Commandes Linux

Les commandes Linux en Bourne-shell et Bash ont une syntaxe commune. Cependant, chaque interpréteur de commandes a ses particularités propres. Dans ce chapitre, nous présentons uniquement la commande propre à l'interpréteur de commandes Bash.

7.1 LA COMMANDE LINUX

7.1.1 Syntaxe générale des commandes Linux

```
commande [ ± option… ] [ paramètre… ]
```

Une commande se compose d'un code mnémonique en minuscules suivi ou non d'arguments séparés par au moins un espace.

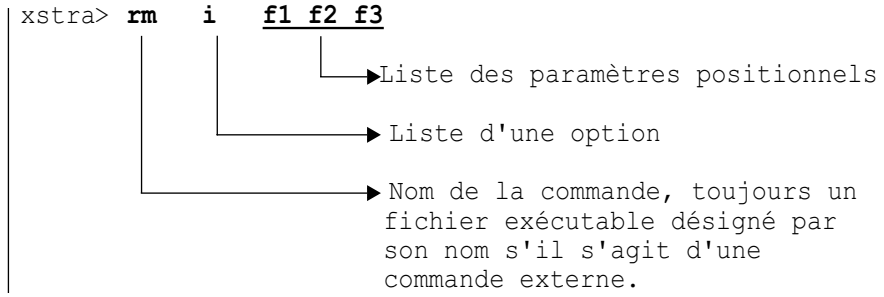
On distingue deux types d'arguments :

- les options, presque toujours précédées d'un signe ( ) et quelquefois d'un signe (+). Certaines options ont un paramètre propre,
- les arguments positionnels ou paramètres, qui sont en général des noms de fichiers ou de répertoires.

En général, la liste des options précède celle des paramètres.

© Dunod – La photocopie non autorisée est un délit.

Exemple



7.1.2 Conventions utilisées pour la syntaxe des commandes

1) Commande et options	Le texte de la commande ainsi que celui des options est à reprendre tel quel. Dans le manuel Linux, ce texte est représenté en caractères gras ou soulignés.
2) Paramètre	Un nom symbolique décrit l'usage de chaque paramètre. Ce nom symbolique doit être remplacé par la valeur effective que l'usager désire donner au paramètre.
3) […]	Le paramètre ou l'option entre crochets est optionnel.
4) Paramètre…	Les trois points qui suivent un nom symbolique d'un paramètre désignent une liste de paramètres.

Exemple

```
cat [ ] [ benstuv] fichier...
```

7.1.3 La ligne de commandes séquentielles

Il est possible de taper plusieurs commandes sur la même ligne en les séparant par des points-virgules `;`. Les commandes sont exécutées séquentiellement, de façon totalement indépendante, la première n'influençant pas la seconde et ainsi de suite.

Exemple

```
xstra> pwd ; who ; ls
/home/xstra
xstra   ttyp0   Mar    15 10:32
xstra1  co      Mar    15 09:12
bin projet1 projet2 essai
xstra>
```

Cette commande affiche le répertoire courant de l'utilisateur, donne la liste des utilisateurs connectés, puis liste les fichiers du répertoire courant.

7.1.4 La commande sur plus d'une ligne

Il est possible de taper une commande sur plusieurs lignes. Pour cela les lignes de commandes, sauf la dernière, doivent se terminer par la suite de touches *\<return>*.

Exemple

```
xstra> ls  l /home/xstra/develop\<return>
/essai.f <return>
 rwx r       1 xstra staff 258 Jan 15 16:42 essai.f
xstra>
```

Cette commande est équivalente à :

```
xstra> ls  l /home/xstra/develop/essai.f <return>
```

Elle liste les droits d'accès du fichier *essai.f* qui se trouve dans le répertoire */home/xstra/develop*.

7.1.5 Les séparateurs conditionnels de commandes

Il est possible de contrôler la séquence d'exécution de commandes en utilisant des séparateurs conditionnels.

Le séparateur **&&** permet d'exécuter la commande qui le suit si et seulement si la commande qui le précède a été exécutée sans erreur (code retour du processus nul).

Le séparateur **||** permet d'exécuter la commande qui le suit si et seulement si la commande qui le précède a été exécutée avec erreur (code retour du processus différent de 0).

Exemple

Suppression des fichiers si la commande *cd projet1* a été correctement exécutée.

```
xstra> cd projet1 && rm *
```

Exemple

Si le répertoire *projet1* n'existe pas, alors il sera créé par la commande *mkdir*.

```
xstra> cd projet1 || mkdir projet1
bash: cd: projet1: No such file or directory
xstra> ls
projet1
xstra>
```

© Dunod – La photocopie non autorisée est un délit.

7.2 LA REDIRECTION DES ENTRÉES-SORTIES

7.2.1 Le principe de redirection

On appelle processus, ou tâche, l'exécution d'un programme exécutable. Au lancement de chaque processus, l'interpréteur de commandes ouvre d'office une entrée standard (par défaut le clavier), une sortie standard (par défaut l'écran) et la sortie d'erreur standard (par défaut l'écran) (Fig. 7.1).

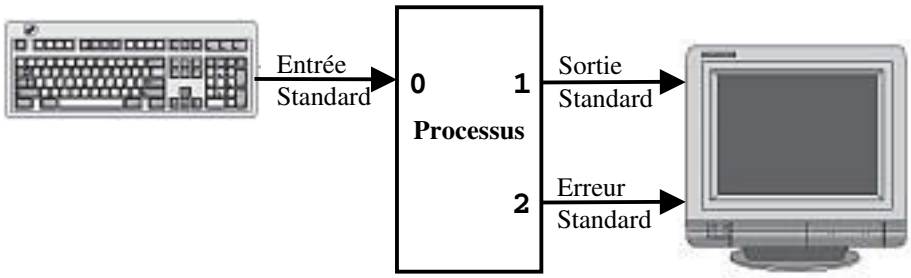


FIGURE 7.1. ENTRÉES/SORTIES STANDARD D'UN PROCESSUS.

Ces entrées-sorties standard peuvent être redirigées vers un fichier, un tube, un périphérique. La redirection de la sortie standard consiste à renvoyer le texte qui apparaît à l'écran vers un fichier (Fig. 7.2). Aucune information n'apparaîtra à l'écran, hormis celles qui transitent par la sortie d'erreur standard.

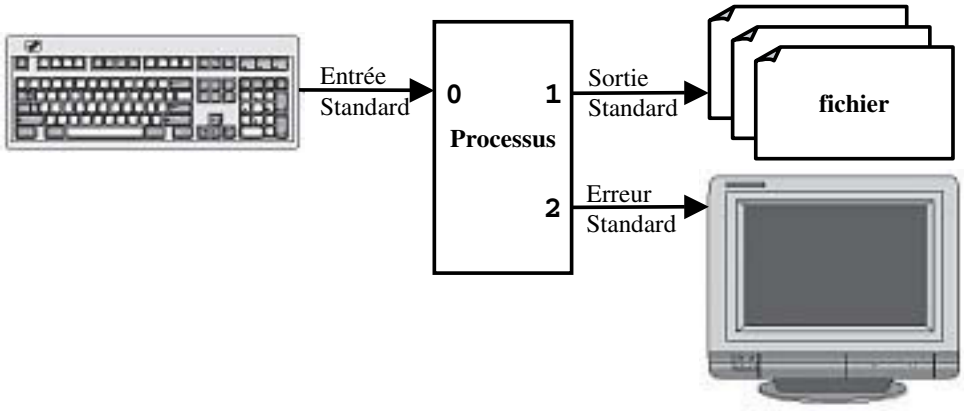


FIGURE 7.2. REDIRECTION VERS LE FICHIER *FS* DE LA SORTIE STANDARD D'UN PROCESSUS.

Il est naturellement possible de rediriger toutes les entrées-sorties standard d'un processus. Par conséquent, le processus recherchera les informations dont il a besoin dans un fichier et non plus au clavier. Il écrira dans des fichiers ce qui devait apparaître à l'écran (Fig. 7.3).

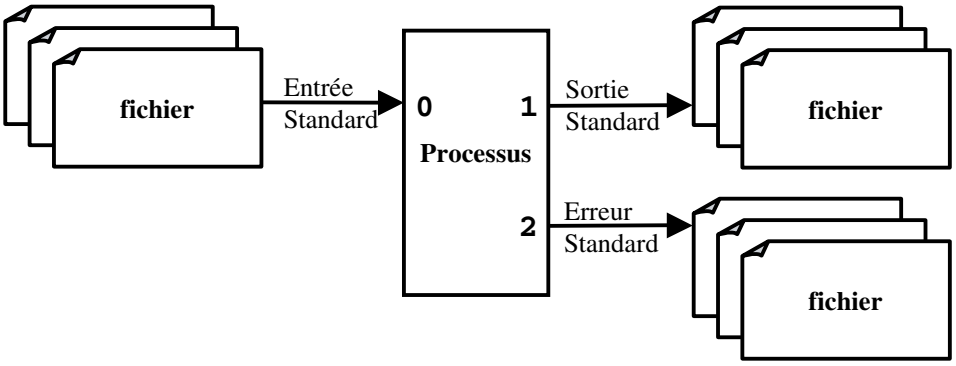


FIGURE 7.3. REDIRECTION VERS DES FICHIERS DE TOUTES LES ENTRÉES/SORTIES STANDARD D'UN PROCESSUS.

La redirection des sorties peut être réalisée par effacement et création du fichier ou par ajout à la fin du fichier si ce dernier existe. Dans le cas contraire, un nouveau fichier sera créé. Dans le cas de la redirection de l'entrée, il est évident que le fichier doit exister. Le tableau 7.1 résume les différentes redirections.

	Le fichier existe-t-il?	
	OUI	NON
Redirection de l'entrée standard	lit le fichier	erreur
Redirection de la sortie standard et de la sortie erreur standard	effacement et création du fichier	création du fichier
Concaténation de redirection de la sortie et de l'erreur standard	ajoute à la fin du fichier	création du fichier

TABLEAU 7.1. RÉSUMÉ DES REDIRECTIONS.

Le caractère **<** suivi du nom d'un fichier indique la redirection de l'entrée standard à partir de ce fichier :

`<fe` Définition de `fe` comme fichier d'entrée standard.

Le caractère **>** suivi du nom d'un fichier indique la redirection de la sortie standard vers ce fichier :

© Dunod – La photocopie non autorisée est un délit.

`>fs` Redirection de la sortie standard vers le fichier `fs` ; le fichier `fs` est créé ou écrasé s'il existe déjà et si la variable booléenne `noclobber` n'est pas initialisée.

Si on double le caractère >, l'information ou la redirection sera ajoutée au fichier `fs`.

`>>fs` La sortie standard rallonge le fichier `fs`.

Les caractères **`2>`** suivis du nom d'un fichier indiquent la redirection de la sortie d'erreur standard vers ce fichier :

`2>erfs` Redirection de la sortie d'erreur standard vers le fichier `erfs`.

Les caractères **`2>>`** suivis du nom d'un fichier indiquent que la redirection de la sortie d'erreur standard sera ajoutée au fichier :

`2>>erfs` La sortie d'erreur standard est ajoutée au fichier `erfs`.

Si l'on souhaite rediriger la sortie standard et la sortie d'erreur standard, la syntaxe sera :

`commande >fs 2>erfs` Dirige la sortie standard vers `fs` et la sortie d'erreur vers `erfs`.

Les caractères **`&>`** suivis du nom d'un fichier indiquent la redirection de la sortie standard et de la sortie d'erreur standard vers ce fichier :

`&>erfs` Redirection des sorties standard et de la sortie d'erreur standard vers le fichier `erfs`.

Exemples

```
xstra> ls >temp
    § Range, dans le fichier temp, la liste
    § des fichiers du répertoire courant.
    § Le fichier temp est créé s'il n'existe
    § pas ou écrasé s'il existe.
xstra> ls >>f.rep
    § Ajoute à f.rep la liste des fichiers
    § du répertoire courant.
xstra> ls /toto 2>temp
    § Range, dans le fichier temp,
    § les messages d'erreurs et affiche
    § à l'écran la liste des fichiers.
    § /toto  n'existe pas.
xstra> ls /toto * >temp 2>ertemp
```

```
§ Range dans le fichier temp, la liste
§ des noms de fichiers du répertoire courant
§ et dans le fichier ertemp,
§ le message d'erreur /toto n'existe pas.
```

7.2.2 La commande cat et les redirections

La commande *cat* est une commande multi-usage qui permet d'afficher, de créer, de copier et de concaténer des fichiers. Elle utilise pleinement les mécanismes de redirection. Elle lit l'entrée standard si aucun fichier n'est spécifié. Ce qui est lu est affiché sur la sortie standard.

a) Lecture sur clavier et écriture sur écran

```
xstra> cat
Toute la musique que j'aime,
Toute la musique que j'aime,
<ctrl d>
xstra>
```

Le texte « Toute la musique que j'aime, » est lu du clavier et est affiché à l'écran. La combinaison de touches *<ctrl d>* interrompt la saisie.

b) Copie d'un fichier

```
xstra> cat f1 >f2  § première possibilité
xstra> cat <f1 >f2 § deuxième possibilité
xstra>
```

Le fichier *f1* est copié dans *f2*.

c) Concaténation des fichiers

```
xstra> cat f1 f2 f3 >f123
xstra>
```

Le fichier *f123* contiendra la concaténation des fichiers *f1*, *f2* et *f3*, dans cet ordre.

d) Ajout d'un fichier

```
xstra> cat f1 >>f2
xstra>
```

Le fichier *f1* est concaténé à la suite du fichier *f2*. *f2* est créé s'il n'existe pas.

e) Création d'un fichier par saisie au clavier

```
xstra> cat >f1
bonjour
```

© Dunod – La photocopie non autorisée est un délit.

```
.....
<ctrl d>
xstra>
```

Le fichier *f1* est créé. Il contiendra le texte saisi jusqu'à interruption par la combinaison de touches *<ctrl d>*.

f) Création d'un fichier avec condition de saisie

```
xstra> cat <<EOT >f1
Merci de votre visite
A bientot
EOT
xstra>
```

Le fichier *f1* est créé par saisie de texte au clavier, jusqu'à la saisie en début de ligne d'une chaîne de caractères prédéfinie (*EOT* dans ce cas). La chaîne de caractères *<<texte* redirige l'entrée standard jusqu'à apparition du mot *texte*. Cette formulation est très employée pour la création d'un fichier dans un fichier de commandes (script).

7.3 LES TUBES DE COMMUNICATION (PIPE) ET LES FILTRES

7.3.1 Les tubes

Un **tube** (**pipe** en anglais) est un flot de données qui permet de relier la sortie standard d'une commande à l'entrée standard d'une autre commande sans passer par un fichier temporaire (Fig. 7.4).

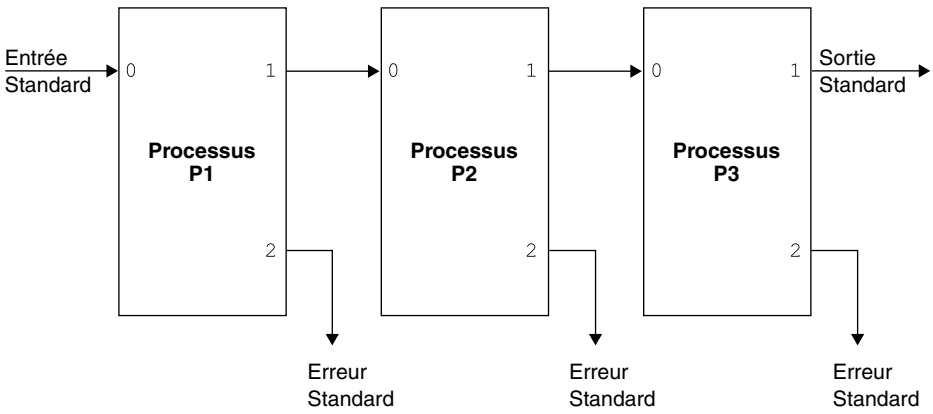


FIGURE 7.4. UN TUBE (PIPE).

Dans une ligne de commandes, le tube est formalisé par la barre verticale |, que l'on place entre deux commandes :

```
P1 | P2 | P3
```

Exemple 1

Affichage page par page du contenu du répertoire courant :

```
xstra> ls  l | less
```

Dans cet exemple, le résultat de la commande `ls  l` n'apparaît pas à l'écran : la sortie standard est redirigée vers l'entrée standard de la commande `less` qui, quant à elle, affichera son entrée standard page par page. La commande `less`, contrairement à la commande `more`, permet à l'aide des touches claviers ↓, ↑, <page-up> et <page-down> de monter et descendre dans le flot de données obtenu dans un tube.

Exemple 2

Affichage page par page du contenu du répertoire courant dont les protections sont `rwxr xr x :`

```
xstra> ls  l | grep "rwxr xr x"| less
```

Pour obtenir le même résultat en utilisant le mécanisme de redirection des entrées-sorties, il faut écrire :

```
xstra> ls  l >temp1; \<return>
grep "rwxr xr x" <temp1 >temp2 \<return>
; less temp2; rm temp1 temp2
```

Cette solution est non seulement plus lourde mais aussi plus lente. Les tubes ont deux qualités supplémentaires :

- Toutes les commandes liées par le tube s'exécutent en parallèle. C'est le système qui réalise la synchronisation entre les processus émetteurs et récepteurs,
- Il n'y a pas de limite de taille pour le flot de données qui transite dans le tube (il n'y a pas création de fichier temporaire).

Exemple 3

```
xstra> who | wc  l
```

Indique le nombre de personnes connectées au système.

```
xstra> ls | wc  w
```

Indique le nombre de fichiers dans le répertoire courant.

© Dunod – La photocopie non autorisée est un délit.

7.3.2 Les filtres

Dans les exemples précédents, nous voyons apparaître, à travers les commandes *wc*, *less* et *grep*, une famille particulièrement importante de commandes Linux : les **filtres.** Un filtre est une commande qui lit les données sur l'entrée standard, les traite et les écrit sur la sortie standard.

Le concept de tube, avec sa simplicité, devient un outil très puissant dans Linux qui propose un choix très vaste de filtres. Les filtres les plus utilisés sont les suivants :

grep recherche les occurrences d'une chaîne de caractères.

egrep une extension de *grep* (voir chapitre 14).

wc compte le nombre de caractères (ou octets), mots et lignes.

less affiche son entrée standard page par page.

dd filtre de conversion.

sed éditeur de flot : il applique des commandes de l'éditeur *ed* sur l'entrée standard et envoie le résultat sur la sortie standard (voir chapitre 14).

awk petit langage de manipulation de texte (voir chapitre 14).

sort filtre de tri.

Attention

Le caractère < n'est pas obligatoire pour rediriger l'entrée standard d'un filtre sur un fichier :

```
less /etc/passwd    équivaut à less </etc/passwd
cat  /etc/group     équivaut à cat  </etc/group
```

La puissance des filtres et des tubes est très bien explicitée dans l'exemple suivant :

Compter le nombre d'utilisateurs de la machine dont le login shell est le Bash.

```
xstra> cat /etc/passwd | grep /bin/bash | wc -l
```

La commande *cat* liste le contenu du fichier */etc/passwd* décrivant tous les utilisateurs. Le filtre *grep* ne conserve que les lignes contenant la chaîne */bin/bash*. Le filtre *wc -l* compte le nombre de lignes passées par le filtre *grep*.

7.3.3 La commande xargs

Le mécanisme de tube est très pratique pour assembler entre elles des commandes Linux. Cependant, beaucoup de commandes ne lisent pas dans leur entrée standard : *ls, rm, cp, ln, mv* et bien d'autres ne sont pas des filtres, mais traitent leurs arguments :

rm alpha beta gamma est une commande correcte, alors que

ls | rm n'a AUCUN sens.

Ces commandes ne peuvent donc pas apparaître dans un tube, sauf au début. La commande *xargs* permet de lever cette restriction en construisant une liste de paramètres à partir de l'entrée standard :

cmd1 | xargs cmd2 signifie :

lancer *cmd2* en lui passant en paramètres ce qui arrive dans l'entrée standard de *xargs*, donc la sortie standard de *cmd1*.

Exemple

Le fichier *listf* contient une liste de noms de fichiers, à raison d'un nom par ligne.

```
xstra> cat listf
/etc/passwd
/etc/group
/bin/bash
/usr/bin/id
xstra> cat listf | ls -i  § Erreur grossière
298014 bin      35364 Desktop   723098 Mail
608698 rep1    494443 listf
§ ls  i n'a rien lu dans son entrée standard
xstra> cat listf | xargs ls -i  § Correct
523314 /bin/bash     232382 /etc/group
231586 /etc/passwd   327284 /usr/bin/id
```

7.4 TÂCHES EN ARRIÈRE-PLAN

Linux attend la fin de l'exécution de la commande en cours d'exécution avant de permettre à l'utilisateur de relancer une nouvelle commande. Lorsqu'une commande ne nécessite pas de dialogue avec l'utilisateur et que sa durée d'exécution est importante, il est possible de l'exécuter en arrière-plan en ajoutant le caractère spécial **&** à la fin de la commande. Le système Linux lance alors la commande et redonne immédiatement la main à l'utilisateur pour d'autres travaux.

Le lancement de commandes en arrière-plan permet à un seul utilisateur de lancer plusieurs tâches à partir d'un même terminal. Cela correspond à l'aspect multitâche du système d'exploitation Linux.

Ces processus en arrière-plan ne peuvent plus être interrompus directement à partir de la console à l'aide de la touche *<ctrl c>*. Pour les interrompre il faut :

© Dunod – La photocopie non autorisée est un délit.

- soit sortir de sa session, dans ce cas tous les processus attachés à la session seront interrompus,
- soit rechercher le numéro du processus et lui envoyer un signal à l'aide de la commande *kill*, (voir aussi paragraphe 12.5).

Il est possible d'éviter qu'un processus en arrière-plan soit interrompu en quittant sa session. Il faut pour cela utiliser la commande *nohup* :

```
nohup commande <entrée >sortie &
```

Ce mécanisme est intéressant si l'on souhaite exécuter un programme de très longue durée et libérer la console de lancement.

Action	État initial du processus	État final du processus
| xstra> **commande**		processus en avant-plan
| xstra> **commande &**		processus en arrière-plan
| xstra> **fg %numéro_job**	processus en arrière-plan	processus en avant-plan
| **<ctrl z>**	processus en avant-plan	processus stoppé (suspendu)
| xstra> **fg %numéro_job**	processus stoppé	processus en avant-plan
| xstra> **bg %numéro_job**	processus stoppé	processus en arrière-plan
• **kill –STOP %numéro_job** • soit tentative lecture sur le terminal • soit tentative d'écriture sur le terminal (il faut avoir positionné stty tostop)	processus en arrière-plan	processus stoppé

Tableau 7.2. Récapitulation de la gestion des processus en arrière-plan.

Lors du lancement d'une commande en arrière-plan, l'interpréteur de commandes Bash attribue un numéro croissant pour chacune des commandes lancées : le *numéro job*. La liste des travaux en arrière-plan, avec leur numéro, est obtenue à l'aide de la commande *jobs*. Cette commande indique si le processus en arrière-plan est en cours d'exécution (*running*) ou suspendu (*stopped*).

Il est possible de ramener un processus en avant-plan, en tapant la commande *fg %numéro job*. Un processus en avant-plan peut être suspendu par l'envoi de l'ordre de suspension, si le terminal le permet (*stty isig*). Il est possible de déterminer la touche qui envoie l'ordre (*stty susp <ctrl z>*).

Un processus suspendu peut être réactivé à l'aide de la commande *bg %numéro job*. Un processus peut être supprimé à l'aide de la fonction *kill %numéro job*. Le paragraphe 12.5 présente ces mécanismes.

Le tableau 7.2 résume ces différentes possibilités.

7.5 LA SUBSTITUTION DE COMMANDE

La substitution de commande ou backquoting permet d'utiliser le résultat d'une commande comme argument d'une autre commande. Pour utiliser cette fonctionnalité, il faut entourer la commande soit d'accents graves, ou « backquotes » (*`commande`*), soit *$(commande)*. La forme *$(commande)* est plus récente et doit être préférée.

La commande placée entre *$(cmd)* est exécutée en premier, avant l'exécution de la ligne de commandes dont elle fait partie. Son résultat, c'est-à-dire la sortie standard, est entièrement intégré à la ligne de commandes en remplacement de la commande « backquotée ». La ligne de commandes est alors exécutée avec ces nouveaux arguments.

Exemple 1

La commande *echo* affiche à l'écran la chaîne de caractères qui la suit.

```
xstra> echo pwd
pwd
xstra> echo 'pwd'
pwd
xstra> echo `pwd`  § Ancien format
/home/xstra
xstra> echo $(pwd)
/home/xstra
xstra>
```

Dans ce dernier cas, la commande *pwd* qui indique le répertoire courant est exécutée. Son résultat devient l'argument de la commande *echo* (*echo /home/xstra*).

Exemple 2

```
xstra> less $(grep  l 'toto' *)
```

Cette commande permet de visualiser le contenu des fichiers du répertoire courant contenant au moins une fois la chaîne de caractères *toto*.

Exemple 3

```
xstra> echo il y a $(who | wc -l) utilisateurs connectes
```

© Dunod – La photocopie non autorisée est un délit.

Cette commande nous indique le nombre d'utilisateurs connectés.

Exemple 4

```
xstra> grep  n "motif" $(find .  type f -print)
```

Cette commande permet de rechercher à partir du répertoire courant et récursivement dans tous les sous-répertoires, les fichiers contenant la chaîne de caractères *motif*.

Exemple 5

```
xstra> rm  i $(find .  mtime +20 -print)
```

Cette commande permet de supprimer les fichiers n'ayant pas été modifiés depuis plus de 20 jours. Elle est équivalente à :

```
xstra> find .  mtime +20  exec rm  i {} \;
```

7.6 LES COMMANDES GROUPÉES

La commande groupée est une succession de commandes séparées par le caractère **;** et considérées comme un ensemble. Cet ensemble, repéré par des parenthèses (…), est exécuté par un nouveau processus shell. Les commandes seront exécutées séquentiellement, sans influence les unes sur les autres.

Le résultat d'une commande groupée est cependant différent de celui qu'auraient les mêmes commandes réalisées séquentiellement.

Exemple

Suppression du fichier *temporaire* dans le répertoire *projet1*.

```
xstra> cd
xstra> (cd projet1; rm temporaire)
xstra> pwd
/home/xstra
 § Le répertoire courant est toujours /home/xstra
```

Même action que la commande groupée mais le répertoire courant sera *projet1*.

```
xstra> cd projet1; rm temporaire
xstra> pwd
/home/xstra/projet1
xstra>
```

À la différence d'une ligne de commandes séquentielles (paragraphe 7.1.3), Linux interprétera ceci comme une seule commande au niveau de la redirection des entrées-sorties.

Exemple

```
xstra> echo "aujourd'hui"; date; \<return>
echo les personnes suivantes; \<return>
 who; echo sont connectees >fs
xstra>
```

Le fichier *fs* contiendra la chaîne de caractères "*sont connectees*", résultat de la dernière commande à laquelle a été appliquée la redirection.

```
xstra> (echo "aujourd'hui"; date; echo les \<return>
> personnes suivantes; who; echo sont connectees) >fs
```

Dans ce cas, le fichier *fs* contiendra le résultat de chacune des commandes :

```
aujourd'hui
14 fevrier 2000
les personnes suivantes
xavier   console   fev 14 08:30
yannick  tty1      fev 14 08:45
soline   tty2      fev 14 09:00
sont connectees
```

Une autre particularité de la commande groupée est de permettre de lancer tout le groupe de commandes en arrière-plan. En effet, le caractère ***&*** à la fin d'une ligne séquentielle ne lance en arrière-plan que la dernière commande.

7.7 LES CARACTÈRES SPÉCIAUX GÉNÉRATEURS DE NOMS DE FICHIER

Les caractères génériques permettent de désigner un ensemble d'objets. Ils peuvent être utilisés pour la génération d'un ensemble de noms de fichiers. Ils s'appliquent donc aux paramètres des commandes qui désignent des **noms** de fichiers.

Ils peuvent aussi désigner un ensemble de chaînes de caractères. On parle alors d'expressions régulières. Ces expressions s'appliquent aux commandes d'édition (*ex*, *vi*, *sed*, …) ou à des filtres (*grep*, *egrep*, *awk*, …) et permettent la recherche d'une chaîne de caractères dans un fichier. Elles s'appliquent donc aux **contenus** des fichiers.

Il y a des différences d'interprétation entre les caractères spéciaux (ou métacaractères) utilisés dans la génération des noms de fichier (présentation ci-dessous) et ceux utilisés dans les expressions régulières (voir le chapitre 14).

© Dunod – La photocopie non autorisée est un délit.

L'interpréteur de commandes permet de générer une liste de noms de fichier en utilisant les caractères spéciaux suivants :

`*` désigne toutes chaînes de caractères, y compris la chaîne vide.

Exemple

`a*b` désigne tous les noms de fichier commençant par a et finissant par b.

`?` désigne un caractère quelconque.

Exemple

`a?b` désigne tous les noms de fichier commençant par `a` suivi d'un caractère quelconque et finissant par `b`.

`[...]` désigne un caractère quelconque appartenant à la liste donnée entre crochets. Deux caractères séparés par un tiret ( ) définissent une liste de caractères rangés par ordre alphabétique, dont le premier élément est le premier caractère et le dernier élément le dernier caractère.

Exemple

`a[a z0 9A Z]b` désigne tous les noms de fichier commençant par `a` suivi d'un caractère alphanumérique et finissant par `b`.

`[!...]` désigne une liste de caractères à exclure.

Exemple

`a[!a z]b` désigne tous les noms de fichier commençant par `a` suivi d'un caractère autre qu'un caractère alphabétique en minuscule, et finissant par `b`.

Il est possible d'annuler l'interprétation d'un caractère spécial en le faisant précéder de \. Cela est également possible en délimitant par une apostrophe (') ou une double apostrophe (") une chaîne de caractères contenant un ou plusieurs caractères spéciaux. Il existe une différence entre les deux délimiteurs (voir chapitre 8).

En Bash, le caractère ~ (tilde) désigne le répertoire d'accueil défini dans la variable $HOME.

Le caractère ! (point d'exclamation) est aussi utilisé par l'éditeur de commandes du Bash (voir le chapitre 6.5).

7.8 LES CARACTÈRES DE NEUTRALISATION

Nous avons vu que certains caractères spéciaux ont une signification particulière pour l'interpréteur de commandes, comme par exemple : *<,>,*,?,!,...* Il est possible, en plaçant le caractère **** (backslash) devant un tel caractère spécial, de **neutraliser son interprétation** par l'interpréteur de commandes. On peut également neutraliser l'interprétation d'un alias.

Par exemple, le paragraphe 7.1.4 présente le caractère \ placé en fin de ligne comme technique permettant l'écriture d'une commande sur plusieurs lignes. En fait le caractère \ neutralise le caractère qui suit, c'est-à-dire le caractère <return> désignant le caractère de fin de commande.

Exemple

```
xstra> touch f\*1
xstra> ls
f*1 fichier1 fichier2
xstra> rm f\*1
xstra> ls
fichier1 fichier2
xstra>
```

Attention

*rm f*1* supprime tous les fichiers commençant par *f* et se terminant par *1*.

L'utilisation du caractère **** n'est pas pratique si on souhaite neutraliser une chaîne de caractères. Dans ce cas, il est préférable d'encadrer la chaîne à neutraliser par des apostrophes, appelées aussi **quotes** (’). Le seul caractère ne pouvant être neutralisé est la quote elle-même.

Exemple

```
xstra> touch 'f*?1'
xstra> ls
f*?1 fichier1 fichier2
xstra> rm 'f*?1'
xstra> ls
fichier1 fichier2
xstra>
```

Attention

rm f?1* supprime tous les fichiers commençant par *f*, suivi d'au moins un caractère et se terminant par *1*.

© Dunod – La photocopie non autorisée est un délit.

Il existe un troisième mécanisme de neutralisation qui utilise le caractère " (double quotes). Dans une chaîne délimitée par ce caractère ", certains métacaractères sont interprétés, dont le remplacement d'un paramètre (exemple $VAR) et l'évaluation d'une commande. Par contre il n'y a pas d'interprétation des espaces, ni de génération de nom de fichier.

Exemple

```
xstra> echo  e "Les fichiers suivants sont dans \<return>
le repertoire $PWD : \n $(ls)"
Les fichiers suivants sont dans le repertoire /xstra :
listf projet1
xstra>
```

7.9 EXERCICES

Exercice 7.9.1

Quelles commandes Linux devez-vous exécuter pour obtenir à l'écran :

Il y a xxx utilisateurs de ce systeme dont le login shell est bash

Exercice 7.9.2

En utilisant la commande *find*, trouvez et listez les noms de :

1) tous les fichiers sous le répertoire */etc* dont les noms commencent par *rc*,

2) tous les fichiers réguliers vous appartenant ; mettez le résultat dans le fichier */tmp/findmoi* et les erreurs dans */dev/null*,

3) tous les sous–répertoires de */etc*,

4) tous les fichiers réguliers se trouvant sous votre répertoire d'accueil et qui n'ont pas été modifiés dans les 10 derniers jours

Exercice 7.9.3

Trouvez à partir de votre répertoire d'accueil, le nombre de fichiers ayant une taille supérieure à 1 Mega-octets et stockez leurs noms dans un fichier (utilisez la commande *tee*).

Exercice 7.9.4

Créez dans le répertoire *rep1* le fichiers suivants : *fich1*, *fich2*, *fich11*, *fich12*, *fich1a*, *ficha1*, *fich33*, *.fich1*, *.fich2*, *toto*, *afich*.

Listez les fichiers :

1) dont les noms commencent par *fich*,

2) dont les noms commencent par *fich* suivi d'un seul caractère,

3) dont les noms commencent par `fich` suivi d'un chiffre,
4) dont les noms commencent par `.`,
5) dont les noms ne commencent pas par `f`,
6) dont les noms contiennent `fich`.

Exercice 7.9.5

Écrivez un alias en Bash permettant de lister page par page et dans l'ordre alphabétique l'ensemble des variables d'environnement.

Exercice 7.9.6

Créez un fichier de nom -i, puis supprimez-le.

Exercice 7.9.7

Il arrive souvent de lancer une commande Linux produisant plusieurs pages d'écran à toute vitesse (par exemple : `ls  l /etc`). Il faut alors relancer la même commande, en envoyant sa sortie standard dans `less`, ce qui permet d'examiner le résultat page par page. Si votre shell interactif est le Bash, cela ne doit pas être fastidieux : créez l'alias `p` qui relance la dernière commande en envoyant sa sortie standard dans la commande `less`.

© Dunod – La photocopie non autorisée est un délit.