Cours UNIX

Chapitre 3

Commandes de base

L'utilisation de ce document ne remplace pas la consultation du manuel UNIX (voir commande `man`), d'où il est tiré (c'est en fait un condensé de la version française du manuel Linux)

Exécution de commandes

➔ Lancement d'un commande, arguments

Il existe trois types de commandes exécutables depuis un shell (voir Chapitre 2): les commandes internes au shell, les commandes externes (sous forme de programmes exécutables) et les scripts.

Une exécution de commande se compose du nom de la commande suivi éventuellement des paramètres séparés par des espaces. Les paramètres qui commencent par un tiret sont généralement des options spécifiques à la commande.

Exemple:

```
# tar -ztvf man-fr-0.9.tar.gz
```

(le # représente dans tout ce document l'invite de commande, le texte en gras les commandes tapées par l'utilisateur).

➔ Enchaînement des commandes

Plusieurs commandes peuvent êtres enchaînées sur une même ligne en les séparant par un point-virgule.

Exemple:

```
# clear; pwd; echo Coucou;
```

➔ Lancement en arrière plan

Une commande peut être exécutée en arrière plan en plaçant l'opérateur & en fin de commande (après les paramètres).

Exemple:

```
# sleep 10 &
```

Manipulation des répertoires

➔ Généralités

Sur UNIX, les noms des répertoires sont séparés par le caractère / (division). Le répertoire courant est appelé . (point), le répertoire parent .. (deux point), le répertoire racine / (division) et le répertoire home (le répertoire de l'utilisateur) ~ (tilde).

➔ ls

Afficher le contenu d'un répertoire.

Syntaxe:

```
ls [options] [fichier...]
```

La commande ls affiche tout d'abord l'ensemble de ses arguments fichier autres que des répertoires, puis affiche l'ensemble des fichiers contenus dans chaque répertoire indiqué. Si aucun argument (autre qu'une option) n'est fourni, le contenu du répertoire en cours (`.') est affiché.

Options couramment utilisées:

- -a Afficher tous les fichiers, y compris les fichiers cachés.
- -l En plus du nom, afficher le type du fichier, les permissions d'accès, le nombre de liens physiques, le nom du propriétaire et du groupe, la taille en octets, et la date de dernière modification.
- -L Afficher les informations concernant les fichiers pointés par les liens symboliques et non pas celles concernant les liens eux-mêmes.

Exemple: afficher la description du fichier `/bin/cp` et le contenu du répertoire `/var`.

```
# ls -l /bin/cp /var
-r-xr-xr-x  1 root  wheel  66440 Apr 21 09:05 /bin/cp
/var:
total 32
drwxr-xr-x  2 root    wheel    512 Apr 21 09:02 account
drwxr-xr-x  4 root    wheel    512 May  4 16:56 at
drwxr-x---  2 root    wheel    512 May 10 03:01 backups
...
```

➔ cd

Changer le répertoire courant.

Syntaxe:

```
cd répertoire
```

Note: cette commande est en réalité une commande interne des shells.

Exemple: « aller » à la racine de l'arborescence du système.

```
# cd /
```

➔ mkdir

Créer des répertoires.

Syntaxe:

```
mkdir [options] répertoires...
```

Crée un répertoire correspondant a chacun des noms passés en paramètre.

Options couramment utilisées:

-p Créer les répertoires parents manquants.

Exemple: créer le répertoire `toto` dans le répertoire courant.

```
# ls -l
total 2
-rwx------  1 michelon  users  503 May 14 15:57 burn.sh
# mkdir toto
# ls -l
total 2
-rwx------  1 michelon  users  503 May 14 15:57 burn.sh
drwxr-xr-x  2 michelon  users  512 May 14 15:58 toto
```

➔ rmdir

Supprimer des répertoires vides.

Syntaxe:

```
rmdir [options] répertoires...
```

Supprimer chacun des répertoires passés en paramètres, uniquement si ils sont vides.

Exemple: effaces le répertoire (vide) `toto` qui se trouve dans le répertoire courant.

```
# ls -l
total 2
-rwx------  1 michelon  users  503 May 14 15:57 burn.sh
drwxr-xr-x  2 michelon  users  512 May 14 15:58 toto
# rmdir toto
# ls -l
total 2
-rwx------  1 michelon  users  503 May 14 15:57 burn.sh
```

➔ find

Rechercher des fichiers de façon récursive.

Syntaxe:

```
find [chemins...] [expressions...]
```

`find` parcourt les arborescences de répertoires commençant en chacun des chemins passés en paramètre, en évaluant les expressions fournies pour chaque fichier trouvé.

Tests couramment utilisées dans les expressions:

-name motif	Fichiers dont le nom (sans les répertoires) correspond au motif du shell.
-regex motif	Fichiers correspondants à l'expression régulière motif.
-user utilisateur	Fichiers appartenant à l'utilisateur indiqué (nom ou UID).
-newer fichier	Fichiers modifiés plus récemment que le fichier indiqué.

Actions couramment utilisées dans les expressions:

-exec commande ;	Exécute la commande. La chaîne {} est remplacée par le nom du fichier trouvé.
-print	Affiche le nom complet de chaque fichier trouvé.

Exemple: éditer avec vi tous les fichiers C++ trouvés dans le répertoire de l'utilisateur.

```
# find ~ -name "*.cpp" -exec vi {} \;
```


Manipulation des fichiers

➔ rm

Effacer des fichiers.

Syntaxe:

```
rm [options] fichiers...
```

`rm` efface chaque fichier passé en paramètre. Par défaut, il n'efface pas les répertoires.

Options couramment utilisées:

- -i Interactif: demander à l'utilisateur de confirmer l'effacement de chaque fichier.
- -f Force. Annule -i.
- -r Récursif. Supprimer récursivement le contenu des répertoires. **A utiliser avec précaution !**
- -v Afficher le nom de chaque fichier/répertoire avant de supprimer.

Il est conseillé de créer un alias (voir Chapitre 2, Shells) sur cette commande, de façon à toujours avoir une confirmation (UNIX étant un système multi-utilisateurs, il n'existe pas de commande "undelete" vraiment efficace):

```
# alias rm='rm -i'
```

Exemple: sélectionner les fichiers à effacer dans le répertoire /tmp.

```
# rm -i /tmp/*
```

➔ cp

Copier des fichiers.

Syntaxe:

```
cp [options] fichiers... destination
```

Permet de copier des fichiers et des répertoires. Si le dernier argument correspond à un nom de répertoire, `cp` copie dans ce répertoire chaque fichier indique en conservant le même nom.

Les autorisations d'accès des fichiers et des répertoires crées seront les mêmes que celles des fichiers d'origine masquées avec un ET binaire avec 0777, et modifiées par le umask de l'utilisateur.

Options couramment utilisées:

- -i Interactif. Interroger l'utilisateur avant d'écraser la destination.
- -p Conserver le propriétaire, le groupe, les permissions d'accès et les dates du fichier original.
- -R Récursif. Copier récursivement les répertoires.

Exemple: Faire un miroir du répertoire /etc (et de ses sous-répertoires) dans /tmp.

```
# cp -R -p /etc /tmp
```

➔ mv

Déplacer ou renommer des fichiers.

Syntaxe:

```
mv [options] source... destination
```

Si le dernier argument est le nom d'un répertoire existant, `mv` placera tous les autres fichiers à l'intérieur de ce répertoire, en conservant leurs noms.

Sinon, s'il n'y a que deux fichiers indiqués, il déplacera le premier pour remplacer le second.

Options couramment utilisées:

- -i Interactif. Demander la confirmation pour écraser tout fichier existant.
- -f Force. Annule -i.
- -v Affiche le nom des fichiers avant de les déplacer.

➔ ln

Créer des liens entre fichiers.

Syntaxe:

```
ln [options] source... [destination]
```

`ln` crée des liens entre fichiers. Par défaut il s'agit de liens physiques. Si l'on utilise l'option -s, les liens seront symboliques (logiques).

Un fichier peut avoir plusieurs noms. Un lien physique est simplement une manière de nommer un fichier. Un fichier n'est effacé réellement que lorsque son dernier nom est supprimé.

Un lien symbolique est un petit fichier spécial, qui contient un chemin d'accès vers un fichier ou répertoire. Un lien symbolique ne pointe pas nécessairement vers un fichier existant.

Si l'on n'indique qu'un seul nom de fichier, un lien vers ce fichier est crée dans le répertoire courant. Si le dernier argument indique un répertoire existant, `ln` créera des liens sur chacun des fichiers source indiqués dans ce répertoire.

Par défaut, `ln` ne supprime pas les fichiers ni les liens symboliques existants.

Options couramment utilisées:

-f Forcer l'écrasement du fichier destination s'il existe.

-s Créer des liens symboliques et non pas des liens physiques.

Exemple: donne un nom supplémentaire au fichier .Xresources du répertoire courant.

```
# ls -l -a .X*
-rw-r--r--  1 michelon  users   62 May 10 08:30 .Xresources
# ln .Xresources .Xdefaults
# ls -l -a .X*
-rw-r--r--  1 michelon  users   62 May 10 08:30 .Xresources
-rw-r--r--  2 michelon  users   62 May 10 08:33 .Xdefaults
```

➔ cat

Concaténer des fichiers et les afficher.

Syntaxe:

```
cat [options] fichiers...
```

Affiche sur la sortie standard le contenu de chacun des fichiers passés en paramètre.

Exemple: afficher le contenu de .Xresources.

```
# cat .Xresources
XTerm*background: black
XTerm*foreground: white
*font: 10x20
```

Droits & permissions

➔ *chmod*

Modifier les droits d'accès à un fichier.

Syntaxe:

```
chmod [options] mode fichiers...
```

`chmod` modifie les permissions d'accès de chacun des fichiers passés en paramètre en fonction du paramètre mode.

Le paramètre mode peut être:

- un nombre octal représentant l'état des bits (voir chapitre 1).
- une représentation symbolique sous la forme [ugoa][[+-=][rwxs]:
  - u, g, o, a: change les droits pour l'utilisateur, le groupe, les autres ou tout le monde.
  - -,+,=: enlève ou ajoute un droit sans toucher aux autres ou change tous les droits.
  - r,w,x,s: le(s) droit(s) à changer: lecture, écriture, exécution et set-uid bit.

➔ chown

Modifier le propriétaire d'un fichier.

Syntaxe:

```
chown [options] propriétaire fichiers...
```

`chown` modifie l'utilisateur de chacun des fichiers passés en paramètre. Le propriétaire peut être mentionné par son nom ou par son UID.

Options couramment utilisées:

-v Décrire les changements.

-R Effectuer les changements récursivement.

➔ chgrp

Changer le groupe propriétaire d'un fichier.

Syntaxe:

```
chgrp [options] groupe fichiers...
```


`chgrp` change l'appartenance de chacun des fichiers indiqués pour qu'ils deviennent propriétés du groupe indiqué.

Le groupe peut être mentionné par son nom ou par son GID.

Options couramment utilisées:

-R Effectuer les changements récursivement.

➔ id

Afficher les UIDs et GIDs effectifs et réels.

Syntaxe:

```
id [options] [utilisateur]
```

`id` affiche les informations concernant l'utilisateur indiqué, ou l'utilisateur du processus appelant si aucun utilisateur n'est mentionné.

Par exemple:

```
# id
uid=1001(michelon) gid=1002(users) groups=1002(users), 0(wheel),
1004(webadmin)
```

➔ su

Exécuter un shell ou une commande avec un UID et un GID différents.

Syntaxe:

```
su [options] [utilisateur]
```

`su` permet à un utilisateur de se transformer temporairement en un autre utilisateur. Un shell est exécuté avec les UID, GID effectifs et réels, ainsi que les groupes supplémentaires de l'utilisateur indiqué.

Si aucun nom d'utilisateur n'est mentionné, le nom root, le super-utilisateur, est utilisé par défaut.

Options couramment utilisées:

-c commande Transmet la commande (sur une seule ligne) au shell plutôt que de démarrer un shell interactif.

Exemple: exécuter la commande `burncd` sous l'identité du super utilisateur qui seul à le droit de graver des CDs.

```
# su -c 'burncd -s 16 -e data ./image.iso fixate'
```

Affichage

➔ echo

Afficher une ligne de texte.

Syntaxe:

```
echo [options] [message...]
```

Options couramment utilisées:

-n Ne pas effectuer le saut de ligne final.

➔ more, less

Filtre lecteur de fichier.

`More` est un filtre permettant de se déplacer dans un texte écran par écran. La commande `less` est plus puissante que more car elle permet, entre autres, de ce déplacer librement d'avant en arrière. Généralement utilisée avec le redirecteur pipe |.

Exemple: afficher le log système page par page:

```
# dmesg | less
```


Édition de flux d'entrée / sortie et de fichier

Toutes les commandes de ce chapitre peuvent évidement s'utiliser seule mais elle sont généralement utilisées, comme `more` et `less`, avec le redirecteur pipe |.

➔ tee

Copier l'entrée standard sur la sortie standard et, en même temps, dans un fichier. Permet de voir le résultat d'un commande tout en l'enregistrant dans un fichier.

Exemple: sauver la sortie du log système dans le fichier `monlog` tout en le visualisant page par page.

```
# dmesg | tee monlog | less
...
```

➔ sort

Trier les lignes d'un fichier texte.

`sort` trie, regroupe ou compare toutes les lignes des fichiers indiqués. Par défaut, elle trie dans l'ordre alphanumérique (codes ASCII).

Options couramment utilisées:

-b	Ignorer les blancs en début de ligne.
-f	Ignorer la casse des caractères.
-n	Trier suivant la valeur arithmétique (numérique).
-r	Inverser l'ordre de tri.
-t carac	Utiliser le caractère carac afin de distinguer les champs
+POS1 [-POS2]	Indiquer un ou plusieurs champs à utiliser comme clé de tri plutôt que la ligne entière.

Exemple: afficher le contenu du répertoire courant en triant les fichiers par taille décroissante:

```
# ls -l | sort -r -n +4
```

➔ tail, head

Afficher la première/dernière partie d'un fichier.

`tail` affiche la dernière partie (par défaut : 10 lignes) de chacun des fichiers indiques, `head` affiche la première partie (10 lignes par défaut) de chacun des fichiers indiqués.

Options couramment utilisées:

-n N	Afficher les N premières lignes.

Exemple: afficher les trois plus gros fichiers du répertoire courant.

```
# ls -l | sort -r -n +4 | head -n 3
```

➔ cut

Supprimer une partie de chaque ligne d'un fichier.

Options couramment utilisées:

-f start-[end]	Afficher seulement la ou les colonnes données.
-d carac	Utiliser le caractère carac comme séparateur de colonne.
-c start-[end]	Afficher uniquement les caractères au positions données.

Exemple: voir ci-dessous.

➔ uniq

Éliminer les lignes dupliquées dans un fichier trié.

`Uniq` affiche les lignes uniques d'un fichier préalablement trié, en ne conservant qu'un seul exemplaire de chacune d'elles.

Options couramment utilisées:

-c	Afficher également le nombre d'occurrences de chaque ligne.

Exemple: afficher les différents utilisateurs qui ont au moins un fichier dans le répertoire /tmp, ainsi que le nombre de fichiers qu'ils ont.

```
# ls -l /tmp | cut -f 5 -d ' ' | sort | uniq -c
   1
  23 michelon
  16 root
   2 toto
```


➔ tr

Transposer ou éliminer des caractères.

Syntaxe:

```
tr [options] chaine1 [chaine2]
```

Les arguments chaine1 et (éventuellement) chaine2 décrivent des ensembles ordonnés de caractères.

Exemple d'ensembles: a, aaa, a-z, '[:lower:]', '[:alnum:]', '[:blank:]'

Options couramment utilisées:

-d supprime les caractères chaine1

Exemple: afficher le contenu du fichier `/etc/passwd` en remplaçant les caractère ':' en '@'.

```
# cat /etc/passwd | tr : @
root@*@0@0@Charlie &@/root@/usr/local/bin/bash
toor@*@0@0@Bourne-again Superuser@/root@
...
```

➔ wc

Afficher le nombre d'octets, de mots et de lignes d'un fichier. Par défaut wc affiche les trois valeurs. Les options permettent de n'en afficher que certaines d'entre elles.

Options couramment utilisées:

-c Afficher uniquement le nombre d'octets.

-w Afficher uniquement le nombre de mots.

-l Afficher uniquement le nombre de sauts de lignes.

Exemple:compter le nombre de ligne du fichier /etc/passwd.

```
# cat /etc/passwd | wc -l
      18
```

Environnement de travail

➔ pwd

Afficher le nom du répertoire de travail en cours.

Syntaxe:

```
pwd
```

Exemple:

```
# pwd
/home/michelon
```

➔ date

Afficher ou configurer la date et l'heure du système.

Syntaxe:

```
date [options] [+FORMAT] [MMJJhhmm[[SS]AA][.ss]]
```

`date` sans argument affiche la date et l'heure actuelles. Si un argument commençant par un `+' est indiqué, la date et l'heure sont affichées avec un format contrôlé par cet argument. La structure de la chaîne précisant le format est semblable à celle utilisée avec la fonction C `strftime`. La page man de `date` décrit précisément ce format.

Si l'on fournit un argument ne commençant pas par `+', date le considère comme une heure à utiliser pour configurer l'horloge système.

Exemple: afficher la date sous la forme jour/mois/année.

```
# date +%d/%m/%Y
16/05/2001
```


➔ who, w

Montrer qui est connecté.

Syntaxe:

```
who [options] [am i]
```

who affiche les informations suivantes pour chaque utilisateur connecté :

nom de connexion

terminal

heure de connexion

nom d'hôte distant, ou numéro de terminal X

Sur certain systèmes, la commande w ou encore whodo permet de savoir qui est connecter et que font ces utilisateurs.

Exemple:

```
# w
 2:16PM  up  1:53, 2 users, load averages: 1.00, 1.00, 1.02
USER             TTY      FROM             LOGIN@  IDLE WHAT
michelon         p0       :0               1:03PM     - bash
toto             p1       -                2:16PM     - w
```

➔ passwd

Changer le mot de passe d'un utilisateur.

Syntaxe:

```
passwd [options] [utilisateur]
```

passwd permet de changer de façon interactive le mot de passe d'un utilisateur (de l'utilisateur courant si aucun utilisateur n'est précisé).

gestion des processus

➔ ps

Afficher l'état des processus en cours.

Syntaxe:

```
ps [options]
```

ps présente un cliché instantané des processus en cours. Les options sont complètement différentes suivant le type d'UNIX (Système V, BSD, Linux).

Options couramment utilisées (BSD, Linux):

l	affichage long (plus détaillé).
u	infos utilisateurs.
x	y compris les processus non attachés à un terminal (daemons)
a	y compris les processus des autres utilisateurs
r	seulement les processus en cours d'exécution.

Exemple: une vue de l'ensemble des processus d'un serveur qui n'en a pas beaucoup.

```
# ps ax
  PID  TT  STAT      TIME COMMAND
    0  ??  DLs    0:00.99  (swapper)
    1  ??  ILs    0:00.04 /sbin/init --
    2  ??  DL     0:18.35  (pagedaemon)
    3  ??  DL     0:00.00  (vmdaemon)
    4  ??  DL     0:39.00  (bufdaemon)
    5  ??  DL     3:36.63  (syncer)
  129  ??  Is     0:12.38 syslogd -s
  157  ??  Is     0:18.29 /usr/sbin/cron
  160  ??  Is     0:09.13 /usr/sbin/sshd
29384  ??  S      0:00.24 sshd: backup@ttyp0 (sshd)
29385  p0  Ss     0:00.10 -bash (bash)
29391  p0  R+     0:00.00 ps ax
  193  v1  Is+    0:00.02 /usr/libexec/getty Pc ttyv1
```


➔ kill

Envoyer un signal à un processus.

Syntaxe:

`kill` envoie le signal indiqué aux processus mentionnés (PID seulement). Si on ne précise pas de signal, TERM est envoyé.

```
kill [options] [pid]
```

Options couramment utilisées:

-s signal Indique le signal à envoyer. Celui-ci peut être spécifié par son nom ou par son numéro.

-l Affiche une liste des noms de signaux connus.

Exemple: tuer le processus 243 en lui envoyant un signal KILL.

```
# kill -s KILL 243
```

➔ killall, pkill

Envoyer un signal à des processus indiqués par leurs noms.

Syntaxe:

```
killall [options] [-signal] processus...
```

`killall` envoie un signal à tous les processus en train d'exécuter les commandes mentionnées. Si aucun signal n'est précisé, TERM est envoyé.

Cette commande est remplacée par la commande `pkill` équivalente sur certains UNIX System V, et certains ont les deux commandes, comme SGI IRIX. Attention cependant, car souvent lorsque ces deux commandes sont présentes en même temps, `killall` sert plutôt à tuer TOUS les processus d'un seul coup !

Options couramment utilisées:

-l Affiche une liste des noms de signaux connus.

-v Affiche un compte-rendu de l'émission du signal.

Exemple: demander au processus `inetd` de relire sont fichier de configuration.

```
# killall -HUP inetd
```

Gestion des disques et partitions

➔ mount

Monter un système de fichiers.

Syntaxe:

```
mount [options] périphérique | répertoire
```

La commande `mount` permet d'attacher un système de fichiers sur un périphérique quelconque à l'arborescence du système.

La forme standard de la commande mount est:

```
mount -t type périphérique répertoire
```

Utilisée sans paramètres ni options, mount affiche la liste des périphériques déjà montés, avec leur paramètres de montage.

Options couramment utilisées:

-a monte tous les systèmes de fichiers listés dans /etc/fstab ou /etc/vfstab qui ne sont pas encore montés.

-t spécifie un type de périphérique (type du système de fichier en général).

Exemple: affiche la liste des périphériques montés.

```
# mount
/dev/ad0s3a on / (ufs, asynchronous, local)
/dev/ad0s3f on /usr (ufs, asynchronous, local)
/dev/ad0s3e on /var (ufs, asynchronous, local)
procfs on /proc (procfs, local)
/dev/ad0s1 on /DOS/C (msdos, local)
/dev/ad0s5 on /DOS/D (msdos, local)
linprocfs on /usr/compat/linux/proc (linprocfs, local)
servux:/opt on /opt (nfs)
servux:/opt2 on /opt2 (nfs)
servux:/usr/ports on /usr/ports (nfs)
//SERVUX@SERVNT/PUBLIC on /opt/home/public (smbfs)
```


➔ umount

Démonter des systèmes de fichiers.

Syntaxe:

```
umount [options] périphérique | répertoire
```

La commande `umount` détache le(s) système(s) de fichiers mentionne(s) de la hiérarchie des fichiers.

Options couramment utilisées:

-a Tente de tout démonter.

Exemple: démonter le cdrom /dev/acd0c

```
# umount /dev/acd0c
```

➔ df

Affiche des informations sur l'espace libre et occupé des systèmes de fichiers.

Syntaxe:

```
df [options] [fichier...]
```

Sans arguments, `df` indiquera les quantités correspondant à tous les systèmes de fichiers montés.

Options couramment utilisées:

-h Plus lisible, sur certains systèmes affiche en octets, ko, mo et go au lieu du nombre de blocs.

➔ du

Statistiques sur l'utilisation des répertoires.

Syntaxe:

```
du [options] [répertoires...]
```

`du` affiche la quantité d'espace disque utilisée par chacun des arguments, et pour chaque sous-répertoire des répertoires indiqués en argument.

Options couramment utilisées:

-h Plus lisible, sur certains systèmes affiche en octets, Ko, Mo ou Go au lieu du nombre de blocs.

Exécution différées & programmées

➔ at

Mémoriser un travail à exécuter ultérieurement.

Syntaxe:

```
at [-f fichier] [options] HEURE
```

`at` lit, depuis l'entrée standard, ou depuis un fichier (option -f) des commandes qui seront exécuterées ultérieurement, en utilisant le shell /bin/sh.

At permet d'indiquer l'heure de lancement de manière assez complète. Par exemple:

HHMM , HH:MM (heure, minute)

midnight, noon, teatime, now, tomorrow

MMJJAA, MM/JJ/AA, JJ.MM.AA. (jour, mois, années)

etc.

Options couramment utilisées:

-f fichier Lire la commande à exécuter dans le fichier.

Exemple: exécuter la commande `sleep` dans 10 minutes.

```
# echo sleep 100 | at -m now + 10 minutes
Job 3 will be executed using /bin/sh
```

➔ atq

Lister les travaux programmés avec la commande `at.`

Exemple:

```
# atq
Date                    Owner       Queue    Job#
16:44:00 05/17/01       michelon        c       3
```


➔ atrm (1)

Supprimer des travaux programmés.

Syntaxe:

```
atrm travaux...
```

Exemple:

```
# atq
Date                    Owner       Queue    Job#
16:44:00 05/17/01       michelon        c       3
# atrm 3
```

➔ crontab

Gérer son fichier crontab personnel.

Syntaxe:

```
crontab [options] [-u utilisateur] [fichier]
```

Le démon `cron` permet de lancer des commandes différées et surtout répétitives. `cron` s'éveille toutes les minutes, examine les tables crontab mémorises, et vérifie chaque commande pour savoir s'il doit la lancer dans la minute à venir.

`crontab` est le programme utilisé pour installer, supprimer, ou afficher le contenu des tables permettant de piloter le fonctionnement du démon `cron`. Chaque utilisateur dispose de sa propre table crontab.

Si l'option -u est indiquée, elle permet de préciser le nom de l'utilisateur dont la table crontab doit être manipulée, sinon c'est la table crontab de l'utilisateur courant.

Si un nom de fichier est indiqué, le contenu de ce fichier remplace celui de la table crontab courante.

Options couramment utilisées:

- -l liste la table crontab courante.
- -r supprime la table crontab courante.
- -e lance un éditeur de texte pour éditer la table crontab courante. Utilise la variable d'environnement EDITOR pour choisir l'éditeur à lancer.

La page du manuel sur crontab dans la section 5 contient toutes les informations concernant la syntaxe d'un fichier crontab:

```
# man 5 crontab
```

Aide

➔ apropos, whatis

Rechercher une chaîne de caractère sdans la base de données whatis.

Syntaxe:

```
apropos [mots...]
whatis [mots...]
```

La base de données whatis est créée à partir des pages du manuel UNIX pour faciliter et accélérer la recherche.

Exemple: rechercher les commandes ayant un rapport avec le protocole NTP (Network Time Protocol).

```
# apropos NTP
ntp.conf(5)              - Network Time Protocol (NTP) daemon configuration
file
ntp.keys(5)              - NTP daemon key file format
```

➔ man

Afficher une page du manuel UNIX.

Syntaxe:

```
man [section] page
```


Archivage

➔ gzip, gunzip, zcat, bzip2, bunzip2

Compresser ou décompresser des fichiers.

Syntaxe:

```
gzip|gunzip|zcat [options] [fichiers...]
```

`gzip` réduit la taille des fichiers nommes en utilisant le codage Lempel-Ziv (LZ77).

Quand c'est possible, chaque fichier est remplacé par un autre fichier portant l'extension .gz, en gardant les mêmes modes de permissions, et les mêmes dates de dernier accès et de modification.

`bzip2` et `bunzip2` on la même fonction que gzip et gunzip mais utilisent un algorithme de compression différent qui donne dans la plupart des cas de meilleurs résultats.

➔ tar

Utilitaire de gestion d'archives.

Syntaxe:

```
tar [options] [fichiers...] [répertoires...]
```

`tar` est utilisé pour créer et restaurer des fichiers a partir d'une archive connue sous le nom de *tarfile*. Cette commande est très complète et ne peut être expliquée complètement ici. Voici seulement quelques exemples d'utilisation courante:

Créer une archive du répertoire personnel dans un fichier:

```
# tar -c -v -f fichier.tar $HOME
```

Restaurer l'archive précédente dans le répertoire courant:

```
# tar -x -v -f fichier.tar
```

Lister le contenu de l'archive précédente:

```
# tar -t -v -f fichier.tar
```

