Experiment 17

Aim: Process scheduling: at, cron. **Process Priority:** nice, renice. **Running Process in background:** No logging out, nohup

One Time Task Scheduling using at Command in Linux

The at command schedules a command to be run once at a particular time that you normally have permission to run. The at command can be anything from a simple reminder message, to a complex script. You start by running the at command at the command line, passing it the scheduled time as the option. It then places you at a special prompt, where you can type in the command (or series of commands) to be run at the scheduled time. When you're done, press Control-D on a new line, and your command will be placed in the queue.

A typical at command sequence looks like this (commands you type are shown here in the blue box, or in bold face below):

```
at 9:30 PM Tue
warning: commands will be executed using /bin/sh
at> echo "It's 9:30 PM on Sunday."
at> ^D
job 1 at Sun Nov 16 09:30:00 2014
```

When we ran the command, the first thing at did was give us a "warning" telling us what command shell our commands will be run with: in this case, /bin/sh, the Bourne Shell. This shell is the traditional standard UNIX shell.

It then places us at the at> prompt. Here we type in a simple echo command, which echoes a string of text. We press enter, and we're placed at a new at> prompt. We then press Control-D, telling at we're all done with our commands. It then tells us that our job is job number 1 and that it will run next Tuesday.

Examples of at Command:

Example 1: Schedule task at coming 10:00 AM.

```
# at 10:00 AM
```

Example 2: Schedule task at 10:00 AM on coming Sunday.

```
at 10:00 AM Sun
```

Example 3: Schedule task at 10:00 AM on coming 25'th July.

```
at 10:00 AM July 25
```

Example 4: Schedule task at 10:00 AM on coming 22'nd June 2015.

```
at 10:00 AM 6/22/2015
at 10:00 AM 6.22.2015
```

Example 5: Schedule task at 10:00 AM on the same date at next month.

```
at 10:00 AM next month
```

Example 6: Schedule task at 10:00 AM tomorrow.

```
at 10:00 AM tomorrow
```

Example 7: Schedule task at 10:00 AM tomorrow.

```
at 10:00 AM tomorrow
```

Example 8: Schedule task to execute just after 1 hour.

```
at now + 1 hour
```

Example 9: Schedule task to execute just after 30 minutes.

```
at now + 30 minutes
```

Example 10: Schedule task to execute just after 1 and 2 weeks.

```
at now + 1 week
at now + 2 weeks
```

Example 11: Schedule task to execute just after 1 and 2 years.

```
at now + 1 year
at now + 2 years
```

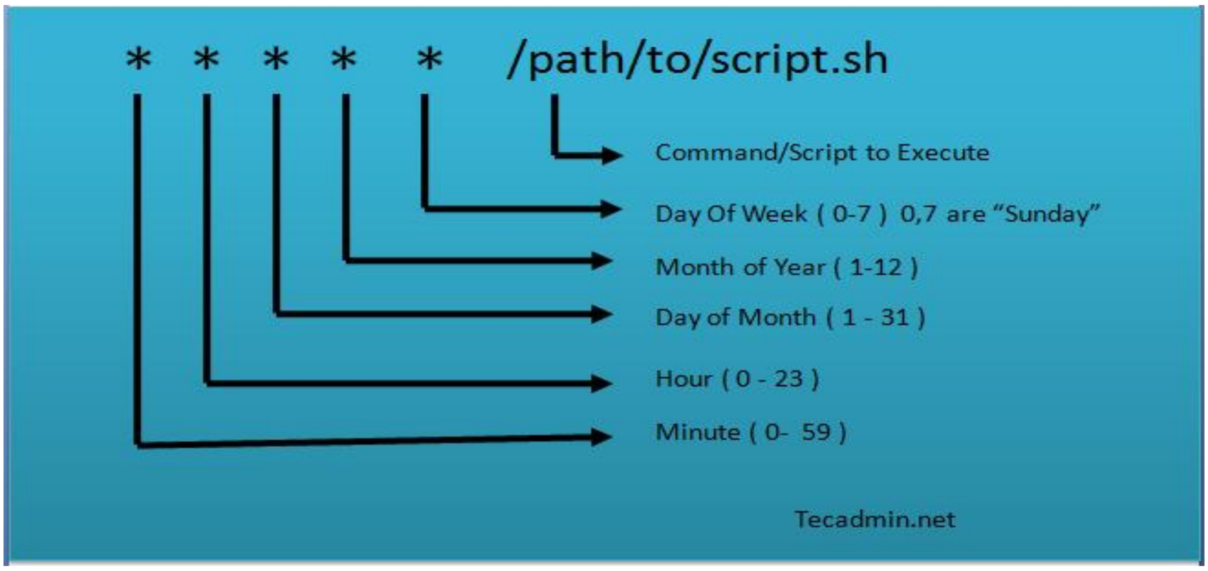
Example 12: Schedule task to execute at midnight.

```
at midnight
```

Linux Crontab Syntax

Linux crontab has six fields. 1-5 fields defines the date and time of execution. The 6'th fields are used for command or script to be executed.The Linux crontab syntax are as following:

```
[Minute] [hour] [Day_of_the_Month] [Month_of_the_Year] [Day_of_the_Week] [command]
```


* * * * * /path/to/script.sh
Command/Script to Execute
Day Of Week (0-7) 0,7 are "Sunday"
Month of Year (1-12)
Day of Month (1 - 31)
Hour (0 - 23)
Minute (0- 59)
Tecadmin.net

Scheduling a Job For a Specific Time

The basic usage of cron is to execute a job in a specific time as shown below. This will execute the Full backup shell script (full-backup) on 10th June 08:30 AM.

The time field uses 24 hours format. So, for 8 AM use 8, and for 8 PM use 20.

```
30 08 10 06 * /home/maverick/full-backup
```

30 – 30th Minute
08 – 08 AM
10 – 10th Day
06 – 6th Month (June)
* – Every day of the week

To schedule a job for every minute using Cron.

Ideally you may not have a requirement to schedule a job every minute. But understanding this example will will help you understand the other examples.

```
* * * * * CMD
```

The * means all the possible unit — i.e every minute of every hour through out the year. More than using this * directly, you will find it very useful in the following cases.

When you specify */5 in minute field means every 5 minutes.
When you specify 0-10/2 in minute field mean every 2 minutes in the first 10 minute.

Thus the above convention can be used for all the other 4 fields.

To schedule a job for more than one time (e.g. Twice a Day)
The following script take a incremental backup twice a day every day.

This example executes the specified incremental backup shell script (incremental-backup) at 11:00 and 16:00 on every day. The comma separated value in a field specifies that the command needs to be executed in all the mentioned time.

```
00 11, 16 * * * /home/maverick/bin/incremental-backup
```

00 – 0th Minute (Top of the hour)
11, 16 – 11 AM and 4 PM
* – Every day
* – Every month
* – Every day of the week

To schedule a job for certain range of time (e.g. Only on Weekdays)
If you wanted a job to be scheduled for every hour with in a specific range of time then use the following.

- Cron Job everyday during working hours :

This example checks the status of the database everyday (including weekends) during the working hours 9 a.m – 6 p.m

```
00 09-18 * * * /home/maverick/bin/check-db-status
```

00 – 0th Minute (Top of the hour)

09-18 – 9 am, 10 am, 11 am, 12 am, 1 pm, 2 pm, 3 pm, 4 pm, 5 pm, 6 pm

* – Every day

* – Every month

* – Every day of the week

- Cron Job every weekday during working hours :

  This example checks the status of the database every weekday (i.e excluding Sat and Sun) during the working hours 9 a.m – 6 p.m.

```
00 09-18 * * 1-5 /home/maverick/bin/check-db-status
```

00 – 0th Minute (Top of the hour)

09-18 – 9 am, 10 am, 11 am, 12 am, 1 pm, 2 pm, 3 pm, 4 pm, 5 pm, 6 pm
* – Every day

* – Every month1-5 -Mon, Tue, Wed, Thu and Fri (Every Weekday)

To schedule a background Cron job for every 10 minutes.

Use the following, if you want to check the disk space every 10 minutes.

```
*/10 * * * * /home/maverick/check-disk-space
```

It executes the specified command check-disk-space every 10 minutes throughout the year. But you may have a requirement of executing the command only during certain hours or vice versa. The above example shows how to do those things. Instead of specifying values in the 5 fields, we can specify it using a single keyword as mentioned below.

There are special cases in which instead of the above 5 fields you can use @ followed by a keyword — such as reboot, midnight, yearly, hourly.

Special words

For the first (minute) field, you can also put in a keyword instead of a number:

```
@reboot     Run once, at startup

@yearly     Run once  a year     "0 0 1 1 *"

@annually   (same as  @yearly)

@monthly    Run once  a month    "0 0 1 * *"

@weekly     Run once  a week     "0 0 * * 0"

@daily      Run once  a day      "0 0 * * *"

@midnight   (same as  @daily)

@hourly     Run once  an hour    "0 * * * *"
```

Leaving the rest of the fields empty, this would be valid:

```
@daily /bin/execute/this/script.sh
```

Storing the crontab output

By default cron saves the output of /bin/execute/this/script.sh in the user's mailbox (root in this case). But it's prettier if the output is saved in a separate logfile. Here's how:

```
*/10 * * * * /bin/execute/this/script.sh >> /var/log/script_output.log 2>&1
```

Explained

Linux can report on different levels. There's standard output (STDOUT) and standard errors (STDERR). STDOUT is marked 1, STDERR is marked 2. So the following statement tells Linux to store STDERR in STDOUT as well, creating one datastream for messages & errors:

```
2>&1
```

Now that we have 1 output stream, we can pour it into a file. Where > will overwrite the file, >> will append to the file. In this case we'd like to to append:

```
>> /var/log/script_output.log
```

To see what crontabs are currently running on your system, you can open a terminal and run:

```
$ sudo crontab -l
```

To edit the list of *cronjobs* you can run:

```
$ sudo crontab -e
```

Remove your crontab, effectively un-scheduling all crontab jobs.

```
crontab -r
```

niceness

When you run multiple things on your computer, like perhaps Chrome, Microsoft Word or Photoshop at the same time, it may seem like these processes are running at the same time, but that isn't quite true.

Processes use the CPU for a small amount of time called a time slice. Then they pause for milliseconds and another process gets a little time slice. By default, process scheduling happens in this round-robin fashion. Every process gets enough time slices until it's finished processing. The kernel handles all of these switching of processes and it does a pretty good job at it most of the time.

Processes aren't able to decide when and how long they get CPU time, if all processes behaved normally they would each (roughly) get an equal amount of CPU time. However, there is a way to influence the kernel's process scheduling algorithm with a nice value. Niceness is a pretty weird name, but what it means is that processes have a number to determine their priority for the CPU. A high number means the process is nice and has a lower priority for the CPU and a low or negative number means the process is not very nice and it wants to get as much of the CPU as possible.

The running instance of program is process, and each process needs space in RAM and CPU time to be executed, each process has its priority in which it is executed.

Now observe the below image and see column **NI**

```
$ top
```

You can see a column for NI right now, that is the niceness level of a process.

The column NI represents nice value of a process. It’s value ranges from -20 to 20(on most unix like operating systems).

```
  -20                 20
most priority        least priority
  process             process
```

One important thing to note is nice value only controls CPU time assigned to process and not utilisation of memory and I/O devices.

To change the niceness level you can use the nice and renice commands:

```
$ nice -n 5 apt upgrade
```

The nice command is used to set priority for a new process. The renice command is used to set priority on an existing process.

```
$ renice 10 -p 3245
```

What is the range of nice priority values?

Process priority values range from -20 to 19.

A process with the nice value of -20 is considered to be on top of the priority. And a process with nice value of 19 is considered to be low on the priority list.

Understand the above line very carefully, a nice value of numerically higher number is low on priority and a nice value of numerically low number is higher on priority.

Running a process with nice command with no options will set that process's priority to 10.(so you can considerably increase your process priority without any options.)

Normal users can only decrease their process priority. However root user can increase/decrease all process's priority.

Lets clear a confusion that majorly exists in terms of using nice.

```
nice -10 <command name>
```

and

```
nice -n 10 <command name>
```

will do the same thing.(both the above commands will make the process priority to the value 10).

A major misconception about nice command is that nice -10 <command name> will run that process with -10(a higher priority).

In order to assign -10 priority to a command then you should run it as shown below.

```
nice --10 <command name>
```

Make a note of the double --, the first one is the command option(-), and the second one is the numerical -(minus).

Note: You can only use nice command to change the process priority when you launch it. You cannot use nice command to change the priority of an already running process.

Usage of renice command :

To alter priority of running process, we use renice command.

renice value PID

value is new priority to be assigned
PID is PID of process whose priority is to be changed

One thing to note is you can't set high priority to any process without having root permissions though any normal user can set high priority to low priority of a process.
We will see one example of how you alter priority of process.

You may have nice value of process (PID = 2371) is 0, now let's try to set the new priority of 5 to this process.

```
renice 5 2371
```

Output:

```
2371 (process ID) old priority 0, new priority 5
```

Change the Scheduling Priority of a Process in Linux

As we mentioned before, Linux allows dynamic priority-based scheduling. Therefore, if a program is already running, you can change its priority with the renice command in this form:

```
$ renice -n  -12  -p 1055
$ renice -n -2  -u apache
```

Some examples of renice command is as follows:

(this will set the priority of process id no 3423 to -4, which will inturn increase its priority over others)

```
renice -4 -p 3423
```

(this will set the priority of the process id 3564 to 13, and all the process owned by user "sarath" to the priority of 13).

```
renice 13 -p 3564 -u sarath
```

(this will set all process owned by "sarath","satish" and also the group "custom" to 14)

```
renice 14 -u sarath,satish -g custom
```

How to change priority for all processes belonging to a group?

You can use the -g option for this. For example:

```
renice -n 20 -g howtoforge
```

The above command will change the priority of all processes belonging to the group 'howtoforge'.

How to change priority for all processes belonging to a user?

To change the priority for all programs associated to a user, use the -u option. For example:

```
renice -n 5 -u himanshu
```

Running Process in background:

&: No logging out

& runs the job in the background

Nohup Command in Linux

The meaning of **nohup** is '**no hangup**'. Normally, when we log out from the system then all the running programs or processes are hangup or terminated. If you want to run any program after log out or exit from Linux operating system then you have to use nohup command. There are many programs that require many hours to complete. We don't need to log in for long times to complete the task of the command. We can keep these type of programs running in the background by using nohup command and check the output later.

Using nohup command without '&'

When you run nohup command without **'&'** then it returns to shell command prompt immediately after running that particular command in the background. In the following example, **nohup** run **bash** command without '&' to execute sleep1.sh file in the background. The output of the **nohup** command will write in **nohup.out** the file if any redirecting filename is not mentioned in **nohup** command. For the following command, you can check the output of sleep1.sh by checking the output of nohup.out file.

```
$ nohup bash sleep1.sh
$ cat nohup.out
```

You can execute the command in the following way to redirect the output to the **output.txt** file. Check the output of **output.txt**.

```
$ nohup bash sleep2.sh > output.txt
$ cat output.txt
```

Using nohup command with '&'

When **nohup** command use with **'&'** then it doesn't return to shell command prompt after running the command in the background. But if you want you can return to shell command prompt by typing '**fg**'

```
$ nohup bash sleep1.sh &
$ fg
```