

Planification de Tâches dans Linux

TRADUCTION en cours — [Cedric M.](#) 2015/09/11 01:52

Introduction

Cet article traite de certains outils utilisés dans un système Linux pour planifier des tâches à exécuter automatiquement à des intervalles de temps spécifiques ou à un moment donné. Cet article ne détaillera pas les commandes en profondeur; il est juste une brève introduction à l'utilisation de ces commandes. Lisez les HOWTOS individuels de chaque commandes pour plus de précisions sur leurs options.

Quelques daemons Linux/UNIX de planification de tâches :

NdT: Les liens suivants font référence aux versions originales.

- [at](#) - planification de tâches unique
- [cron](#) - le planificateur périodique le plus utilisé
- [anacron](#) - anachronistic cron; a periodic scheduler that doesn't rely on the system being left on 24x7

Utiliser at

La commande **at** permet à un utilisateur d'exécuter des commandes ou des scripts à une date (obligatoire) et heure (facultatif) spécifiques. La commande peut être passé sur l'entrée standard, en redirection ou dans un fichier.

```
darkstar:~% at
```

at en mode Interactif

Utiliser la commande **at** avec l'entrée standard (avec le clavier) est un petit peu plus compliqué que de taper une ligne de commande au prompt. La commande utilise un "sub-shell" pour rassembler les information demandées. Une fois que l'entrée de la commande d'information est complète, **Ctrl+D** (EOT) signifiera la fin de l'entrée. L'argument **-m** spécifie qu'un message mèl sera envoyé à l'utilisateur lorsque le job sera terminé, à moins qu'une sortie ait été créée.

```
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 number¹⁾ 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 remove**) 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 queue**) 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

- Initialement écrit par [vharishankar](#)
- Contributions de [mfillpot](#), [tdrssb](#)

- crontab d'exemple modifiée de en.wikipedia.org/wiki/cron

[translation in progress](#), [howtos](#), [task scheduling](#), [needs attention](#), [author vharishankar](#), [author mfillpot](#), [translator cedric](#)

¹⁾

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

²⁾

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

From:

<https://docs.slackware.com/> - **SlackDocs**

Permanent link:

https://docs.slackware.com/fr:howtos:general_admin:task_scheduling

Last update: **2015/09/14 17:54 (UTC)**

