Sunday, March 05, 2017

LIKE GEEKS

LINUX

Linux Bash Scripting The Awesome Guide Part5

February 15, 2017 *admin* 0 Comments

On the last post, we've talked about **input and output and redirection in bash scripting**. Now you start building some Linux bash **scripts**, you may wonder how to run and control them on your Linux system. The only way we've run scripts is directly from the command line interface in real-time mode. This isn't the only way to run Linux bash scripts in Linux.

Our main points are:

Linux signals

There are various control methods include sending signals to your script, modifying a script's priority, and switching the run mode while a script is running. This post describes the different ways you can control your Linux bash scripts.

Linux signals

There are more than 30 Linux signals that can be generated by the system and applications and this is the most common Linux system signals that you'll run across in your Linux bash script writing

Signal	Value	Description
1	SIGHUP	Hangs up the process
2	SIGINT	Interrupts the process
3	SIGQUIT	Stops the process
9	SIGKILL	Unconditionally terminates the process
15	SIGTERM	Terminates the process if possible
17	SIGSTOP	Unconditionally stops, but doesn't terminate, the process
18	SIGTSTP	Stops or pauses the process, but doesn't terminate
19	SIGCONT	Continues a stopped process

If the bash shell receives a SIGHUP signal, such as when you leave an interactive shell, it exits. Before it exits, it passes the SIGHUP signal to any processes started by the shell, including any running shell scripts

With a SIGINT signal, the shell is just interrupted. The Linux kernel stops giving the shell processing time on the CPU. When this happens, the shell passes the SIGINT signal to any processes started by the shell to notify them.

You Linux bash scripts don't control these signals, you can program your bash script to recognize signals and perform commands to prepare the script for the consequences of the signal.

Generating signals

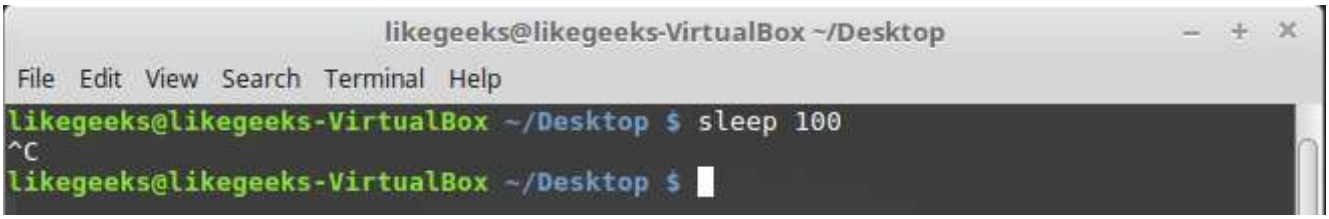
The bash shell allows you to generate two basic Linux signals using key combinations on the keyboard. This feature comes in handy if you need to stop or pause a running bash script

Stop a process

The Ctrl+C key combination generates a SIGINT signal and sends it to any processes currently running in the shell which simply stops the current process running in the shell.

```
$ sleep 100
```

Ctrl+C

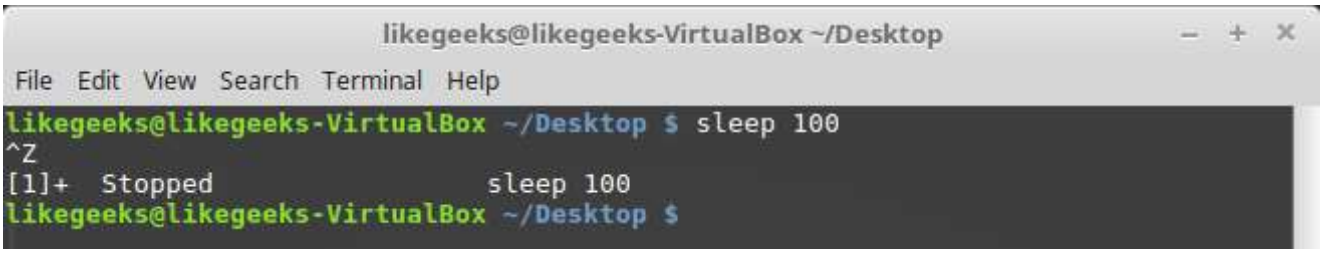


Pause a process

The Ctrl+Z key combination generates a SIGTSTP signal, stopping any processes running in the shell. Stopping a process is different than terminating the process. Stopping the process leaves the program in memory and able to continue running from where it left off.

```
$ sleep 100
```

Ctrl+Z

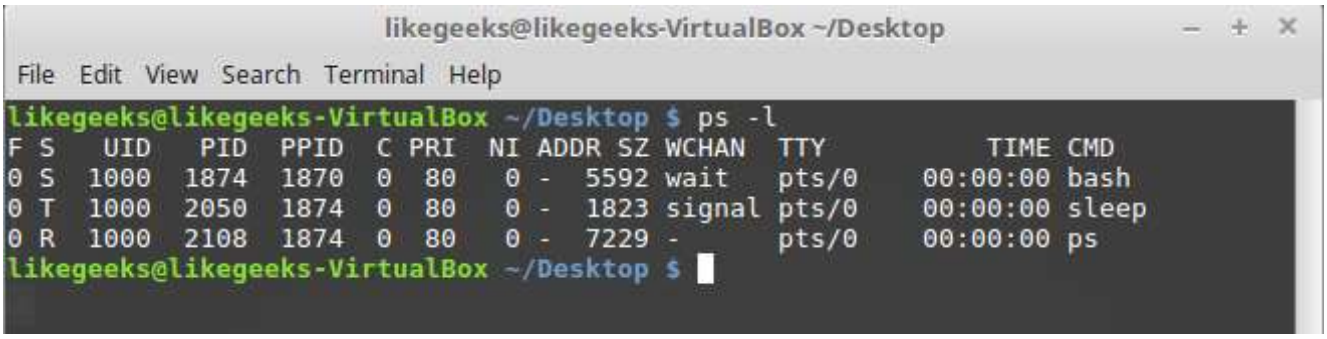


The number in the square brackets is the *job number* assigned by the shell. The shell refers to each process running in the shell as a job which is unique. It assigns the first started process job number 1, the second job number 2, and so on

If you have a stopped job assigned to your shell the bash warns you if you try to exit the shell.

You can view the stopped jobs using the ps command

```
ps -l
```



In the S column (process state), shows the stopped job's state as T. This indicates the command is either being traced or is stopped

If you want to terminate a stopped job you can kill its process by using kill command I recommend you to review the **basic Linux commands** if you need more info about kill command

```
kill processID
```

Trapping signals

The trap command allows you to specify which Linux signals your shell script can watch for and intercept from the shell. If the script receives a signal listed in the trap command, it prevents it from being processed by the shell and instead handles it locally.

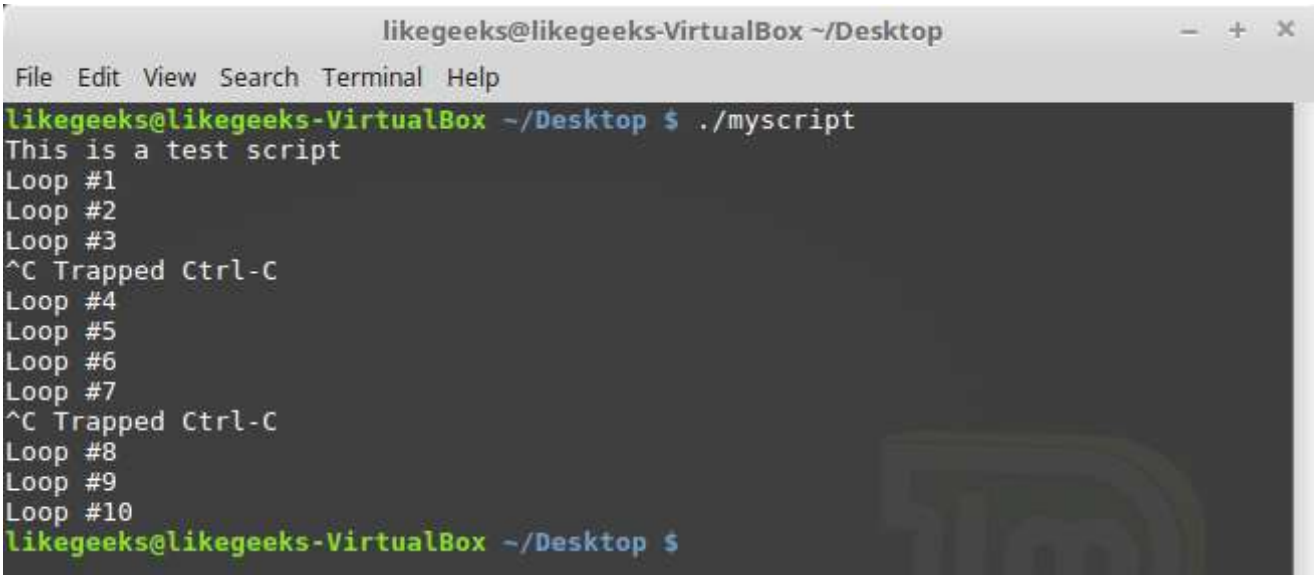
So instead of allowing your Linux bash script to leave signals ungoverned, you can use trap command to do that.

You just list the commands you want the shell to execute, along with a space-separated list of signals you want to trap like the following example

```
#!/bin/bash
trap "echo ' Trapped Ctrl-C'" SIGINT
echo This is a test script
```

```
count=1
while [ $count -le 10 ]
do
echo "Loop #$count"
sleep 1
count=$(( $count + 1 ))
done
```

The trap command used in this example displays a simple text message each time it detects the SIGINT signal when hitting Ctrl+C.



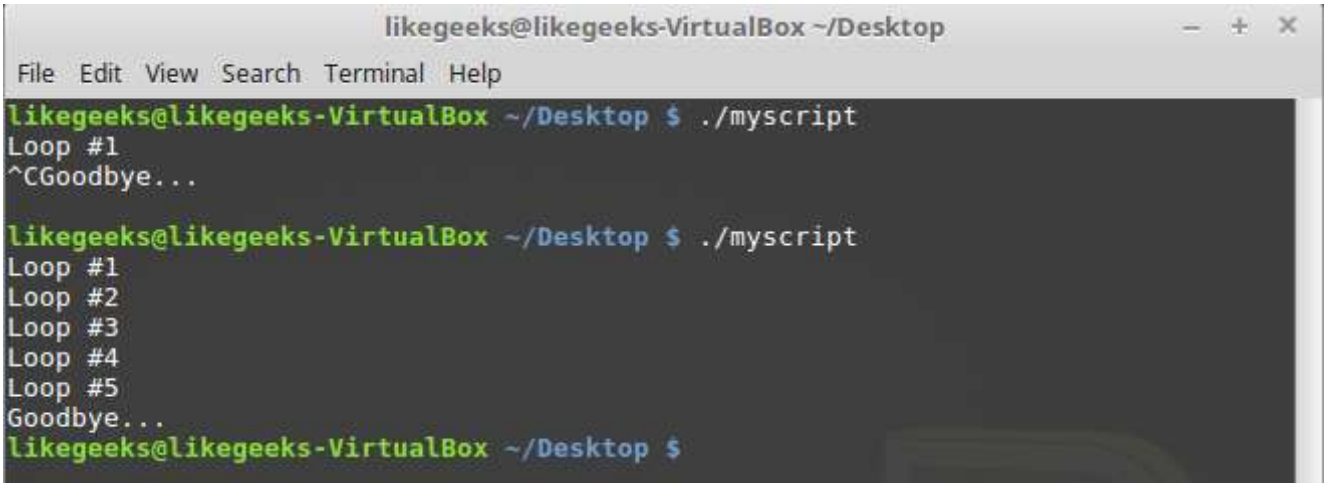
Each time the Ctrl+C key combination was used, the script executed the echo statement specified in the trap command instead of allowing the shell to stop the script. Cool right?

Trapping the script exit

You can trap them when the shell script exits; just add the EXIT signal to the trap command

```
#!/bin/bash
trap "echo Goodbye..." EXIT
count=1
while [ $count -le 5 ]
```

```
do
echo "Loop #$count"
sleep 1
count=$(( $count + 1 ))
done
```



When the Linux bash script gets exit, the trap is triggered and the shell executes the echo command specified

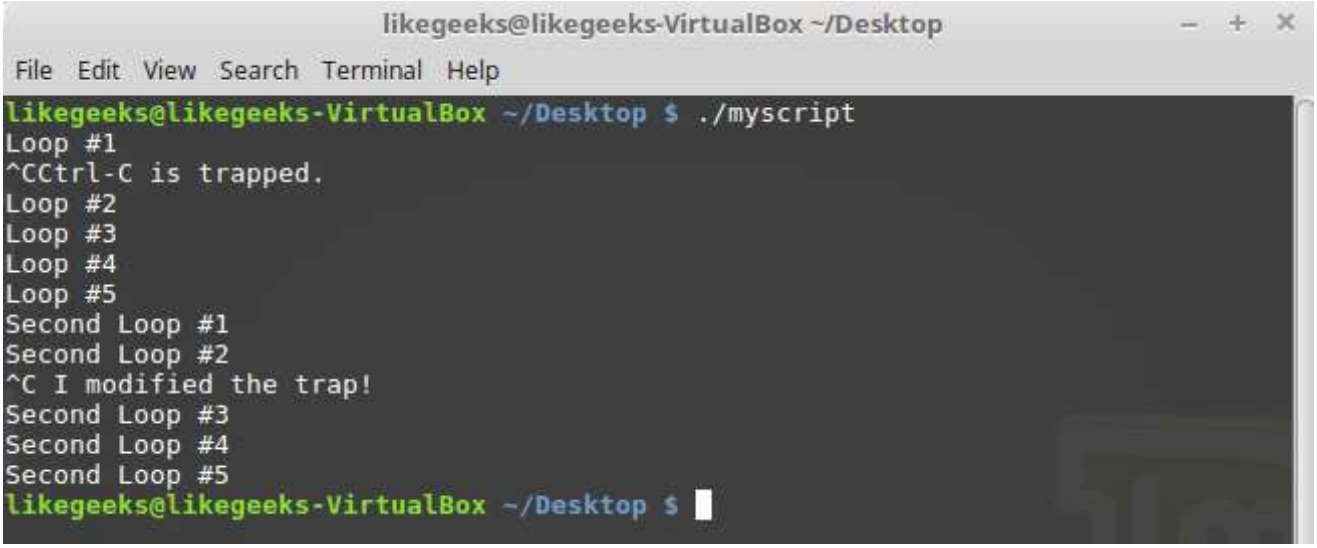
The EXIT trap also works if you prematurely exit the script

Modifying or removing a trap

You can reissue the trap command with new options

```
#!/bin/bash
trap "echo 'Ctrl-C is trapped.'" SIGINT
count=1
while [ $count -le 5 ]
do
echo "Loop #$count"
sleep 1
count=$(( $count + 1 ))
```

```
done
trap "echo ' I modified the trap!'" SIGINT
count=1
while [ $count -le 5 ]
do
echo "Second Loop #$count"
sleep 1
count=$(( $count + 1 ))
done
```



After the signal trap is modified, the bash script manages the signal or signals differently.

You can also remove a set trap. Simply add two dashes after the trap command

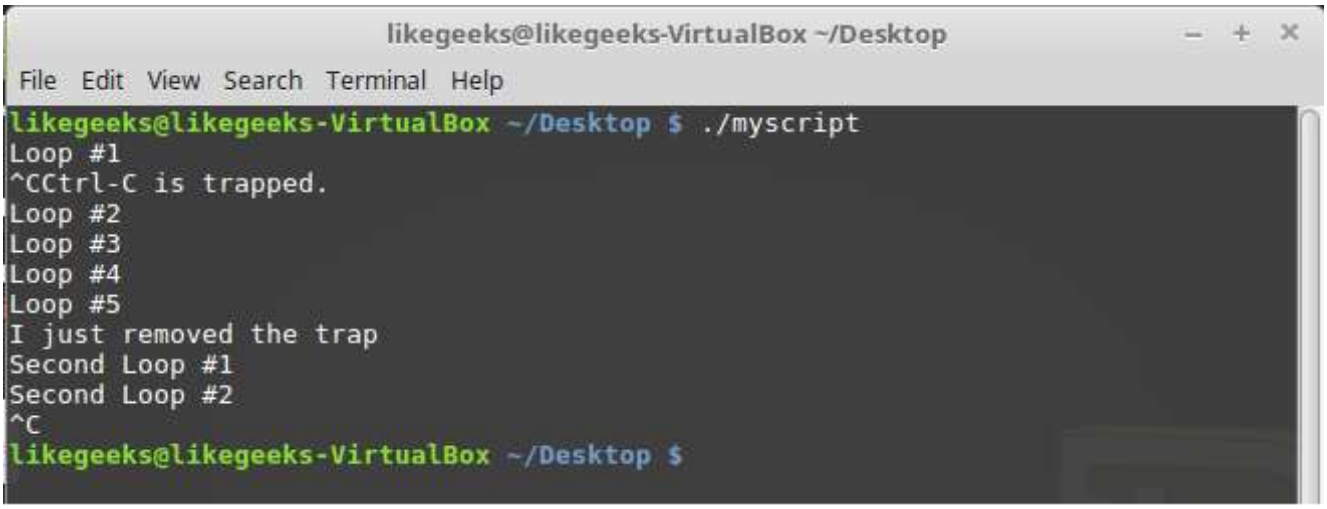
```
#!/bin/bash
trap "echo 'Ctrl-C is trapped.'" SIGINT
count=1
while [ $count -le 5 ]
do
echo "Loop #$count"
sleep 1
```

```
count=$(( $count + 1 ))
done
trap -- SIGINT
echo "I just removed the trap"
count=1
while [ $count -le 5 ]
do
echo "Second Loop #$count"
sleep 1
count=$(( $count + 1 ))
done
```

If a signal is received before the trap is removed, the script processes it per the original trap command

```
$ ./myscript
```

Crtl+C



The first Ctrl+C were used to attempt to terminate the script. Because the signal was received before the trap was removed, the script executed the echo command specified in the trap. After the script executed the trap removal, then Ctrl+C could terminate the bash script

Running Linux bash scripts in background mode

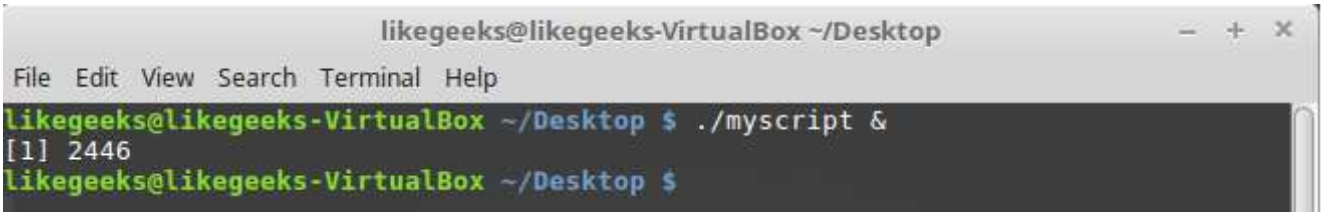
Sometimes your Linux bash scripts can take a long time to process, and you may not want to tie up the command line interface waiting, you can't do anything else in your terminal session. Fortunately, there's a simple solution to that problem.

If you see the output of the ps command you will see all the running processes in the background and not tied to the terminal.

We can do the same just place ampersand symbol after the command

```
#!/bin/bash
count=1
while [ $count -le 10 ]
do
sleep 1
count=$(( $count + 1 ))
done
```

```
$ ./myscipt &
```



Once you've done that it runs in a separate background process on the system and you can see the process id between the square brackets and when the background process finishes, it displays a message on the terminal that it is done.

Notice that while the background process is running, it still uses your terminal monitor for STDOUT and STDERR messages so if the error occurs you will see the error message and normal output also.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File  Edit  View  Search  Terminal  Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript &
[1] 2552
likegeeks@likegeeks-VirtualBox ~/Desktop $ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  2536  2532  0  80   0 -  5592 wait   pts/0    00:00:00 bash
0 S  1000  2552  2536  0  80   0 -  3134 wait   pts/0    00:00:00 myscript
0 S  1000  2555  2552  0  80   0 -  1823 hrtime pts/0    00:00:00 sleep
0 R  1000  2556  2536  0  80   0 -  7229 -      pts/0    00:00:00 ps
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

If the terminal session exit, the background process also exit

So what if you want to continue running even if you close the terminal?

Running Scripts without a Hang-Up

You can run your Linux bash scripts in the background process even if you exit the terminal session

You can do this using the nohup command.

The nohup command runs another command blocking any SIGHUP signals that are sent to the process. This prevents the process from exiting when you exit your terminal.

```
$ nohup ./myscript &
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File  Edit  View  Search  Terminal  Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ nohup ./myscript &
[1] 2605
likegeeks@likegeeks-VirtualBox ~/Desktop $ nohup: ignoring input and appending o
utput to 'nohup.out'
```

The nohup command disassociates the process from the terminal, the process loses the STDOUT and STDERR output links. To accommodate any output generated by the command, the nohup command automatically redirects STDOUT and STDERR messages to a file, called nohup.out

Note when running multiple commands from the same directory, because all the output is sent to the same nohup.out file

Viewing jobs

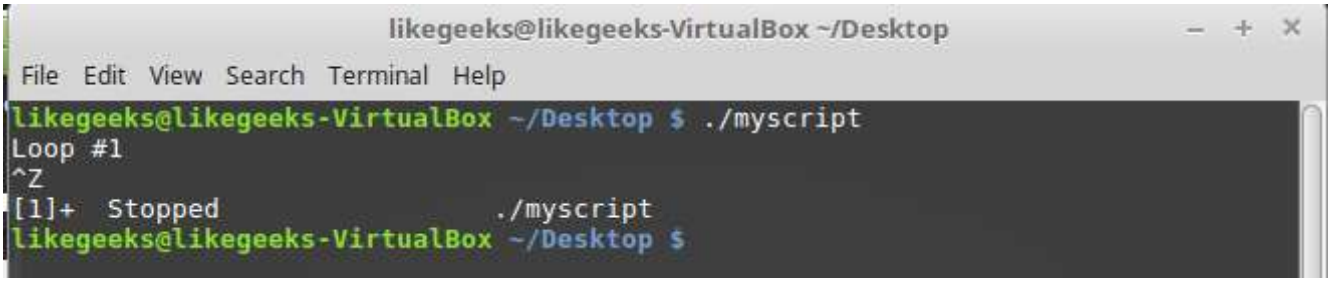
The jobs command allows you to view the current jobs being handled by the shell.

```
#!/bin/bash
count=1
while [ $count -le 10 ]
do
echo "Loop #$count"
sleep 10
count=$(( $count + 1 ))
done
```

Then run it

```
$ ./myscript
```

Then stop it using the Ctrl+Z



Run the same bash script but in background using the ampersand symbol and to make life a little easier, I'm going to make the output of that script is redirected to a file so it doesn't appear on the screen

```
$ ./myscript > outfile &
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File  Edit  View  Search  Terminal  Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Loop #1
^Z
[1]+  Stopped                 ./myscript
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript > outfile &
[2] 2732
likegeeks@likegeeks-VirtualBox ~/Desktop $ jobs -l
[1]+  2721 Stopped                 ./myscript
[2]-  2732 Running                 ./myscript > outfile &
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The jobs command shows both the stopped and the running jobs

```
jobs -l
```

-l parameter to view the process ID

Restarting stopped jobs

To restart a job in background mode, use the bg command

```
$ ./myscript
```

Ten press Ctrl+z

Now it is stopped

```
$ bg
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File  Edit  View  Search  Terminal  Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Loop #1
^Z
[1]+  Stopped                 ./myscript
likegeeks@likegeeks-VirtualBox ~/Desktop $ bg
[1]+ ./myscript &
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

As you can see it is now running in background mode.

If you have multiple stopped jobs you can do the same by specifying the job number to the bg command.

To restart a job in foreground mode, use the fg command.

```
$ fg 1
```

Scheduling a job

The Linux system provides a couple of ways to run a bash script at a preselected time: the at command and the cron table

The at command

This is the format of the command

at [-f *filename*] *time*

The at command recognizes lots of different time formats

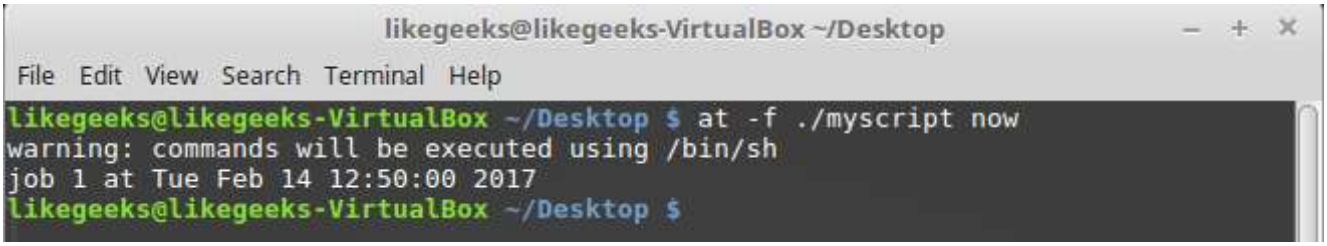
- A standard hour and minute, such as 10:15
- An AM/PM indicator, such as 10:15PM
- A specific named time, such as now, noon, midnight

In addition to specifying the time to run the job, you can also include a specific date, using a few different date formats

- A standard date format, such as MMDDYY, MM/DD/YY, or DD.MM.YY
- A text date, such as Jul 4 or Dec 25, with or without the year
- Now + 25 minutes
- 10:15PM tomorrow
- 10:15 + 7 days

We don't want to dig deep into the at command but for now, just make it simple and we will discuss it in detail in future posts.

```
$ at -f ./myscript now
```



The –M parameter is to send output to e-mail if the system has e-mail and if not this will suppress the output of the at command

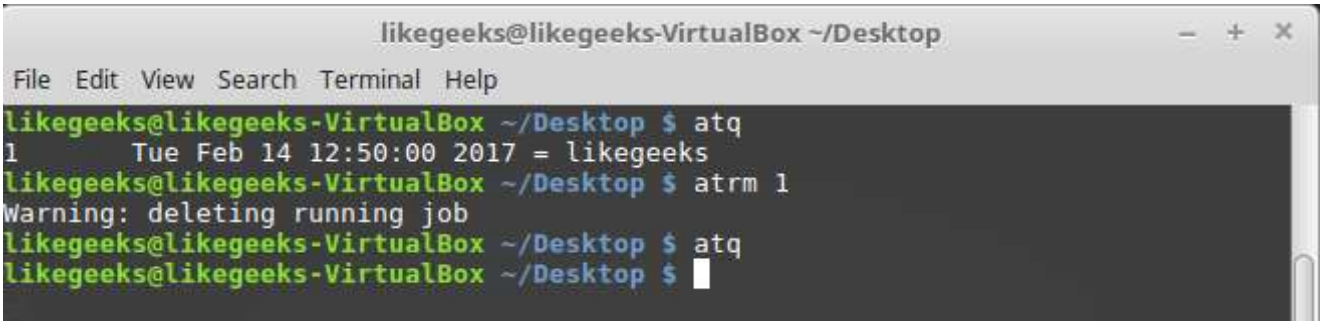
To list the pending jobs use atq command

```
$ atq
```

Removing jobs

you can use the atrm command to remove a pending job by specifying the job number

```
$ atrm 18
```



Scheduling scripts

Using the at command to schedule a script to run at a preset time is great, but what if you need that script to run at the same time every day or once a week or once a month.

The Linux system uses the crontab command to allow you to schedule jobs that need to run regularly.

The crontab program runs in the background and checks special tables, called cron tables, for jobs that are scheduled to run

To list an existing cron table, use the -l parameter

```
$ crontab -l
```

The format for crontab is

minute,Hour, dayofmonth, month, and dayofweek

So if you want to run a command at 10:30 on every day, you would use this cron table entry

30 10 * * * command

The wildcard character (*) used in the dayofmonth, month, and dayofweek fields indicates that cron will execute the command every day of every month at 10:30.

To specify a command to run at 4:30 PM every Monday, you would use the following

30 16 * * 1 command

The day of the week start from 0 to 6 where 0 is Sunday and 6 is Saturday

Here's another example: to execute a command at 12 noon on the first day of every month, you would use the following format

00 12 1 * * command

The day of the month is from 1 to 31

Let's keep it simple for now and we will discuss the cron in great detail in future posts.

To add entries to your cron table, use the -e parameter like this

```
crontab -e
```

Then type your command like the following

```
30 10 * * * /home/likegeeks/Desktop/myscript
```

This will schedule our script to run at 10:30 every day

Note sometimes you see error says Resource temporarily unavailable.

All you have to do is this

$ rm -f /var/run/crond.pid

You should be root user

Just that simple!

You can use one of the pre-configured cron script directories. There are four basic directories: hourly, daily, monthly, and weekly

/etc/cron.hourly

/etc/cron.daily

/etc/cron.weekly

/etc/cron.monthly

Just put your bash script file on any of those directories and it will run periodically.

Starting scripts with a new shell

Remember from the previous posts we've talked about startup files I recommend you to review the previous posts to get the point.

$HOME/.bash_profile

$HOME/.bash_login

$HOME/.profile

Just place any scripts you want to run at login time in the first file listed.

Ok but what about running our bash script when the shell opens? Easy

Type your script on `.bashrc` file

And now if you open the shell window it will execute that command

This for now, I hope you find the post useful.

Thanks.

60

Admin

https://likegeeks.com

RELATED ARTICLES

LINUX

Bash scripting the awesome guide part6 Bash functions

February 17, 2017 admin

Before we talk about bash functions let's discuss this situation. When writing bash scripts, you'll find yourself that you are using the same code in multiple places. If you get tired writing the same blocks of code over and over in your bash script. It would be nice to just write the block of code […]

LINUX

Main Linux Commands Easy Guide

January 31, 2017 admin

Main Linux Commands That You Should Know As A Beginner On previous post we discussed how to install Linux now we are going to talk about the most powerful thing in Linux which is Linux commands or shell commands for the whole documentation of Linux Commands, you can check Linux Documentation if you will use Linux […]

LINUX

Best Linux Distro For 2017 That Fits Your Needs

January 29, 2017 admin

What Is The Best Linux Distro For 2017? So what is the best Linux distro? If you know Linux you may know that there are a lot and a lot of Linux distros out there and you can check most of them from distro watch website https://distrowatch.com/ You may try few of them so […]

Shell scripting the awesome guide part4

Bash scripting the awesome guide part6 Bash functions