Task Scheduling in Linux

Overview

This article discusses some tools used in a Linux system to schedule tasks to run automatically at specified time intervals or at any given point of time in the future. This primer will not cover these commands in-depth; this is just a brief introduction to using these commands. See the individual HOWTOS for each command for an in-depth look at all relevant options and configurations.

Some task-scheduling daemons used in Linux/UNIX are:

- `at` – schedule one-time tasks for the future
- `cron` – the periodic scheduler most commonly used
- `anacron` – anachronistic cron; a periodic scheduler that doesn't rely on the system being left on 24×7

Using at

The ***at*** command allows a user to execute commands or scripts at a specified time (required) and date (optional). The commands can be entered via standard input, redirection, or file.

```
darkstar:~% at
```

Interactive at

Using the command ***at*** with standard input (keyboard) is a little more complicated than typing one line in at the prompt. The command uses an internal “sub-shell” to gather the required information. Once the command information entry is complete, Ctrl+D (EOT) will signify entry completion. The ***-m*** flag specifies a mail message will be sent to the user when the job is finished, regardless if any output was created.

```
darkstar:~% at 12:01 -m
warning: commands will be executed using (in order) a) $SHELL b) login shell
c) /bin/sh
at> ./my_script.sh
at> <EOT>
job 4 at 2015-06-22 12:01
darkstar:~%
```

File-driven at

Commands can also be contained within a file and run by ***at***:

```
darkstar:~% at 12:32 -m -f /usr/local/bin/my_script.sh
warning: commands will be executed using (in order) a) $SHELL b) login shell
c) /bin/sh
job 8 at 2015-06-22 12:10
```

The **-m** flag will email the user after completion of the command; the **-f** flag specifies the command will read the job from a file, not from standard input. After the command is typed in (and the appropriate warning is displayed), the ***at*** job number[1)] is displayed.

at Internal Scheduling

The job numbers provided after a command is typed in, or when a file is read, allow the user to know which internal job will be run in sequential order. If a user wants to delete a specific task, all that needs to be known is this internal job number. To remove the job, the command ***atrm*** (**at** **rem**ove) is used:

```
darkstar:~%  at -l
7      2015-06-22 12:10 p tux
8      2015-06-22 12:15 p root
```

The command ***atq*** (**at** **q**ueue) is the same as ***at -l***:

```
darkstar:~%  atq
7      2015-06-22 12:10 p tux
8      2015-06-22 12:15 p root
```

To remove the user job, use ***atrm*** with the job number:

```
darkstar:~%  atrm 7
```

Using cron

cron is a daemon that runs tasks in the background at specific times. For example, if you want to automate downloads of patches on a specific day (Monday), date (2 July), or time (1300), ***cron*** will allow you to set this up in a variety of ways. The flexibility inherent in ***cron*** can allow administrators and power users to automate repetitive tasks, such as creating backups and system maintenance.

cron is usually configured using a *crontab* file. The following command will open your user account *crontab* file:

```
darkstar:~% crontab -e
```

To edit the system-level *crontab*, first log into the root account:

```
darkstar:~# crontab -e
```

If your system has ***sudo*** installed, type in:

```
darkstar:~% sudo crontab -e
```

The *crontab* file syntax is:

```
 # * * * * *  command to execute
 # │ │ │ │ │
 # │ │ │ │ │
 # │ │ │ │ └─────── day of week (0 - 6) (Sun(0) /Mon (1)/Tue (2)/Wed (3)/Thu
(4)/Fri (5)/Sat (6))
 # │ │ │ └──────────── month (1 - 12)
 # │ │ └────────────────── day of month (1 - 31)
 # │ └──────────────────────── hour (0 - 23)
 # └────────────────────────────── min (0 - 59)
```

Using an asterisk in any placeholder location, will match any value. For example, the following will run *example_script.sh* at noon (1200) everyday during the first three months of the year:

```
#For more information see the manual pages of crontab(5) and cron(8)
#
# min hr day month weekday command
#
#

0 11 * 1-3 * /home/user/example_script.sh
```

Using anacron

anacron is not installed in Slackware by default.[2)]

anacron is unique from ***cron*** in the respect that it does not expect the operating system to be running continuously like a 24×7 server. If the time of execution passes while the system is turned off, ***anacron*** executes the command automatically when the machine is turned back on. The reverse is **not** true for ***cron*** - if the computer is turned off during the time of scheduled execution, ***cron*** will not execute the job. Another key difference between ***anacron*** and ***cron*** is the minimum chronological "granularity" - ***anacron*** can only execute jobs by *day*, versus the ability of ***cron*** to execute by the *minute*. Finally, ***anacron*** can only be used by root, while ***cron*** can be used by root and normal users.

Sources

- Originally written by vharishankar
- Contrbutions by mfillpot, tdrssb

- Example crontab example modified from en.wikipedia.org/wiki/cron

howtos, task scheduling, needs attention, author vharishankar, author mfillpot

1) As distinct from a process ID (PID) known to the operating system

2) See Slackbuilds.org for more information on ***anacron*** on Slackware