin:/usr/bin/X11:/usr/local/bin: This can be

used from *etc*csh.login (for tcsh and csh)

set EQ=" " set EXPORT="setenv "

source *etc*environment-common And

from *etc*profile (for bash, doesn't work

for ordinary sh)

EQ='=' EXPORT="export " .

*etc*environment-common And

from *etc*environment (for XDM)

EQ="=" EXPORT="export " . *etc*environment-common
This strategy works mostly but ssh will complain of the lines in *etc*environment (and defined environment variables EQ and EXPORT). And still, rsh commands executed with bash won't get this path.

Question 07: Recover root password

How can I recover the root password on RH Linux AS?

A: If you are using Lilo : Type at lilo: linux 1 and it will let you in Single User Mode.

If you are using Grub: Press e when grub menu shows. There is line which starts with kernel, then highlight that line and type e again and then put -s and press escape and press b to boot your System. This procedure will let you in Single User Mode.

Change your password by passwd command.

Question 08: Init

What is initialization?

A: The term “init” (short for "initialization") is the program on Unix and Unix- like systems which spawns all other processes. It runs as a daemon and typically has PID 1.

The functionality diverged considerably between BSD and System V. The usage on most Linux distributions is compatible with System V, but some distributions, such as Slackware, use a BSD-style and others, such as Gentoo Linux, have their own customized version.

Init is a parent process for all the other processes of the system. Other processes inherit environment of the init process and the path is the init path in the rare case that no other path is set.

The 'init path' is fixed in the source of the init

program and it is:
*usr*local/sbin:/sbin:/bin:*usr*sbin:*usr*bin

Note that init path does not contain *usr*local/bin.

All the programs that are started from *etc*inittab work in init environment, especially system initialization scripts in *etc*init.d (Debian 1.3).

Everything that is started from system initialization scripts has init environment as default environment. For example, syslogd, kerneld, pppd (when started from startup), gpm and most importantly lpd and inetd have init environment and they do not change it.

A group of programs are started from startup scripts but the PATH environment variable is explicitly set in the startup script. Examples are: atd, sendmail, apache and squid.

There are other programs that are started from

boot scripts but they change the path completely. One such example is cron.

Question 09: Change user ID

How do I change user ID?

A: Command “su” sets a new user ID to use. If no user ID is given, root is used.

Usually, su invokes a subshell with a different user ID. With argument '-' (more recent synonyms -l or --login) su invokes shell like login shell. However, it does not use login program to do this but uses another built-in path for login 'simulation' (term used in the source code).

For:

normal users: ***usr*local/bin:*usr*bin:/bin:*usr*bin/X11**:

root user: ***sbin:*bin:*usr*sbin:*usr*bin:*usr*bin/X**

11:*usr*local/sbin:*usr*local/bin

There is a group of commands that make use of super user commands safer. They allow better logging, user-based restrictions and usage of individual passwords. The most widely used is sudo:

$ sudo env

This command modifies the search path so that the current directory is always the last one but it does not modify PATH environment variable. 'sudo env' and 'env' give the same value for PATH variable. Sudo just adds a couple of environment variables like SUDO_USER.

Question 10: Head and Tail

How does head and tail function?

A: These two commands display the n first/last lines of a file respectively. To see the last ten commands entered:

```
wayne:~> tail -10 .bash_history
locate configure | grep bin
```

man bash

cd

xawt

v &

grep usable

*usr*share/dict/words

grep advisable

*usr*share/dict/words

info quota

The command head works similarly. The tail command has a handy feature to continuously show the last n lines of a file that changes all the time.

Source: From Internet

Follow Me For next series of Questions coming

https://www.linkedin.com/in/mukeshkumarrao/

Visit at: www.ineuron.ai

For Live Training courses and Job Assurance guidance from our mentors

********** THANK YOU **********