Reprenez le contrôle à l'aide de Linux !

30 heures Facile

Mis à jour le 29/06/2021

Le contenu de ce cours n'est plus à jour

Nous avons archivé ce cours et n'actualiserons plus son contenu.

Accédez au contenu le plus récent en découvrant ce cours :

VOIR LE NOUVEAU COURS

Exécuter un programme à une heure différée

Nous savons lancer une commande pour qu'elle s'exécute tout de suite. Il est cependant aussi

possible de « retarder » son lancement.

Linux vous propose toute une série d'outils qui vous permettent de programmer à l'avance l'exécution d'une tâche, comme par exemple la commande **crontab** que nous allons étudier. Tous les outils que nous allons découvrir dans ce chapitre feront en outre appel à la notion de date. Nous allons donc dans un premier temps nous intéresser au formatage de la date.

date : régler l'heure

Nous commencerons par nous intéresser à la date et l'heure du moment, car tout dans ce chapitre tourne autour de la notion de date. ;-)

Je vous ai déjà présenté brièvement la commande `date`. Essayez-la à nouveau :

```
$ date
mercredi 10 novembre 2010, 12:27:25 (UTC+0100)
```

Sans paramètre, la commande nous renvoie donc la date actuelle, l'heure et le décalage horaire.

Personnaliser l'affichage de la date

Si vous regardez le manuel (`man date`), vous verrez qu'il est possible de personnaliser l'affichage de la date : vous pouvez choisir quelles informations vous voulez afficher et dans quel ordre (vous pouvez par exemple ajouter les nanosecondes ou encore le numéro du siècle actuel).

Pour spécifier un affichage personnalisé, vous devez utiliser un symbole `+` suivi d'une série de symboles qui indiquent l'information que vous désirez. Je vous recommande de mettre le tout entre guillemets.

Prenons quelques exemples pour bien comprendre :

```
$ date "+%H"
12
```

Le `+%H` est le **format de date**. `%H` signifie « le numéro de l'heure actuelle ». Il était donc 12 heures au moment où j'ai lancé la commande.

Essayons autre chose d'un peu plus compliqué :

```
$ date "+%H:%M:%S"
12:36:15
```

Ici, j'ai rajouté les minutes (`%M`) et les secondes (`%S`).
J'ai séparé les nombres par des deux-points, mais j'aurais très bien pu mettre autre chose à la

place :

```
$ date "+%Hh%Mm%Ss"
12h41m01s
```

Seule la lettre qui suit le `%` est interprétée. Mes lettres « h », « m » et « s » sont donc simplement affichées.

Mais comment tu sais que `%M` affiche le nombre de minutes, par exemple ?

Je lis le man de `date`, tout simplement.

C'est là que j'apprends comment afficher l'année, notamment :

```
$ date "+Bienvenue en %Y"
Bienvenue en 2010
```

À vous de jouer !

Modifier la date

La commande `date` permet aussi de changer la date.

Attention, il faudra être root pour faire cela (vous devrez placer un `sudo` devant par exemple).

Il faut préciser les informations sous la forme suivante : `MMDDhhmmYYYY`. Les lettres signifient :

- `MM` : mois ;
- `DD` : jour ;
- `hh` : heure ;
- `mm` : minutes ;
- `YYYY` : année.

Notez qu'il n'est pas obligatoire de préciser l'année. On peut donc écrire :

```
$ sudo date 11101250
mercredi 10 novembre 2010, 12:50:00 (UTC+0100)
```

La nouvelle date s'affiche automatiquement et elle est mise à jour sur le système.
Attention à bien respecter l'ordre des nombres : Mois - Jour - Heure - Minutes.

at : exécuter une commande plus tard

Vous souhaitez qu'une commande soit exécutée plus tard ? Pas de problème ! Il est possible de programmer l'exécution d'une commande avec `at` .

> Avec `at` , le programme ne sera exécuté qu'**une seule fois**. Si vous voulez que l'exécution soit répétée régulièrement, il faudra utiliser la `crontab` que nous verrons plus loin.

Exécuter une commande à une heure précise

La commande s'utilise en deux temps.

1. Vous indiquez à quel moment (quelle heure, quel jour) vous désirez que la commande soit exécutée.
2. Vous tapez ensuite la commande que vous voulez voir exécutée à l'heure que vous venez d'indiquer.

Il faut donc d'abord indiquer à quelle heure vous voulez exécuter votre commande, sous la forme `HH:MM` :

```
$ at 14:17
```

L'exécution des commandes est demandée à 14 h 17 aujourd'hui. Si vous tapez cela dans votre console, vous devriez voir ceci s'afficher :

```
$ at 14:17
warning: commands will be executed using /bin/sh
at>
```

`at` comprend que vous voulez exécuter des commandes à 14 h 17 et vous demande lesquelles. C'est pour cela qu'un **prompt** est affiché : on vous demande de taper les commandes que vous voulez exécuter à cette heure-là.

Pour cet exemple, nous allons demander de créer un fichier à 14 h 17 :

```
$ at 14:17
warning: commands will be executed using /bin/sh
at> touch fichier.txt
at> <EOT>
```

```
job 5 at Mon Nov 10 14:17:00 2010
```

Après avoir écrit la commande `touch`, `at` affiche à nouveau un prompt et vous demande une autre commande. Vous pouvez indiquer une autre commande à exécuter à la même heure... ou bien arrêter là. Dans ce cas, tapez `Ctrl + D` (comme si vous cherchiez à sortir d'un terminal). Le symbole `<EOT>` devrait alors s'afficher, et `at` s'arrêtera.

`at` affiche ensuite le numéro associé à cette tâche (à ce « job », comme il dit) et l'heure à laquelle il sera exécuté.

Attendez 14 h 17, et vous verrez que le fichier sera créé. :-)

Et si je veux exécuter la commande demain à 14 h 17 et non pas aujourd'hui ?

```
$ at 14:17 tomorrow
```

`tomorrow` signifie « demain ».

Et si je veux exécuter la commande le 15 novembre à 14 h 17 ?

```
$ at 14:17 11/15/10
```

La date est au format américain, les numéros du jour et du mois sont donc inversés : 11/15/10. 11 correspond au mois (novembre) et 15 au numéro du jour !

Exécuter une commande après un certain délai

Il est possible d'exécuter une commande dans 5 minutes, 2 heures ou 3 jours sans avoir à écrire la date.
Par exemple, pour exécuter la commande dans 5 minutes :

```
$ at now +5 minutes
```

... ce qui signifie « Dans maintenant (*now*) + 5 minutes ». Les mots-clés utilisables sont les suivants :

- `minutes` ;
- `hours` (heures) ;
- `days` (jours) ;

- `weeks` (semaines) ;
- `months` (mois) ;
- `years` (années).

Un autre exemple :

```
$ at now +2 weeks
```

... exécutera les commandes dans deux semaines.

`atq` et `atrm` : lister et supprimer les *jobs* en attente

Chaque fois qu'une commande est « enregistrée », `at` nous indique un numéro de job ainsi que l'heure à laquelle il sera exécuté.

Il est possible d'avoir la liste des jobs en attente avec la commande `atq` :

```
$ atq
13        Mon Nov 10 14:44:00 2010 a mateo21
12        Mon Nov 10 14:42:00 2010 a mateo21
```

Si vous souhaitez supprimer le job n° 13 (je ne sais pas, parce que ça porte malheur par exemple), utilisez `atrm` :

```
$ atrm 13
```

sleep : faire une pause

Le saviez-vous ? Vous pouvez enchaîner plusieurs commandes à la suite en les séparant par des points-virgules comme ceci :

```
$ touch fichier.txt; rm fichier.txt
```

`touch` est d'abord exécuté, puis une fois qu'il a fini ce sera le tour de `rm` (qui supprimera le fichier que nous venons de créer).

Parfois, enchaîner les commandes comme ceci est bien pratique... mais on a besoin de faire une pause entre les deux.
C'est là qu'intervient `sleep` : cette commande permet de faire une pause.

```
$ touch fichier.txt; sleep 10; rm fichier.txt
```

Cette fois, il va se passer les choses suivantes :

- `fichier.txt` est créé ;
- `sleep` fait une pause de 10 secondes ;
- `rm` supprime ensuite le fichier.

Par défaut, la pause est exprimée en secondes. Il est aussi possible d'utiliser d'autres symboles pour changer l'unité :

- `m` : minutes ;
- `h` : heures ;
- `d` : jours.

Pour faire une pause d'une minute :

```
$ touch fichier.txt; sleep 1m; rm fichier.txt
```

L'intérêt de `sleep` ne vous paraîtra peut-être pas évident tout de suite, mais retenez que cette commande existe car il est parfois bien pratique de faire une pause, par exemple pour s'assurer que la première commande a bien eu le temps de se terminer. ;-)

Vous pouvez aussi remplacer les points-virgules par des `&&` , comme ceci :

```
touch fichier.txt && sleep 10 && rm fichier.txt
```

Dans ce cas, les instructions ne s'enchaîneront que si elles se sont correctement exécutées. Par exemple, si `touch` renvoie une erreur pour une raison ou une autre, alors les commandes qui suivent (`sleep` , `rm`) ne seront pas exécutées.

crontab : exécuter une commande régulièrement

La « crontab » constitue un incontournable sous Linux : cet outil nous permet de programmer l'exécution régulière d'un programme.
Contrairement à `at` qui n'exécutera le programme qu'une seule fois, `crontab` permet de faire en sorte que l'exécution soit répétée : toutes les heures, toutes les minutes, tous les jours, tous les trois jours, etc.

Un peu de configuration...

Avant toute chose, nous devons modifier notre configuration (notre fichier `.bashrc`) pour

demander à ce que Nano soit l'éditeur par défaut. En général, c'est le programme « vi » qui fait office d'éditeur par défaut. C'est un bon éditeur de texte, mais bien plus complexe que Nano et je ne vous le présenterai que plus tard.
En attendant, rajoutez la ligne suivante à la fin de votre fichier `.bashrc` :

```
export EDITOR=nano
```

Vous pouvez aussi écrire la commande suivante :

```
$ echo "export EDITOR=nano" >>  ~/.bashrc
```

Cela aura pour effet d'écrire cette ligne à la fin de votre fichier `.bashrc` situé dans votre répertoire personnel.
Fermez ensuite votre console et rouvrez-la pour que cette nouvelle configuration soit bien prise en compte.

Cette petite configuration étant faite, attaquons les choses sérieuses.

La « crontab », qu'est-ce que c'est ?

`crontab` est en fait une commande qui permet de lire et de modifier un fichier appelé la « crontab ».
Ce fichier contient la liste des programmes que vous souhaitez exécuter régulièrement, et à quelle heure vous souhaitez qu'ils soient exécutés.

`crontab` permet donc de changer la liste des programmes régulièrement exécutés. C'est toutefois le programme `cron` qui se charge d'exécuter ces programmes aux heures demandées.
Ne confondez donc pas `crontab` et `cron` : le premier permet de modifier la liste des programmes à exécuter, le second les exécute.

Comment utilise-t-on `crontab` ?
Il y a trois paramètres différents à connaître, pas plus :

- `-e` : modifier la crontab ;
- `-l` : afficher la crontab actuelle ;
- `-r` : supprimer votre crontab. Attention, la suppression est immédiate et sans confirmation !

Commençons par afficher la crontab actuelle :

```
$ crontab -l
no crontab for mateo21
```

Normalement, vous n'avez pas encore créé de crontab. Vous noterez qu'il y a une crontab par utilisateur. Là j'édite la crontab de mateo21 car je suis loggé avec l'utilisateur mateo21, mais root a aussi sa propre crontab. La preuve :

```
$ sudo crontab -l
no crontab for root
```

Bien, intéressons-nous à la modification de la crontab. Tapez :

```
$ crontab -e
```

Si vous avez bien configuré votre `.bashrc` tout à l'heure (et que vous avez relancé votre console), cela devrait ouvrir le programme Nano que vous connaissez.
Si par hasard vous n'avez pas fait quelque chose correctement, c'est le programme « vi » qui se lancera. Comme vous ne le connaissez pas encore, tapez `:q` puis `Entrée` pour sortir. Vérifiez à nouveau votre configuration du `.bashrc` et n'oubliez pas de fermer puis de rouvrir votre console.

Modifier la crontab

Pour le moment, si votre crontab est vide comme la mienne, vous devriez voir uniquement ceci (capture d'écran de Nano) :

```
GNU nano 2.0.7       Fichier : /tmp/crontab.4u4jHU/crontab

# m h  dom mon dow   command

                           [ Lecture de 1 ligne ]
^G Aide      ^O Écrire    ^R Lire fich.^Y Page préc.^K Couper    ^C Pos. cur.
^X Quitter   ^J Justifier ^W Chercher  ^V Page suiv.^U Coller    ^T Orthograp.
```

Les champs

Le fichier ne contient qu'une seule ligne :

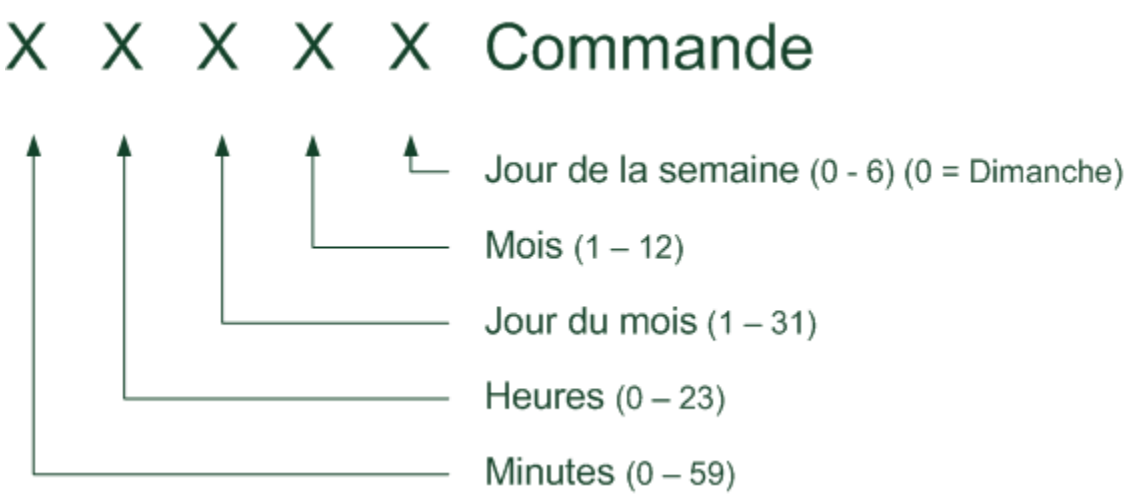
```
# m h  dom mon dow   command
```

Comme cette ligne est précédée d'un `#` , il s'agit d'un commentaire (qui sera donc ignoré).
Cette ligne vous donne quelques indications sur la syntaxe du fichier :

- `m` : minutes (0 - 59) ;

- `h` : heures (0 - 23) ;
- `dom` (*day of month*) : jour du mois (1 - 31) ;
- `mon` (*month*) : mois (1 - 12) ;
- `dow` (*day of week*) : jour de la semaine (0 - 6, 0 étant le dimanche) ;
- `command` : c'est la commande à exécuter.

Chaque ligne du fichier correspond à une commande que l'on veut voir exécutée régulièrement. Vous trouverez en figure suivante un schéma qui résume la syntaxe d'une ligne.



Crontab

En clair, vous devez d'abord indiquer à quel moment vous voulez que la commande soit exécutée, puis ensuite écrire à la fin la commande à exécuter.
C'est un peu comme un tableau. Chaque champ est séparé par un espace.

Chaque « X » sur mon schéma peut être remplacé soit par un nombre, soit par une étoile qui signifie « tous les nombres sont valables ».

Bien comprendre la crontab n'est pas si simple, je vous propose donc de nous baser sur quelques exemples pour voir comment ça fonctionne.

Imaginons que je veuille exécuter une commande tous les jours à 15 h 47. Je vais écrire ceci :

```
47 15 * * * touch /home/mateo21/fichier.txt
```

Seules les deux premières valeurs sont précisées : les minutes et les heures. Chaque fois qu'il est 15 h 47, la commande indiquée à la fin sera exécutée.

J'ai écrit le chemin du fichier en entier, car vous ne pouvez pas être sûrs que le `cron`

s'exécutera dans le répertoire que vous voulez. Il est donc toujours préférable d'écrire le chemin du fichier en absolu comme je l'ai fait ici : `/home/mateo21/fichier.txt` .

Au fait, pourquoi passer par la commande `crontab -e` pour modifier un fichier ? Il ne serait pas plus simple d'ouvrir le fichier directement avec `nano .crontab` , par exemple ?

Oui, mais ce n'est pas comme cela que ça fonctionne. La crontab exige de passer par une commande, c'est comme ça.
Il y a quelques avantages à cela, puisque cela permet au programme de vérifier si votre fichier est correctement écrit avant de mettre à jour la crontab. S'il y a une erreur de syntaxe, on vous le dira et aucun changement ne sera apporté.

Essayez d'enregistrer et de quitter Nano. Vous verrez que la crontab vous dit qu'elle « installe » les changements (elle les prend en compte, en quelque sorte) :

```
crontab: installing new crontab
mateo21@mateo21-desktop:~$
```

Désormais, `fichier.txt` sera créé dans mon répertoire personnel tous les jours à 15 h 47 (s'il n'existe pas déjà).

Revenez dans la crontab, nous allons voir d'autres exemples (tableau suivante).

Crontab	Signification
`47 * * * * commande`	Toutes les heures à 47 minutes exactement.> & Donc à 00 h 47, 01 h 47, 02 h 47, etc.
`0 0 * * 1 commande`	Tous les lundis à minuit (dans la nuit de dimanche à lundi).
`0 4 1 * * commande`	Tous les premiers du mois à 4 h du matin.
`0 4 * 12 * commande`	Tous les jours du mois de décembre à 4 h du matin.
`0 * 4 12 * commande`	Toutes les heures les 4 décembre.

Crontab	Signification
`* * * * * comman de`	Toutes les minutes !

Est-il possible d'exécuter une commande plus fréquemment que toutes les minutes ?

Non, c'est impossible avec `cron` . La fréquence minimale, c'est toutes les minutes.

Les différentes notations possibles

Pour chaque champ, on a le droit à différentes notations :

- `5` (un nombre) : exécuté lorsque le champ prend la valeur 5 ;
- `*` : exécuté tout le temps (toutes les valeurs sont bonnes) ;
- `3,5,10` : exécuté lorsque le champ prend la valeur 3, 5 ou 10. Ne pas mettre d'espace après la virgule ;
- `3-7` : exécuté pour les valeurs 3 à 7 ;
- `*/3` : exécuté tous les multiples de 3 (par exemple à 0 h, 3 h, 6 h, 9 h...).

Vous connaissiez déjà les deux premières notations. Celles que nous venons de découvrir nous permettent de démultiplier les possibilités offertes par la crontab.
Voici, sur le tableau suivante, quelques exemples d'utilisation.

Crontab	Signification
`30 5 1-15 * * commande`	À 5 h 30 du matin du 1er au 15 de chaque mois.
`0 0 * * 1,3,4 commande`	À minuit le lundi, le mercredi et le jeudi.
`0 */2 * * * commande`	Toutes les 2 heures (00 h 00, 02 h 00, 04 h 00...)
`*/10 * * * 1-5 commande`	Toutes les 10 minutes du lundi au vendredi.

Comme vous le voyez, la crontab offre de très larges possibilités (pour peu que l'on ait compris comment elle fonctionne).

Rediriger la sortie

Pour le moment, nous avons exécuté notre commande très simplement dans la crontab :

```
47 15 * * * touch /home/mateo21/fichier.txt
```

Toutefois, il faut savoir que si la commande renvoie une information ou une erreur, vous ne la verrez pas apparaître dans la console. Normal : ce n'est pas vous qui exécutez la commande, mais le programme `cron` .

Que se passe-t-il alors si la commande renvoie un message ? En fait, le résultat de la commande vous est envoyé par e-mail. Chaque utilisateur possède sa propre boîte e-mail sur les machines de type Unix, mais je ne vais pas m'attarder là-dessus. Nous allons plutôt voir comment rediriger le résultat.

Tenez : rediriger une sortie, vous savez faire ça, non ?

```
47 15 * * * touch /home/mateo21/fichier.txt >> /home/mateo21/cron.log
```

Tous les messages seront désormais ajoutés à la fin de `cron.log` . Tous ? Non, on oublie d'y rediriger aussi les erreurs !

```
47 15 * * * touch /home/mateo21/fichier.txt >> /home/mateo21/cron.log 2>&1
```

Voilà, c'est mieux.
Cette fois, tout sera envoyé dans `cron.log` : les messages et les erreurs.

Et si je ne veux pas du tout récupérer ce qui est affiché ?

Nous avons déjà appris à le faire ! Il suffit de rediriger dans `/dev/null` (le fameux « trou noir » du système). Tout ce qui est envoyé là-dedans est immédiatement supprimé : hop, plus de trace, le crime parfait.

```
47 15 * * * touch /home/mateo21/fichier.txt > /dev/null 2>&1
```

En résumé

- `date` permet d'obtenir la date et l'heure mais aussi de modifier celles-ci.
- `at` retarde l'exécution d'une commande à une heure ultérieure.
- On peut exécuter plusieurs commandes d'affilée en les séparant par des points-virgules : `touch fichier.txt; rm fichier.txt` .
- La commande `sleep` permet de faire une pause entre deux commandes exécutées

d'affilée.

- `crontab` permet de programmer des commandes pour une exécution régulière. Par exemple : tous les jours à 18 h 30, tous les lundis et mardis à 12 h, tous les 5 du mois, etc. On modifie la programmation avec `crontab -e` .

Que pensez-vous de ce cours ?

J'AI TERMINÉ CE CHAPITRE ET JE PASSE AU SUIVANT

EXÉCUTER DES PROGRAMMES EN ARRIÈRE-PLAN

QUIZ : QUIZ 3

Le professeur

Mathieu Nebra

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

Découvrez aussi ce cours en...

Livre PDF

Nous rejoindre

Devenir mentor

Devenir coach carrière

AIDE

FAQ

POUR LES ENTREPRISES

Former et recruter

EN PLUS

Boutique

Mentions légales

Conditions générales d'utilisation

Politique de protection des données personnelles

Cookies

Accessibilité

Français

Télécharger dans l'App Store