Schedules and Routine activities

Session Objectives

- Scheduling
- Using the cron daemon
- The crontab file
- The crontab utility
- The at utility
- Background processing
- Identify background process
- Killing the background process
- Printing in Linux

Scheduling

Shell scripts are used for batch processing. But the user has to explicitly invoke the shell. Sometimes the user might want the shell script or utility to execute at scheduled times without having to explicitly call the utility.

For example, there can be an utility that reminds the user of his appointments for the day. Another example could be a utility that reminds the user to take a backup of his files at specific time intervals.

The shell provides two simple utilities to schedule the utilities that have to be run at specific times. These are the `cron` and `at` utilities.

Using the cron daemon

The cron daemon, `crond` is a program that starts after the user boots linux by the cron script in the `/etc/rc.d/init.d` directory on his system. This takes place automatically, so the user need not worry about starting the `cron` daemon every time he boots linux. The `crond` program runs in the background and checks several files. The first file to be checked is the crontab file in the /etc directory.

The `crontab` utility is used to submit the tasks to the cron daemon. All the tasks that have to be automated are submitted to the crontab file. The cron daemon looks at each entry of the crontab file and whenever the date, time and day of an entry in the file matches the system date, cron executes the command. If the command produces any output, the cron daemon sends the user a mail notifying the execution of the command. For every task executed by the scheduler, a mail is sent to the user by the cron daemon. Each user on the linux system has a separate crontab file which can be submitted to the cron daemon.

The Linux system administrator can grant or deny access to certain users. This can be done by specifying the names of the users in the `/etc/cron.allow` and `/etc/cron.deny` files. The user names that appear in the cron.allow file have access to the cron utilities. The user names that appear in the cron.deny file will be denied access to cron. Both these files contain the login names of the users to whom access is to be allowed or denied. Each line in this file should match with an entry in the `/etc/passwd` file. The passwd file stores one line information about each user on the Linux system such as the login name and default shell.

The crontab file

The crontab file contains the list of tasks that should be automated on the Linux system. It contains one line for each of the tasks that is to be automated. The format of the line is given below:

```
minute hour day-of-month month-of-year day-of-week command
```

The columns are described in the table below:

Columns	Valid values
Minute	0-59
Hour	0-23
Day-of-month	1-31

Month-of-year	1-12
Day-of-week	Sunday=0,Monday=1 and so on
Command	Any utility to be executed at the specified time

If any column has an asterisk (*), it implies that it can be any of the valid values applicable for that column.

Consider the following extract from a sample crontab file.

```
0 20 * * * sh back_up
```

The above line specifies the execution of the script back_up at 8 p.m every day. * have been used in the above file to denote all valid values for the other columns.

The crontab file should be stored in the user's HOME directory with the name crontab to avoid confusion. Only one crontab file can be submitted to the cron daemon at a time.

The crontab utility

The crontab utility instructs cron to execute the commands on a specific date and time. It supplies the crontab file to the cron daemon.

The syntax of the crontab command is:

```
$ crontab [-u user] file
$ crontab [-u user] [options]
```

For example, the following command

```
$ crontab crontab
```

invokes the crontab utility with the crontab file as the parameter. This command sends the crontab file to the cron daemon. This will replicate the file in the /var/spool/cron directory. It will create one file for each user with the user login name.

The following options can be used with the crontab command

-e The –e option allows us to directly modify the crontab file that resides with the cron daemon. The user need not resubmit his crontab file.

-l The –l option lists the contents of the crontab file stored for the user.

-r The –r option is used to delete the crontab file. This will remove the crontab file from the /var/spool/cron directory. But the crontab file created in the HOME directory is not affected.

The at utility

The at utility is another utility provided for the purpose of scheduling tasks. It can schedule tasks to be run only once (whereas the cron daemon is used to execute tasks at regular intervals). The crontab utility can be used when we want to execute tasks repetitively. But, the at command is used for scheduling a task which has to be done only once. Assume, you want to send a mail to your manager at a specified time. The at command can be used to send the mail even though you are not logged on to the Linux system at that particular time.

The following files are used with the at utility:

File	Usage
`/var/spool/at`	Stores all the scheduled commands of the users in this directory
`/var/spool/at/spool`	Maintains some files during the execution of a task
`/var/run/utmp`	Allows one to discover information about the current users of the system
`/etc/at.allow`	Lists the login names of the users who should be allowed to use the at utility.
`/etc/at.deny`	Lists the login names of the users who should be denied access to the at utility

The list of commands used to work with the at utility are:

Command	Usage
`at`	Accepts commands or shell scripts to be executed later by using the /bin/sh
`atq`	Displays the list of jobs pending for the current user. If the administrator is logged on, the pending jobs of all the users will be displayed
`atrm`	Removes a job from the list of pending jobs
`batch`	Executes the scheduled tasks only when the system load level is below a specific level. This is useful if your command is not related to an urgent task and can execute when the load on the system is not very high.

The syntax is

```
at [-V] [-q queue] [-f file] [-mldbv] TIME
atq [-V] [-q queue] [-v]
atrm [-V] job [-q queue] [-f file] [-mldbv] TIME
batch [-V] [-q queue][-f file][-mv][TIME]
```

The various options available with the at command are given below:

Option	Meaning
-V	Prints the version number on the monitor
-q queue	Specifies a single letter queue designation. Valid

	queue designations are a-z and A-Z. The a queue is the default queue for at and b queue for batch.
-m	Sends a mail to the user when the job is completed even if the output is not present
-f file	Specifies the file to be executed at the scheduled time
-l	Displays the list of jobs pending for the current user
-d	Same as the atrm command

For example, consider the following command

```
$ at –f  mess 10:33
warning: commands will be executed using /bin/sh
job 2 at 2001-05-18 10:33
```

The above command executes the file called mess at 10:33 a.m.

REVIEW QUESTIONS

1. The _____ utility supplies the crontab file to the cron daemon.
2. The _____ file contains the list of tasks that should be automated on the Linux system.
3. The user names that appear in the _____ file have access to the cron utilities.
4. The _____ utility can schedule tasks to be run only once

Background processing

Job control is a feature provided by many shells which allows controlling multiple running commands, or **jobs**, at once.

Every time the user runs a program, a process is started. The command `ps` displays a list of currently running processes. Here's an example:

```
$ ps
PID   TTY   TIME        CMD
9226  tty2  00:00:00    sh
9268  tty2  00:00:00    ps
```

The PID listed in the first column is the process ID, a unique number given to every running process. The last column, CMD, is the name of the running command.

A running process is known as a *job* to the shell. The terms *process* and *job* are interchangeable. However, a process is usually referred to as a "job" when used in conjunction with job control-a feature of the shell which allows you to switch between several independent jobs.
In most cases users are only running a single job at a time – the job being, whatever command they last typed to the shell. However, using job control, you can run several jobs at once, switching between them as needed. How might this be useful? Let's say that you're editing a text file and need to

suddenly interrupt your editing and do something else. With job control, you can temporarily suspend the editor, and get back at the shell prompt start working on something else. When you have finished, you can start the editor, and be back where you started, as if you never left the editor. This is just one example. There are many practical uses for job control.

Background processing is a feature of the shell by which time-consuming, non-interactive tasks can proceed in the background while the user continues with other processing. Thereby the system can perform many different tasks simultaneously.

Initiating a background process

Background process can be initiated by typing the **&** symbol at the end of the command. The shell understands that this command is to be executed in the background and issues a PID. The $ prompt is then issued for us to continue our work.

For example, consider the command `yes` which sends an endless stream of y's to standard output.

```
$ yes
```

You can kill the process by hitting your interrupt key, which is usually ctrl-C.

In case, we do not want to see the stream of y's, let us redirect the standard output of yes to /dev/null.

```
$ yes >/dev/null
```

Nothing is printed, but the shell prompt doesn't come back. This is because yes is still running, and is sending the stream of y's to /dev/null. Again, to kill the job, hit the interrupt key.
Let us suppose that we wanted the yes command to continue to run, but wanted to get our shell prompt back to work on other things. We can put yes into the background, which will allow it to run, but without need for interaction.

One way to put a process in the background is to append an `` &" character to the end of the command.

```
$ yes >/dev/null &
[1] 9346
$_
```

As you can see, we have our shell prompt back. But what is this [1] and 9346? Is the yes command really running?

The number [1] represents the job number for the yes process. The shell assigns a job number to every running job. Because yes is the one and only job that we're currently running, it is assigned job number 1. 9346 is the process ID, or PID, number given by the system to the job. Either number may be used to refer to the job.

Now the yes process is running in the background, continuously sending a stream of y's to /dev/null. To check on the status of this process, use the shell internal command `jobs`.

```
$ jobs
[1]+  Running     yes>/dev/null &
```

We can also use the ps command as demonstrated below to check on the status of the job.

```
$ ps
PID   TTY   TIME        CMD
9226  tty2  00:00:00    sh
9268  tty2  00:00:00    ps
9346  tty2  00:00:13    yes
```

Terminating the background process

To terminate the job, use the command kill. This command takes either a job number or a process ID number as an argument. This was job number 1, so using the command

```
$ kill %1
```

will kill the job. When identifying the job with the job number, you must prefix the number with a percent (`` %") character. Now that we've killed the job, we can use jobs again to check on it:

```
$ jobs
[1]+  Terminated  yes>/dev/null
```

The job can also be killed using the process ID (PID) number, which is printed along with the job ID when the job is started. In our example, the process ID is 9346, so the command

```
$ kill 9346
```

will kill the process.

Printing in Linux

Linux is a multi-user operating system. It helps users to share hardware resources like the printer. In linux, all users can print their files at a centrally located printer. The printer can be connected to a Linux server, a remote Linux/Unix machine.

By default, the name of the first printer that is installed is lp. The subsequent printers that are installed will have the names lp0,lp1,lp2 etc. By changing the value of the environment variable PRINTER, it is possible to change the default name of the printer. All the printing utilities have a –p option for specifying the destination printer. The printer name should be specified with this option.

The lpr command

The lpr command is used to print files on a local or a network printer. If any file name is not specified, the lpr command expects input from the standard input.

For example, the command

```
$ lpr myfile
```

sends the file `myfile` to the default printer to be printed.

The lpq command

The lpq command can be used to check the status of the print jobs that have been issued.

```
$ lpq

Rank        Owner       Job   Files       Total Size
active      meerav      3     myfile      555 bytes
```

The lprm command

The `lprm` command can be used to remove a print job from the line printer spooling queue. The print job identification number can be found from the output of the `lpq` command.

For example,

```
$ lprm 3
```

will remove the print job with the job ID 3 from the default printer.

The lpc command

The `lpc` command can be used by the Linux system administrator to control the operations of the line printer system. It can be used to

- Disable/enable printer spooling queues
- Rearrange the order of the jobs in a spooling queue
- Find the status of the printers and their queues

The pr command

The pr command can be used to format a large text file. It helps in formatting the page such as
- specifying the page size,
- the number of lines in a page,
- the borders and the headers and footers.

REVIEW QUESTIONS

1. The _____ command is used to terminate a process.
2. The information about the background process can be obtained using the _____ command.
3. The _____command can be used to remove a print job from the line printer spooling queue.
4. The kill command takes either a _____ or a _____ as an argument.
5. Background process can be initiated by typing the _____symbol at the end of the command.