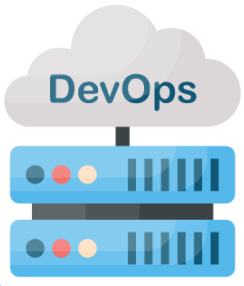
iNeuron

# DevOps

## Interview Questions

PART -2: Linux Basics



## Question 11: The ps command

How do I get the list of processes currently running?

**A:** The “ps” command will provide you a list of processes currently running. Use the “ps –ef” command and you will be able to see exactly what is running on your system and kill run-away processes or those that are causing problems.

Another use of the ps command is to see what users are currently doing. Use ps –u user (e.g. jim) will give you a sample output of:

user@server:~>
ps -u jim 20876
pts/100:00:00
bash
20904 pts/200:00:00 bash
20951 pts/200:00:00 ssh
21012 pts/100:00:00 ps

From this we can see that the

user is doing ps ssh.

COMMAND-LINE

OPTIONS:
This version of ps accepts several kinds of options.

Unix98 options may be grouped and must be preceeded by a dash. BSD options may be grouped and must not be used with a dash.
GNU long options are

preceeded by two dashes.

Options of different types may

be freely mixed.

Set the I_WANT_A_BROKEN_PS environment variable to force BSD syntax even when options are preceeded by a dash. The

PS_PERSONALITY environment variable (described below) provides more detailed control of ps behavior.

SIMPLE PROCESS SELECTION:

SwitchDescription
-Aselect all processes
-Nnegate selection
-aselect all with a tty except session leaders
-dselect all, but omit session leaders
-eselect all processes
Tselect all processes on this terminal
aselect all processes on a terminal,
including those of other users greally
all, even group leaders (does nothing
w/o SunOS settings) rrestrict output to
running processes
xselect processes without controlling ttys
--deselectnegate selection

PROCESS SELECTION BY LIST:
Switch Description
-C select by command name
-Gselect by RGID (supports names)
-Uselect by RUID (supports names)
-gselect by session leader OR by group name
-pselect by PID
-sselect processes belonging to the sessions
given

-tselect by tty
-uselect by effective user ID (supports names)
Uselect processes for
specified users pselect
by process ID
tselect by tty
--Group select by real group name or ID
--Userselect by real user name or ID
--group select by effective group name or ID
--pidselect by process ID
--sidselect by session ID
--ttyselect by terminal
--userselect by effective user name or ID
-123implied --sid

**123implied --pid**

OUTPUT FORMAT CONTROL:
Switch Description
-O is preloaded "-o"
-c different scheduler info for -l option
-f does full listing
-j jobs format
-l long format
-o user-defined format
-y do not show flags; show
rss in place of addr O is
preloaded "o" (overloaded)
X old Linux i386
register format j
job control format
l display long format
o specify user-
defined format s
display signal
format
u display user-oriented format
v display virtual memory format
--format user-defined format

OUTPUT MODIFIERS:
Switch Description
-H show process hierarchy (forest)
-m show all threads
-n set namelist file
-w wide output
Cuse raw CPU time for %CPU instead
of decaying average Nspecify namelist
file
Osorting order (overloaded)
Sinclude some dead child process data
(as a sum with the parent) ctrue
command name
eshow environment
after the command
fASCII-art process
hierarchy (forest)
hdo not print header lines (repeat
header lines in BSD personality) mall
threads
nnumeric output for
WCHAN and USER
wwide output

--colsset screen width
--columns set screen width
--cumulative include some dead child process data (as a sum with the parent)
--forestASCII art process tree
--htmlHTML escaped output
--headersrepeat header lines
--no-headersprint no header line at all
--linesset screen height
--nulunjustified output with NULs
--nullunjustified output with NULs
--rowsset screen height
--sortsspecify sorting order
--widthset screen width
--zerounjustified output with NULs

INFORMATION:
Switch Description
-V print version
L list all
format
specifiers V
show version
info

--helpprint help message
--info print debugging info
--version print version

## Question 12: pstree

What is a pstree? How does it work?

**A:** A pstree is a UNIX command that shows the running processes as a tree. Its root is either init process or process for given PID. If a user name is specified, all process trees rooted at processes owned by that user are shown. It is used as alternative to ps command.

The command pstree visually merges identical branches by putting them in square brackets and prefixing them with the repetition count, *e.g.*

```
init-+-getty
|-getty
|-getty
`-getty
```

becomes

init---4*[getty]

OPTIONS:

-a

Show command line arguments. If the command line of a process is swapped out, that process is shown in parentheses. -a implicitly disables compaction.

-c

Disable compaction of identical subtrees. By default, subtrees are compacted whenever possible.

**-G**

Use VT100 line drawing characters.

-h
Highlight the current process and its ancestors. This is a no-op if the terminal doesn't support highlighting or if neither the current process nor any of its ancestors are in the subtree being shown.

**-H**

Like -h, but highlight the specified process instead. Unlike with -h, pstree fails when using -H if highlighting is not available.

-l
Display long lines. By default, lines are truncated to the display width or 132 if output is
sent to a non-tty or if the display width is unknown.

-n
Sort processes with the same ancestor by PID instead of by name. (Numeric sort.)

-p
Show PIDs. PIDs are shown as decimal numbers in parentheses after each process

name. -p implicitly disables compaction.
-u
Show uid transitions. Whenever the uid of a process differs from the uid of its parent, the new uid is shown in parentheses after the process name.

**-U**

Use UTF-8 (Unicode) line drawing characters. Under Linux 1.1-54 and above, UTF-8 mode is entered on the console with echo -e '\033%8' and left with echo -e '\033%@'

**-V**

Display version information.

-s

(Flask) Show Security ID (SID) for each process.

-x

(Flask) Show security context for each process.

## Question 13: Standard Input/Output

Many CLI programs use a feature

called input/output redirection. How do

they work?

**A:** The input/output redirection feature of CLI programs allows you to attach or fuse simple commands together in order to construct a more complex command.

STANDARD OUTPUT:

Many Linux commands print their output to screen. For example, cat does this when it lists the contents of a file and makes you see the output on your screen. However, the screen isn't the only place where the commands can print their output because you can redirect the output of several commands to files, devices and even to

the input of other commands.

The CLI programs that display their results do so usually by sending the results to standard output or stdout for short. By default, standard output directs its contents to the screen as you can see with the cat command but if you want to direct the output to somewhere else, you can use the > character, *e.g.* :

$ ls > dir_listing.txt

The above redirects the output of the ls command to a file called dir_listing.txt. You won't see any results of ls on your screen since the output is redirected to a file.

Each time you repeat $ ls > dir_listing.txt command, the file dir_listing.txt is overwritten with the results of the ls command. If you want to append the new results to the file instead of rewriting it, you can use >> instead:

$ ls >> dir_listing.txt

The new output of ls is added at the end of the dir_listing.txt file instead of overwriting the file each time you repeat $ ls >> dir_listing.txt command.

The following adds the contents of File1 at the end of File8:

$ cat File1 >> File8

You can also redirect it to devices using:

$ cat sound.wav > *dev*audio

Assume that the cat command concatenates a file named sound.wav and the results are sent to a device called *dev*audio. The above command plays the file sound.wav if your sound is properly configured.

STANDARD INPUT:

Many commands accept input from standard

input, or stdin for short. By default, standard input reads information from your keyboard and just like standard output, it can be redirected. To give you a clear idea, we can use a little program called tac when experimenting. This program reads standard input and then displays it to you with all the lines reversed. See how tac works when it reads the input from your keyboard. Give the command tac and type a few lines of text, using Enter for starting new lines. Then press Ctrl+D when you're done, tac then displays the text with all the lines reversed.

me@puter: ~$ tac

gorgeous me

also you

me@put
er: ~$

You can redirect the input so that tac gets it from a file instead of the keyboard. You can do it with the < character, like this:

$ tac < list_abc.txt
The above reads the input from a file called list_abc.txt and sends the results to standard output. Since the output isn't redirected anywhere, it will be displayed on your screen where you see the lines of lines_abc.txt in reverse order.

You can also redirect both a command's input and output:

$ tac < list_abc.txt > list_cba.txt

The above does the same thing as the previous command, only this time the results aren't displayed on your screen. Since the output is redirected, the results are written to a file called

list_cba.txt which then contains the same lines as list_abc.txt but in the reverse order.

## Question 14: Pipes

Can I take the output of one program and send it as the input of another one?

**A:** Yes and it is called piping. Pipes let you use the output of a program as the input of another one. For example, a simple pipe with sed:

```
ls -l | sed -e "s/[aeio]/u/g"
```

What happens here is that the command ls -l is executed and it's output. Instead of being printed, it is sent or piped to the sed program which in turn prints what it has to.

You can use pipes with many commands and one of the most commonly used is grep. It is a program that examines every line of the standard input it gets and searches for a specified pattern of characters. Then it sends to standard output every line that contains those characters.

Assume we have a text file called list.txt and we want to find out what lines of it contain the word "peace". It can be done easily with pipes. List the contents of the file list.txt and send the results to grep, which in turn filters all lines containing the desired word "peace" and displays those lines on your screen:

$ cat list.txt | grep peace

Like many Linux commands, grep is case sensitive. This means that the above matches only "peace", not "Peace" or "PEACE". With the -i option, the search is case insensitive:

$ cat applist.txt | grep -i desktop

A pipe is a very useful feature. Once you get more familiar with the CLI and learn more commands then you'll start to appreciate this feature each time you use it.

## Question 15: Command substitution

What is a command substitution? How does it function?

**A:** Command substitution is basically another way to do a pipe. You can use pipes and command substitution interchangeably. Command substitution can be done in two distinct ways.

Simply type: **`command_1 `command_2 -options``**

This will execute "command_2" and its output will become the input to "command_1".

The back-quote key is usually located at the same place as the tilde, above the [Tab] key.

Or type: **`command_1 $(command_2)`**
This will execute "command_2" and its output will become the input to "command_1".

You can of course use pipes to perform the same thing. For example, instead of doing:

less $cat file1.txt file2.txt

Do: **`cat file1.txt file2.txt | less`**
This way, you will end up with exactly the same result.

## Question 16: Top command

How important is it for an administrator to monitor the performance of their system?

**A:** One of the most important responsibilities a system administrator is to monitor their systems. If system resources become too low, it can cause a lot of problems. System resources can be taken up by individual users or by services your system may host such as email or web pages. The ability to know what is happening can help determine whether system upgrades are needed or if some services need to be moved to another machine.

The most common of these commands is top. The <command>top</command> provides an ongoing look at processor activity in real time. It displays a listing of the most CPU-intensive tasks on the system, and can provide an interactive interface for manipulating processes. It can sort the tasks by CPU usage, memory usage and runtime and can be better configured than the

standard top from the procps suite. Most features can either be selected by an interactive command or by specifying the feature in the personal or system-wide configuration file.

You can modify the output of top while it is running. If you hit an “I”, top will no longer display idle processes. Hit i again to see them again. Hitting M will sort by memory usage, S will sort by how long they processes have been running and P will sort by CPU usage again.

In addition to viewing options, you can also modify processes from within the top command. You can use u to view processes owned by a specific user, k to kill processes and r to remise them.

For more in-depth information about processes you can look in the *proc file system. In the* proc file system you will find a series of sub-directories with numeric names. These directories are associated with the processes ids of currently running processes. In each directory

you will find a series of files containing information about the process.

## Question 17: vi editor

Why should I opt to use vi editor?

**A:** Despite its age, “vi” is still an important tool today. This is demonstrated by the fact that every UNIX and Linux distribution released over the last twenty years is likely to have a copy installed by default. Since the original release of “vi”, many derivatives have been written (such as vim, vile and elvis), with usability and functionality added along the way. Because of this, “vi” is probably one of the most evolved and stable text editors in the world.

Many of the more recent vi clones are also intelligent enough to know about the type of document you are editing, and will help you out by automatically indenting your text (in the case of HTML or C, for example) or by highlighting the syntax to make your work much clearer. In the great Unix tradition, features that bug you or help you can be turned on and off at your

preference, and spending a few minutes customizing vi to your liking can save you hours of work in the long run.

The vi editor is a very powerful tool and has a very extensive built-in manual, which you can activate using the help command when the program is started. What makes vi confusing to the beginner is that it can operate in two modes: command mode and insert mode. The editor always starts in command mode. Commands move you through the text, search, replace, mark blocks and perform other editing tasks and some of them switch the editor to insert mode. This means that each key has not one but likely two meanings: it can either represent a command for the editor when in command mode or a character that you want in a text when in insert mode.

## Question 18: Basic vi commands

What are the basic vi commands?

**A:** The “vi” editor is a common editor for UNIX systems in that it makes use of a regular keyboard with an escape key. On the DECstation, the escape key is the F11 key. It therefore works on all UNIX computers. Complete documentation is available by typing
“man vi” at the

UNIX prompt.

Starting an Editing

Session:

To start vi just type vi at the operating system prompt. You will see a screen with a column of tildes (~) down the left side of the screen. This signifies an empty workspace. To edit a file, just include the filename after it, *e.g.* vi filename. You

will see the text of the file you included. Vi is now in command mode. The most basic command to enter insert mode is ‘i’ which lets you insert text to the left of the cursor.

In insert mode the characters you type are inserted into your document. You can use the backspace key to delete any typing mistakes you have made on the current input line. The escape key (<esc>) takes you out of insert mode and back to the command mode. If you are ever in doubt about what mode you are in, just press <esc> a few times until vi starts complaining. You will then know that you are in the command mode.

Command mode is where you do everything that isn't done in insert mode. In command mode the same keys that caused letters to appear on your screen in insert mode now represent totally different functions.

Here are some

popular commands:

Undo Command:

U - undo the last command.

Screen Commands:
CTL/l - Reprints current screen.
CTL/L - Exposes one more line
at top of screen. CTL/E -
Exposes one more line at
bottom of screen. CTL/F -
Pages forward one screen.
CTL/B - Pages
back one screen.
CTL/D - Pages
down half screen.
CTL/U - Pages up
half screen.

Cursor Positioning Commands:

j - Moves cursor down one line, same column.
k - Moves cursor up one
line, same column. h -
Moves cursor back one
character.
l - Moves cursor forward one character.
RET - Moves cursor to
beginning of next line. 0 -

Moves cursor to beginning
of current line.
$ - Moves cursor to end of
current line.  SPACE -
Moves cursor forward one
character.
nG - Moves cursor to beginning of line
n. Default is last line of file.  0 - Moves
the cursor to the first character of the
line.
:n - Moves cursor to beginning of line n.
b - Moves the cursor backward to the
beginning of the previous word.  e -
Moves the cursor backward to the end of
the previous word.
w - Moves the cursor forward to the next word.
/pattern - Moves cursor forward to next
occurrence of pattern.
?pattern - Moves cursor backward to
next occurrence of pattern. N -
Repeats last / or ? pattern search.

If you try to move somewhere that vi doesn't want to go, *e.g.* pressing h when the cursor is in

the left-most column, your terminal will complain by either beeping or flashing the screen.

Text Insertion Commands:

a - Appends text after cursor. Terminated by escape key.

A - Appends text at the end of the line. Terminated the escape key. i - Inserts text before cursor. Terminated by the escape key.
I - Inserts text at the beginning of the line. Terminated by the escape key.
o - Opens new line below the current line for text insertion. Terminated by the escape key.
O - Opens new line above the current line for text insertion. Terminated by the escape key.
DEL - Overwrites last character during text insertion.
ESC - Stops text insertion. The escape key on the DECstations is the F11 key.

Text Deletion Commands:

x - Deletes current character.
dd - Deletes current line.
dw - Deletes the current word.

d) - Deletes the rest of the
current sentence. D, d$ -
Deletes from cursor to
end of line. P - Puts back
text from the previous
delete.

Changing Commands:

cw - Changes characters of current word
until stopped with escape key. c$ -
Changes text up to the end of the line.
C, cc - Changes remaining text on current line
until stopped by pressing the escape key.

~ - Changes case of current character.
xp - Transposes current and
following characters. J -
Joins current line with next
line.
s - Deletes the current character and
goes into the insertion mode. rx -
Replaces current character with x.
R - Replaces the following characters until

terminated with the escape key. Cut and

Paste Commands:

yy - Puts the current line in a buffer. Does not delete the line from its current position.
p - Places the line in the buffer after the

current position of the cursor. Appending

Files into Current File:

:R filename - inserts the file filename where the cursor was before the ``:" was typed.

Exiting vi:

ZZ - Exits vi and saves changes.
:wq - Writes changes to current file and quits edit session.
:q! - Quits edit session (no changes made).

## Question 19: Word count

I want to find out the number of bytes and words in my file. How can I do this?

**A:** Use “wc” (short for word count), this is a command in Unix-like operating systems. The program reads either standard input or a list of files and generates one or more of the following statistics: number of bytes, number of words, and number of lines (specifically, the number of new line characters). If a list of files is provided, both individual file and total statistics follow.

For example:

$ wc ideas.txt
excerpt.txt 40
149 947
ideas.txt
2294166389772 excerpt.txt

**2334167879867 1 total**

The first column shows the number of lines, the second column shows the number of words and the last column is number of characters.

Newer versions of wc can differentiate between byte and character count. This difference arises with Unicode which includes multi-byte characters. The desired behavior is selected with the -c or -m switch.

Description:
Print byte, word, and new line counts for each FILE, and a total line if more than one FILE is specified. With no FILE, or when FILE is -, read standard input.

-c, --bytes
print the byte counts
-m, --chars
print the character counts

-l, --lines
print the newline counts
-L, --max-line-length
print the length of the longest line
-w, --words
print the word counts
--help
display this help and exit
--version
output version information and exit

## Question 20: tr

How can I translate and/or delete characters from a standard input?

**A:** Use tr (abbreviated from translate or transliterate), it is a command in Unix- like operating systems.

When executed, the program reads from the standard input and writes to the standard output. It takes as parameters two sets of characters, and replaces occurrences of the characters in the first set with the corresponding elements from the other set. The following inputs, for instance, shift the input letters of the alphabet back by one character.

```
$ echo "ibm 9000" >computer.txt
$ tr a-z za-y
<computer.txt
hal 9000
```

Description:

-c, --complement
first complement SET1
-d, --delete
delete characters in SET1, do not translate
-s, --squeeze-repeats
replace each input sequence of a repeated character that is listed in SET1 with a single occurrence of that character
-t, --truncate-set1
first truncate SET1 to length of SET2
--help
display this help and exit
--version
output version information and exit

SETs are specified as strings of characters. Mostrepresent themselves. Interpreted sequences are:

**\NNN**

character with octal value NNN (1 to 3 octal digits)

\\

backslash
\a
audible BEL
\b
backs
pace
\f
form feed
\n
new line
\r
retu
rn
\t
horizontal tab
\v
vertical

tab
CHAR1-
CHAR2
all characters from CHAR1 to
CHAR2 in ascending order
[CHAR*]
in SET2, copies of CHAR until
length of SET1
[CHAR*REPEAT]
REPEAT copies of CHAR, REPEAT
octal if starting with 0 [:alnum:]
all letters and
digits [:alpha:]
all
letters
[:blank:]
all horizontal
whitespace
[:cntrl:]
all control
characters
[:digit:]

all
digits
[:gra
ph:]
all printable characters,
not including space
[:lower:]
all lower
case letters
[:print:]
all printable
characters, including
space [:punct:]
all
punctuation
characters
[:space:]
all horizontal or
vertical whitespace
[:upper:]
all upper
case letters
[:xdigit:]

all
hexadecima
l digits
[=CHAR=]
all characters which are equivalent to CHAR

Translation occurs if -d is not given and both SET1 and SET2 appear. -t may be used only when translating. SET2 is extended to length of SET1 by repeating its last character as necessary. Excess characters of SET2 are ignored. Only [:lower:] and [:upper:] are guaranteed to expand in ascending order; used in SET2 while translating, they may only be used in pairs to specify case conversion. -s uses SET1 if not translating nor deleting; else squeezing uses SET2 and occurs after translation or deletion.

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 **********