The Linux *cron* Utility

- The term "**cron**" comes from the Greek word "**chronos**", which means time.

- ***cron*** is a *nix-based utility that allows tasks to be automatically activated by the cron daemon and executed in the background at precise intervals, as specified by the **crontab** file.

- These tasks are referred to as "cron jobs". The **crontab** (cron tabLE) is a file that is used to configure and schedule the activation and execution of any shell script or executable in the system.

- ***cron*** first reads the **/etc/crontab** file and then additionally reads the files in the **/etc/cron.d/** directory.

- The daemon then wakes up every minute, examines all stored crontabs, and checks each command to see if it should be run in the current minute.

- When executing commands, any output is mailed to the owner of the crontab (or to the user named in the MAILTO environment variable in the crontab, if such exists).

- The Fedora implementation of ***cron*** treats the files in **/etc/cron.d** as extensions to the /etc/crontab file (they follow the special format of that file, i.e. they include the user field). The intended purpose of this feature is to allow packages that require finer control of their scheduling than that available in /etc/cron.

- Some *nix systems provide hourly (**cron.hourly**), daily (**cron.daily**), weekly (**cron.weekly**), and monthly (**cron.monthly**) directories that contain scripts that will be activated on an hourly, daily, weekly, and monthly basis.

- Specifically, Fedora Linux uses this method to control when jobs run. These directories take the following form:

  /etc/cron.hourly – jobs that run once per hour
  /etc/cron.daily – jobs that run once per day
  /etc/cron.weekly – jobs that run one per week
  /etc/cron.monthly – jobs that run once per month

- To add a cronjob to run hourly, daily, weekly or monthly, simply add an executable shell script in the appropriate directory.

- Note that you can only use Bourne shell commands in crontab entries. This means you cannot use any extensions recognized by bash and other modern shells.

- Absolute path names are required for commands, for example ***"/bin/ls***".

- The general format of a cronjob specification is written as follows:

  ```
  * * * * * /bin/full-directory-pathname/script.sh
  ```

- As can be observed above, there are fields, represented by five asterisks. Each of the asterisks represents a time component of the overall time/date that will specify when the task is scheduled to run.

- These time components are in the following order (left to right):

  - minute (from 0 to 59)
  - hour (from 0 to 23)
  - day of month (from 1 to 31)
  - month (from 1 to 12)
  - day of week (from 0 to 6) (0=Sunday)

- An asterisk as an entry in a crontab line implies “every”. So, the above line means, execute the **/bin/full-directory-pathname/script.sh :**

  - **every** minute
  - of **every** hour
  - of **every** day of the month
  - of **every** month
  - and **every** day in the week.

- In short: This script is being executed every minute. Without exception.

- Consider the following, which is a sample of a crontab for Linux:

  # run-parts
  01 * * * * **root run-parts /etc/cron.hourly**
  02 4 * * * **root run-parts /etc/cron.daily**
  22 4 * * **0 root run-parts /etc/cron.weekly**
  42 4 1 * * **root run-parts /etc/cron.monthly**

- The first line specifies that during the first minute, of every hour, of every day, of every week, and every month, all the scripts in the **/etc/cron.hourly** directory will be run.
- The second line specifies that all of the scripts in the **/etc/cron.daily** will be executed daily at 0402 hrs (using the 24 hour clock).
- The third line specifies that all of the scripts in the **/etc/cron.weekly** directory will be executed every week, on day 0 (Sundays - the days are numbered from 0 to 6, with 0 being Sunday) at 0422 hrs.
- The last line specifies that all of the scripts in the **/etc/cron.monthly** will be executed on the first of every month at 0442 hrs.

- The following will run /usr/bin/foo every 15 minutes on every hour, day-of-month, month, and day-of-week. In other words, it will run every 15 minutes:

```
0,15,30,45 * * * * /usr/bin/foo
```

- The following will run /usr/bin/foo every 10 minutes on every hour, day-of-month, month, and day-of-week. In other words, it will run every 10 minutes:

```
0,10,20,30,40,50 * * * * /usr/bin/foo
```

- A better way to implement the above task would be to write it as follows (it does the same thing):

```
*/10 * * * * /usr/bin/foo
```

- The following example will run **`/usr/bin/foo`** at 12:10am on the first day of the month:

```
10 * 1 * * /usr/bin/foo
```

- Now, say for example we wanted to have a custom script and we wish to have it execute every minute. All we have to do is add the following line to **/etc/crontab** :

    * * * * * **root /home/foo/ex1**

- As an example of what the script "**ex1**" might contain, here is a simple shell script that will send an email (every minute) with a cc entry, a subject line, and an inline text attachment to a specified email address:

```
mail -c foo@domain.ca -s "attachment" bar@domain.ca < /root/file.txt
```

- Note that cron need not be restarted whenever a crontab file is modified. The crontab command updates the modtime of the spool directory whenever it changes a crontab.

- In this way cron checks each minute to see if its spool directory's modtime (or the modtime on /etc/crontab) has changed, and if it has, cron will then examine the modtime on all crontabs and reload those which have changed.

- Here is a sample of what you would see in **/var/log/cron** after a modification to the **crontab** to accommodate the example script given above:

```
Oct  9 16:51:01 milliways crond[10216]: (*system*) RELOAD (/etc/crontab)
Oct  9 16:51:01 milliways CROND[10276]: (root) CMD (/home/aman/email-scripts/etest1)
```

Storing the crontab output

- By default ***cron*** saves the output of ***/bin/full-directory-pathname/script.sh*** in the user's mailbox (root in this case).

- For user and custom application scripts, it is much better to save in a separate logfile:

```
*/10 * * * * /home/foo/ex1 2>&1 >> /var/log/script_output.log
```

- Recall that Linux can report output on different levels. Each level has its own descriptor, standard output (STDOUT) is 1, and standard error (STDERR) is marked 2.

- So in the above example, the statement **2>&1** tells Linux to store STDERR in STDOUT as well, creating one data stream for both messages and errors:

- Now that we have a single output stream, we can store all of the output into a file by redirecting the stream to a file and appending it to the existing contents:

  >> /var/log/script_output.log

Mailing the crontab output

- By default cron saves the output in the user's mailbox (root in this case) on the local system.

- We can modify this setting and configure crontab to forward all output to an email address of our choosing by inserting the following line at the start of the crontab file:

  MAILTO="name@domain.com"

- We can also configure crontab to send the cronjob output to an email address as follows:

***/10 * * * * /home/foo/script.sh 2>&1 | mail -s "cronjob ouput" name@domain.com**