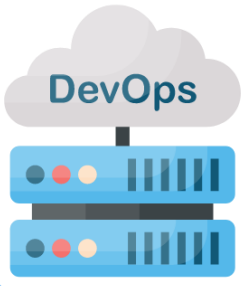
iNeuron

DevOps

Interview Questions

PART -3: Linux Basics



Question 21: nice

Is there a way to control or facilitate processes executing quickly so that the processes execute slowly so as not to interfere with other system activities?

A: UNIX processes have an associated system nice value which is used by the kernel to determine when it should be scheduled to run. This value can be increased to facilitate processes executing quickly or decreased so that the processes execute slowly and thus do not interfere with other system activities.

SYNOPSIS:
nice [OPTION]

[COMMAND

[ARG]...]

DESCRIPTION:

Run COMMAND with an adjusted scheduling priority. With no COMMAND, print the current scheduling priority. ADJUST is 10 by default. Range goes from
-20 (highest priority) to 19 (lowest).

-n, --adjustment=ADJUST

increment priority by ADJUST first
--help

display this help and exit
--version

output version information and exit

Question 22: join

I want to merge the lines of two sorted files based on the presence of a common field.

How do I accomplish that?

A: The command join is Unix-like operating system that merges the lines of two sorted text files based on the presence of a common field. It is a sort of implementation of the join operator used in relational databases but operating on text files.

The join command takes as input two text files and a number of options. If no command-line argument is given, this command looks for a pair of lines from the two files having the same first field (a sequence of characters that are different from space), and outputs a line composed of the first field followed by the rest of the two lines.

The program arguments specify which character

to be used in place of space to separate the fields of the line, which field to use when looking for matching lines, and whether to output lines that do not match. The output can be stored to another file rather than printing using redirection.

SYNOPSIS:
join [OPTION]... FILE1 FILE2

DESCRIPTION:
For each pair of input lines with identical join fields, write a line to standard output. The default join field is the first, delimited by whitespace. When FILE1 or FILE2 (not both) is -, read standard input.

-a SIDE
print unpairable lines coming from file SIDE
-e EMPTY
replace missing input fields with EMPTY
-i, --ignore-case

ignore differences in case when comparing
fields
-j FIELD
(obsolescent) equivalent to `-1 FIELD -2 FIELD'
-j1 FIELD
(obsolescent) equivalent to `-1 FIELD'
-j2 FIELD
(obsolescent) equivalent to `-2 FIELD'
-o FORMAT
obey FORMAT while constructing output line
-t CHAR
use CHAR as input and output field separator
-v SIDE
like -a SIDE, but suppress joined output lines

-1 FIELD

join on this FIELD of file 1

-2 FIELD

join on this FIELD of file 2
--help
display this help and exit
--version

output version information and exit

Unless -t CHAR is given, leading blanks separate fields and are ignored, else fields are separated by CHAR. Any FIELD is a field number counted from 1. FORMAT is one or more comma or blank separated specifications, each being `SIDE.FIELD' or `0'. Default FORMAT outputs the join field, the remaining fields from FILE1, the remaining fields from FILE2, all separated by CHAR.

As an example, the two following files list the known fathers and the mothers of some people:

wayne
nico
terry
solo

solo
krisha
wayne
kristel

The join of these two files (with no

argument) would produce: wayne nico

kristel

Indeed, only "wayne" is common as a first word of both files.

Question 23: Xargs

What is xargs?

A: The command “xargs” reads arguments from the standard input, delimited by blanks (which can be protected with double or single quotes or a backslash) or newlines, and executes the command (default is *bin*echo) one or more times with any initial-arguments followed by arguments read from standard input. Blank lines on the standard input are ignored. The command xargs exits with the following status:

0 if it succeeds **123** if any invocation of the command exited with status 1-125
124 if the command exited with status 255
125 if the command is killed by a signal
126 if the command cannot be run
127 if the command is not found
1 if some other

error occurred.

Options:

--null, -0
Input filenames are terminated by a null character instead of by whitespace, and the quotes and backslash are not special (every character is taken literally). Disables the end of file string, which is treated like any other argument. Useful when arguments might contain white space, quote marks, or backslashes. The GNU find -print0 option produces input suitable for this mode.

--eof[=eof-str], -e[eof-str]
Set the end of file string to eof-str. If the end of file string occurs as a line of input, the

rest of the input is ignored. If eof-str is omitted, there is no end of file string. If this
option is not given, the end of file string defaults to "_".

--help
Print a summary of the options to xargs and exit.

--replace[=replace-str], -i[replace-str]
Replace occurrences of replace-str in the initial arguments with names read from standard input. Also, unquoted blanks do not terminate arguments. If replace-str is
omitted, it defaults to "{}" (like for `find -exec'). Implies -x and -l 1.

--max-lines[=max-lines], -l[max-lines]
Use at most max-lines nonblank input lines per command line; max-lines defaults to 1
if omitted. Trailing blanks cause an input line to be logically continued on the next
input line. Implies -x.

--max-args=max-args, -n max-args
Use at most max-args arguments per command line. Fewer than max-args arguments
will be used if the size (see the -s option) is exceeded, unless the -x option is given, in which case xargs will exit.

--interactive, -p
Prompt the user about whether to run each command line and read a line from the terminal. Run the command line only if the response starts with `y' or `Y'. Implies -t.

--no-run-if-empty, -r
If the standard input does not contain any nonblanks, do not run the command.

Normally, the command is run once even if there is no input.

--max-chars=max-chars, -s max-chars
Use at most max-chars characters per command line, including the command and initial arguments and the terminating nulls at the ends of the argument strings.
The
default is as large as possible, up to 20k characters.

--verbose, -t
Print the command line on the standard error output before executing it.

--version
Print the version number of xargs and exit.

--exit, -x
Exit if the size (see the -s option) is exceeded.

--max-procs=max-procs, -P max-procs
Run up to max-procs processes at a time; the

default is 1. If max-procs is 0, xargs will run as many processes as possible at a time. Use the -n option with -P; otherwise chances are that only one exec will be done.

xargs is often used in conjunction with the Unix

commands find, locate and grep. For example:

1. ***find . -name ".foo" | xargs grep bar***

In practice does the same as grep bar:

`find . -name "*.foo"`

This will work even if there are so many files to search that they will not all fit on a single command line. It searches in all files in the current directory and its subdirectories which end in .foo for occurrences of the string bar.

2. ***find . -name*** **".foo" -print0 | xargs -0 grep bar**

This does the same thing but uses GNU specific extensions to find and xargs to separate filenames using the null character; this will work even if there are whitespace characters, including newlines, in the filenames.

Question 24: SED

What is SED and how is it used?

A: A Stream Editor (sed) is a simple and powerful computer program used to apply various pre-specified textual transformations to a sequential stream of text data. It reads input files line by line, edits each line according to rules specified in its simple language (sed script) and then outputs the line.

Sed is often thought of as a non-interactive text editor. It differs from conventional text editors in that the processing of the two inputs is inverted. Instead of iterating once through a list of edit commands applying each one to the whole text file in memory, sed iterates once through the text file applying the whole list of edit commands to each line. Because only one line at a time is in memory, sed can process text files with an arbitrarily-large number of lines. Some implementations of sed can only process

lines of limited lengths.

Most people use sed first for its substitution features. Sed is often used as a find- and-replace tool.

sed 's/Wayne/Nico/g' oldfile >newfile

It will replace every occurrence of "Wayne" with the word "Nico", wherever it occurs in the file. The "find" portion is a regular expression ("RE"), which can be a simple word or may contain special characters to allow greater flexibility (for example, to prevent "Wayne" from also matching "Wayneme").

sed can be used to add 8 spaces to the left side of a file, the printing wouldn't begin at the absolute left edge of a piece of paper.

sed 's/^/ /' myfile
>newfile# sed script
sed 's/^/ /' myfile | lp #
next sed script

sed could display only one paragraph of a file, beginning at the phrase "once upon a time" and ending at the phrase "happily ever after". The script looked like this:

sed -n '/once upon a time/,/happily ever after/p' myfile

Sed's normal behavior is to print (i.e., display or show on screen) the entire file, including the parts that haven't been altered, unless you use the -n switch. The "- n" stands for "no output". This switch is almost always used in conjunction with a 'p' command somewhere, which says to print only the sections of the file that have been specified. The
-n switch with the 'p' command allow for parts of a file to be printed (i.e., sent to the console).

Question 25: Job Control

I want to carry out a lengthy task in the background while using the terminal for another purpose. Is it possible to selectively stop or suspend the execution of some processes?

A: Most tasks (directory listing, editing files, etc.) can easily be accomplished by letting the program take control of the terminal and returning control to the shell when the program exits; however, sometimes the user will wish to carry out a lengthy task in the background while using the terminal for another purpose. Job control is a facility developed to make this possible. It allows the user to start programs in the background, send programs into the background, bring background programs into the foreground, and start and stop running programs. Programs under the influence of a job control facility are referred to as jobs.

The shell associates a job with each pipeline. It

keeps a table of currently executing jobs, which may be listed with the jobs command. When Bash starts a job asynchronously, it prints a line that looks like:

[1] 12345

Thus, indicating that this job is job number 1 and that the process ID of the last process in the pipeline associated with this job is 12345. All of the processes in a single pipeline are members of the same job. Bash uses the job abstraction as the basis for job control.

To facilitate the implementation of the user interface to job control, the operating system maintains the notion of a current terminal process group ID. Members of this process group (processes whose process group ID is equal to the current terminal process group ID) receive keyboard-generated signals such as SIGINT. These processes are said to be in the foreground. Background processes are those whose process group ID differs from the terminal's; such

processes are immune to keyboard-generated signals. Only foreground processes are allowed to read from or write to the terminal. Background processes which attempt to read from (write to) the terminals are sent a SIGTTIN (SIGTTOU) signal by the terminal driver, which, unless caught, suspends the process.

If the operating system on which Bash is running supports job control, Bash contains facilities to use it. Typing the suspend character (typically `^Z', Control- Z) while a process is running causes that process to be stopped and returns control to Bash. Typing the delayed suspend character (typically `^Y', ControlY) causes the process to be stopped when it attempts to read input from the terminal, and control to be returned to Bash. The user then manipulates the state of this job, using the bg command to continue it in the background, the fg command to continue it in the foreground, or the kill command to kill it. A `^Z' takes effect immediately and has the additional side effect of causing pending output and type ahead to be discarded.

There are a number of ways to refer to a job in the shell. The character `%' introduces a job name.

Job number n may be referred to as `%n'. The

symbols `%%' and `%+' refer to the shell's notion of the current job, which is the last job stopped while it was in the foreground or started in the background. A single `%' (with no accompanying job specification) also refers to the current job. The previous job may be referenced using `%-'. In output pertaining to jobs (e.g., the output of the jobs command), the current job is always flagged with a `+', and the previous job with a `-'.

A job may also be referred to using a prefix of the name used to start it, or using a substring that appears in its command line. For example, `%ab' refers to a stopped ab job. Using `%?ab', on the other hand, refers to any job containing the string `ab' in its command line. If the prefix or substring matches more than one job, Bash reports an error.

Simply naming a job can be used to bring it into the foreground: `%1' is a synonym for `fg %1', bringing job 1 from the background into the foreground. Similarly, `%1 &' resumes job 1 in

the background, equivalent to `bg %1'

The shell learns immediately whenever a job changes state. Normally, Bash waits until it is about to print a prompt before reporting changes in a job's status so as to not interrupt any other output. If the `-b' option to the set built-in is

enabled, Bash reports such changes immediately. Any trap on SIGCHLD is executed for each child process that exits.

If an attempt to exit Bash is made while jobs are stopped, the shell prints a message warning that there are stopped jobs. The jobs command may then be used to inspect their status. If a second attempt to exit is made without an intervening command, Bash does not print another warning, and the stopped jobs are terminated.

Job Control Built-ins:

bg
bg [jobspec ...]

Resume each suspended job jobspec in the background, as if it had been started with `&'. If jobspec is not supplied, the current job is used. The return status is zero unless it is run when job control is not enabled, or, when run with job control enabled, any jobspec was not found or specifies a job that was started without job control.

fg
fg [jobspec]

Resume the job jobspec in the foreground and make it the current job. If jobspec is not supplied, the current job is used. The return status is that of the command placed into the foreground, or non-zero if run when job control is disabled or, when run with job control enabled, jobspec does not specify a valid job or jobspec specifies a job that was started without job control.

jobs
jobs [-lnprs] [jobspec]
jobs -x command [arguments]

The first form lists the active jobs. The options have the following meanings:

-l
List process IDs in addition to the normal information.

-n
Display information only about jobs that have changed status since the user was last notified of their status.

-p
List only the process ID of the job's process group leader.

-r
Restrict output to running jobs.

-s
Restrict output to stopped jobs.

If jobspec is given, output is restricted to information about that job. If jobspec is not supplied, the status of all jobs is listed.

If the `-x' option is supplied, jobs replaces any jobspec found in command or arguments with the corresponding process group ID and execute command, passing it
arguments, returning its exit status.

kill
kill [-s sigspec] [-n signum] [-
sigspec] jobspec or pid kill -l
[exit_status]

Send a signal specified by sigspec or signum to the process named by job specification jobspec or process ID pid. sigspec is either a case-insensitive signal name such as SIGINT (with or without the SIG prefix) or a signal number; signum is a signal number. If sigspec and signum are not present, SIGTERM is used. The `-l' option lists the signal names. If any arguments are supplied when
`-l' is given, the names of the signals corresponding to the arguments are listed, and the return status is zero. exit_status is a number specifying a signal number or the exit status of a process terminated by a signal. The return status is zero if at least one signal was successfully sent, or non-zero if an error occurs or an invalid option is encountered.

wait
wait [jobspec or pid ...]

Wait until the child process specified by each process ID pid or job specification jobspec exits and return the exit status of the last command waited for. If a job spec is given, all processes in the job are waited for. If no arguments are given, all currently active child processes are waited for, and the return status is zero. If neither jobspec nor pid specifies an active child process of the shell, the return status is 127.

disown
disown [-ar] [-h] [jobspec ...]

Without options, each jobspec is removed from the table of active jobs. If the `-h' option is given, the job is not removed from the table, but is marked so that SIGHUP is not sent to the job if the shell receives a SIGHUP. If jobspec is not present, and neither the `-a' nor `-r' option is supplied, the current job is used. If no jobspec is supplied, the `-a' option means to remove or mark all jobs; the `-r' option without a jobspec argument restricts operation to running jobs.

suspend
suspend
[-f]

Suspend the execution of this shell until it receives a SIGCONT signal. The `-f' option means to suspend even if the shell is a login shell.

When job control is not active, the kill and wait built-ins do not accept jobspec arguments. They must be supplied process IDs.

Question 26: man pages

How should I view and search in the man pages?

A: If you want to know how some command is used or if you want more info about a particular command, you don't usually have to search for some text file that contains the help you need. You can read the reference manual for a command by simply typing man, and giving the name of the desired command as an argument to it. Almost every command on a Linux system has a manual page. For example, if you want more info about the ls command, you just type: $ man ls

Many X programs have man pages too. For example, the following gives you the man page for xterm which is a terminal emulator: $ man xterm

Use the following keys and commands for moving around in the manual page:

Command / key Action
e, j, Enter, or Down move
forward one line y, k, or
Up move backward one
line
f, Space, or Page Down
move forward one page b,
or Page Up move backward
one page
/characters search the manual page for the

specified characters

q quit

There are manual pages for different programs, utilities, functions and even for some configuration files. The man pages are a very quick and easy way of getting help. The man command itself has a manual page entry, so if you want more info about the man command itself: $ man man

Man pages reference are manual pages and they are usually very brief and technical. The man pages are great if you already know how to use the command and need only a reference or some extra info. They are definitely not tutorials and sometimes they're hard to understand. However, the man pages are very useful because they're always there, only a command away so you should try to learn to use them. The more man pages you read, the more you start to understand them.

Now, what if you need to do a specific task but

don't know what program or command to use for the task? Every man page entry contains a short description of the program but the problem is that you don't know what program to use and what man page to read. With the apropos command you can search those descriptions and find the right tool for the job. You can use keywords to search both program names and their descriptions.

Suppose you have a compressed gzip file and you want to uncompress it. You probably want to use a program whose description or name contains the word "gzip". You can use apropos for finding such a program:

```
$ apropos gzip
gzip (1) - compress or expand files
zforce (1) - force a '.gz' extension on all gzip
files

$
```

Now when you look at the output of apropos,

you probably see that gzip might be what you're looking for. Now you know the command name, and can go read

its man page:
$ man gzip

Although you can do very simple searches with apropos, it's a very powerful tool and you can do advanced searches. For example, you can use the shell wildcards in your searches with the -w option. For more info about apropos, read its man page.

While apropos searches for the descriptions in the man pages, whatis displays the description. It answers the question what is. Of course you can just take a look at the man page of a program to see what the purpose of it is, but if you're only interested in knowing what the program does, you can just use the whatis command for a quick answer:

$ whatis apropos
apropos (1) - search the manual page names and descriptions
$ whatis man
man (1) - an interface to the

on-line reference manuals
man (7) - macros to format
man pages
$ whatis whatis
whatis (1) - display manual page descriptions

$

Read the manual page for the man command
for more information about whatis.

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 **********