Chapitre 3

Étude
Premières commandes shell sous Linux

Objet du document

Il s'agit ici de manipuler l'interpréteur de commandes. Dans cette partie, nous étudierons les commandes de base qui vous permettront d'utiliser correctement un shell. Le document n'est certes pas exhaustif car le système est extrêmement riche, mais il doit permettre de démarrer.

I man, manuel en lignes

La plupart des commandes acceptent l'option `-help` pour afficher une aide comme `ls -help`, mais vous pouvez aussi consulter les pages de manuel (qui sont plus complètes) en tapant `man` suivi du nom de la commande (ex : `man ls`). Il est possible de réaliser des recherches dans les gros fichiers du manuel comme celui du bash par exemple :

```
man bash
/ici entrer la chaîne (pattern) à chercher en avant
?ici entrer la chaîne (pattern) à chercher en arrière
n pour suivant
N pour précédent
q pour quitter
h pour tout le reste
```

Une option intéressante du man est `-k`. Elle permet d'obtenir les noms des différents manuels contenant le mot-clé passé en paramètre. Ainsi si l'on désire connaître la liste des pages de manuel qui parlent de tcp, on entrera `man -k tcp`.

`man` utilise le programme `less` pour visualiser ces pages . Pour sortir de la consultation des pages manuels, on tapera donc `q` comme quit.

Les pages de manuel font partie des principaux outils utilisés par les administrateurs de GNU/Linux. Il vous faudra obligatoirement vous familiariser avec cette source de documentation. La plupart des pages de manuel sont désormais traduites.

II Commandes agissant sur les répertoires et les fichiers

1. cd, changement de répertoire

La commande `cd` permet de se déplacer dans les répertoires tant que les permissions le permettent.

```
cd  /home : entrer dans le répertoire /home s'il existe.
cd  ..    : remonte au répertoire parent.
cd  /     : remonte à la racine.
cd        : revenir au répertoire par défaut de l'utilisateur, ou encore
cd ~ ou cd \$HOME.
```

2. pwd

Cette commande affiche le répertoire courant.

3. ls

Cette commande permet de lister le contenu d'un répertoire. Beaucoup d'options sont disponibles, parmi elles, certaines sont intéressantes :

```
ls (sans option) : liste les fichiers en plusieurs colonnes.
ls -l : liste des fichiers avec les droits d'accès.
ls -a : liste des tous les fichiers (même ceux commençant par un point).
ls | more : liste écran par écran.
```

4. rm

Permet de détruire un ou plusieurs fichiers ou lien physique du répertoire courant. Attention un fichier supprimé peut très difficilement être récupéré.

```
rm fichier : efface un fichier
rm * : efface tous les fichiers du répertoire.
rm -i fichier1 fichier2 fichier3 : -i demande confirmation de l'effacement de
      chaque fichier.
rm -rf NomRep suppression récursive du répertoire NomRep des fichiers qu'il contient
              et de tous ses sous-répertoires.
```

5. mkdir

Permet de créer un répertoire :

```
mkdir rep
```

6. rmdir

Permet de supprimer un répertoire (à condition qu'il soit vide) :

```
rmdir rep
```

7. cp

Permet de copier un ou plusieurs fichiers :

```
cp NomFichier /home/perso
```

8. mv

Permet de déplacer ou renommer un ou plusieurs fichiers :

```
mv NomFichier /home/perso
```

9. cat

Permet d'afficher sans interruption un ou plusieurs fichiers :

```
cat fichier1 fichier2 (affiche fichier1 et fichier2)
```

10. more

affichage page par page d'un fichier :

11. less

Permet d'afficher un fichier page par page. Les principales commandes sont :

```
q : pour sortir
Entrée : pour passer à la ligne suivante
Espace : pour passer à la page suivante
b : pour remonter à la page précédente
n : pour poursuivre la recherche d'une chaîne
/ suivi d'une chaîne et de la touche Entrée : pour rechercher une chaîne
h : aide
```

Ces commandes peuvent être utilisées lors de la consultation de pages de manuel.

12. diff

Cette commande affiche les différences entre deux fichiers. Cela est parfois utilisé par des utilitaires stockant des modifications incrémentales dans des sources ou textes sous leur contrôle, permettant de retrouver des versions antérieures ou le travail à plusieurs.

13. echo

Cette commande écrit sur la ligne courante les arguments qui lui sont passés :

```
echo arg1 arg2...
```

14. ps

Cette commande permet de lister les processus actifs.

```
ps                liste des processus courant.
ps -u utilisateur       liste des processus appartenant à l'utilisateur user.
ps aux                liste de tous les processus du système.
```

On notera aussi la commande top, qui permet de connaître les processus gourmands en puissance de calcul. Seul dans un premier temps on ne regardera que la première colonne PID, qui est le numéro assigné aux processus. c'est ce numéro qui doit être utilisé pour kill.

15. kill

Cette commande est utilisée pour envoyer un signal à un processus. Elle permet notamment de détruire un processus. Par exemple, si le processus cible est le numéro 546, `kill 546` tente de détruire le processus 546. `kill -9 546` force la destruction du processus 546. Un aspect intéressant de cette commande est la possibilité de détruire un groupe de processus. Par exemple, pour détruire tous les processus vous appartenant, sauf le shell courant, il suffit de donner les commandes suivantes :

```
kill -15 -1 Tente de détruire tous vos processus
kill -9 -1 Force la destruction de tous vos processus
```

La commande `man kill`, donne la liste des signaux transmissibles à un processus. Voir aussi la commande `killall`.

16. alias

Elle permet la création de raccourcis ou de synonymes pour des commandes qui existent par exemple sur d'autres systèmes.

```
alias dir " ls -laF "
```

substitue la commande `dir` à la commande `ls -laF`.

17. passwd

Cette commande permet de changer son mot de passe. En général, huit caractères, la distinction entre les majuscules et les minuscules est très importante. L'ancien mot de passe est toujours demandé ainsi que la confirmation du nouveau sauf si vous êtes root.

III Les commandes avancées

1. find

Cette commande permet de trouver des fichiers depuis une racine spécifiée suivant plusieurs critères (-name, Nom ; -mode, dernière modification…) Par exemple, la commande suivante affichera tous les fichiers ayant un f dans leur nom, et ceci sur tout le disque, car on commence à chercher de la racine :

```
find . / -name '*f*' -print
```

2. chmod

chmod change les droits d'accès d'un fichier ou d'un répertoire si vous en êtes le propriétaire ou root. La syntaxe de chmod est :

```
chmod [ a , u , g ,o ] [ + , - ] [ r , w , x ] [ nom.de.fichier ]
```

l'argument donne les droits : a = (all) à tous les utilisateurs u = (user) au propriétaire g = (group) aux utilisateurs du groupe o = (other) aux autres groupes la valeur + ajoute - enlève les permissions autorisées r = la lecture w = l'écriture x = l'exécution

3. chown

Modifie le propriétaire.

```
chown NouveauProprio NomFichier
```

4. chgrp

Modifie le groupe propriétaire d ?un fichier

```
chgrp NouveauGroupe NomFichier
```

5. ln

Cette commande permet d'ajouter un lien physique ou symbolique sur un fichier.

```
ln Fichier.origine Nom.lien : crée un lien physique
ln -s Fichier.origine Nom.lien  : crée un lien symbolique.
```

Cette fonction est particulièrement intéressante. Elle permet d'avoir un seul fichier physique sur le disque est de le désigner sous plusieurs noms logiques. Cela peut, dans certains cas faciliter les opérations de mises à jour ou de maintenance d'applications.

```
$ ls > toto
$ ln toto titi
$ ln -s toto tata
$ ls -al
$lrwxrwxrwx   1 mlx mlx      4 2005-10-07 13:42 tata -> toto
$-rw-r--r--   2 mlx mlx    284 2005-10-07 13:42 titi
$-rw-r--r--   2 mlx mlx    284 2005-10-07 13:42 toto
```

`tata` est un lien logique sur le fichier `toto`. `titi` est un lien physique sur le fichier `toto`. Le fichier est dupliqué. Le chiffre 2 indique que le fichier `titi` ou `toto` ont deux références physiques disctinctes sur le disque. En temps normal ce nombre est égal à 1, sauf pour les répertoires. On ne peut pas créer de liens physiques sur des répertoires.

6. du

Cette commande permet de connaître l'espace disque utilisé par un répertoire. L'unité dépend du système (parfois 512 octets, parfois 1 Koctet, ou d'autres valeurs). Le paramètre `-s *` est souvent utilisé, il permet d'obtenir la taille de tous les objets du répertoire courant.

7. df

Celle-ci donne des informations sur les disques du système, leurs capacités, l'espace disponible.

8. grep

grep est une commande permettant de trouver rapidement et facilement des motifs (expressions régulières) dans un ou plusieurs fichiers. Typiquement, pour trouver la chaîne `Bonjour` (différente de la chaîne `bonjour` si l'argument `-i` n'est pas spécifié) dans les fichiers du répertoire courant, on écrira

```
grep Bonjour *
```

9. D'autres commandes

L'option `-i` permet d'ignorer la casse, ce qui est parfois bien pratique. Il existe de très nombreuses autres commandes, dont certaines sont très fréquemment utilisées (`tr`, `cut`, `sort`, `cat`, `xargs`, `tail`, `head`...). Vous les trouverez dans les paquets coreutils et findutils

IV Redirections et tubes

1. Les fichiers standards stdin, stdout et stderr

Pour chaque terminal, il existe trois fichiers spéciaux :

L'entrée standard (stdin) c'est dans ce fichier que sont lues les données des différents processus s'exécutant sur ce terminal. c'est normalement le clavier.

La sortie standard (stdout) c'est dans ce fichier que sont écrits les résultats des commandes. C'est normalement l'écran.

La sortie erreur standard (stderr) c'est dans ce fichier que seront écrits les différents messages d'erreur d'une commande. C'est, aussi, normalement l'écran.

Le schéma normal d'exécution d'une commande sous Linux est d'ouvrir en lecture le fichier stdin, c'est à dire de prendre en compte tous les caractères qui seront tapés au niveau du clavier durant l'exécution de la commande, d'écrire ses résultats dans le fichier stdout, c'est à dire l'écran, et de placer les éventuels messages d'erreurs dans le fichier stderr.

2. Redirection du fichier standard de sortie stdout

Cela s'obtient en utilisant le symbole > :

```
commande > nom de fichier
Exemple: ls>liste
```

Le contenu du répertoire n'apparaîtra pas à l'écran mais sera écrit dans le fichier liste. Si ce fichier n'existe pas, il sera créé. S'il existe, son contenu antérieur sera perdu. Pour que cela ne soit pas le cas il faut utiliser >> à la place de >. Remarque : cette commande peut permettre notamment (en Bourne Shell uniquement et en bash) de créer des fichiers vides :

```
> fic
```

3. Redirection du fichier standard d'entrée (stdin)

Cela s'obtient en utilisant le symbole < :

```
commande < nom de fichier
```

Cette redirection ne fonctionne qu'avec des commandes qui peuvent travailler directement sur l'entrée standard. Il y a beaucoup de commandes Linux (on les appelle des filtres) qui fonctionnent comme cela, en fait la plupart des commandes qui travaillent sur des fichiers autorisent également qu'on les utilise sans argument. Dans ce cas elles rendent la main à l'utilisateur et c'est l'entrée standard (par défaut le clavier) qui sera lue. Par exemple :

```
mail jean <lettre
```

D'ordinaire la commande `mail` rend la main à l'utilisateur afin qu'il puisse taper sa lettre, ici `mail` travaillera directement avec le fichier `lettre` sans intervention de l'utilisateur.

4. L'opérateur < <

Il permet la lecture sur l'entrée standard jusqu'à ce que qu'un mot soit rencontré. Cela permet d'automatiser dans des scripts des traitements qui demandent normalement une interaction avec le clavier.

Par exemple :

```
$ sh << EOF
> ls
> w
> ls | wc -l
> EOF
```

Les commandes sont lues, ici sur l'entrée standard et ne sont exécutées qu'une fois la chaîne `EOF` rencontrée.

5. Redirection du fichier standard d'erreur

Il y a quelques petites différences en fonctions du Shell que l'on utilise. En Bourne Shell et en bash, on utilisera le symbole > précédé d'un chiffre 2 qui symbolise la deuxième sortie standard :

```
commande 2> nom de fichier
```

Exemple :

```
cat toto 2> erreur
```

Si le fichier `toto` n'existe pas aucune erreur n'apparaîtra sur l'écran, mais par contre le message d'erreur sera écrit dans le fichier erreur. Il est possible également d'utiliser `2>>` pour compléter le fichier sans perdre les informations qui s'y trouvaient.

6. Enchaînement des processus

a. Enchaînement simple

Le but est simplement de faire exécuter plusieurs commandes successivement de façon totalement indépendante. Cela s'obtient en séparant les commandes les unes des autres par le symbole `;` :

```
commande;commande1;commmande2;...
```

Exemple :

```
ls ; who am i ; echo salut
```

Ces trois commandes vont s'exécuter les unes après les autres pour donner sur l'écran :

```
Mon Mar 28 21:04:43  1998
thierry     tty9     Mar 28 20:55
salut
```

ou encore :

```
commande && commande1 && commmande2;.....
```

Une commande n'est exécutée que si la précédente s'est bien déroulée.

b. Les « pipes » ou tubes

Cette fois, le but est de faire exécuter des processus de manière concurrente (en parallèle) et communiquant entre eux par l'intermédiaire de zones mémoires temporaires c'est à dire prises sur la mémoire centrale. Les paragraphes précédents ont montré qu'il était possible de rediriger la sortie ou l'entrée d'une commande. On peut donc imaginer des traitements « à la chaîne » qui utiliseraient plusieurs commandes.

Par exemple, si on veut afficher la liste des fichiers se trouvant dans le répertoire courant, trié par nom et obtenir une visualisation page par page :

```
ls > fic1
sort fic1 > fic2
more fic2
```

Cela présente plusieurs inconvénients.

1. création de fichiers intermédiaires qu'il faudra détruire,
2. temps d'exécution très lent dû aux entrées / sorties disque à effectuer.

La notion de pipe ou tube va résoudre ces problèmes, un tube est symbolisé par le symbole `|` :

```
ls | sort | more
```

La sortie standard d'une commande devient le flux de l'entrée standard de la commande qui suit le pipe. Le système Linux assure la synchronisation de l'ensemble :

1. les trois processus correspondant aux trois commandes s'exécutent en parallèle,
2. chaque processus qui lit dans un tube se bloque lorsque le tube est vide,
3. chaque processus qui écrit dans un tube s'arrête lorsque le tube est plein.

Les tubes vont permettre, en enchaînant des commandes de bases entre elles, de réaliser des traitements économisant le recours à la programmation. On pourra ainsi successivement sélectionner certaines lignes d'un fichier (selon un motif défini), réaliser un tri sur le fichier, supprimer éventuellement les doublons, ne garder que les 20 premiers, . . .

Exemple :

```
grep http log | sort -n + 2 | uniq | head - 20
```

Cet exemple permet de traiter un fichier de journalisation, de ne sélectionner que les lignes comportant le mot `http`, de trier ce fichier selon la deuxième colonne (en numérique), de supprimer les doublons et enfin d'en extraire les 20 premières lignes.