TP introduction à l'utilisation du shell Unix

Hervé Charlery & Emmanuel Viaud

1 Resumé de quelques commandes de base

Si vous voulez plus d'informations sur une commande, n'oubliez pas d'aller voir avant tout la page man...

1.1 pour copier

syntaxe : `cp` [options] *in out*

paramètres :

-i mode interactif

-f pas de confirmation

-- fin des options

-R copie récursive

1.2 pour déplacer ou renommer

syntaxe : `mv` [options] *in out*

paramètres :

-i mode interactif

-f pas de confirmation

-- fin des options

1.3 pour rechercher une chaîne de caractères

syntaxe : `grep` [options] *motif fichier*
motif est une chaîne ou une expression régulière

paramètres :

-n affiche le numéro de ligne

-R recherche récursive

-c compter le nombre de lignes contenant le motif

1.4 pour comparer deux fichiers

syntaxe : `diff` [options] *source cible*

paramètres :

-b ne pas tenir compte des différences concernant les espaces

-B ne pas tenir compte des différences concernant les lignes blanches

-i ne pas tenir compte des différences de casse

-r comparaison récursive

note : `diff` peut également servir à créer des patches. Pour cela, on pourra utiliser diff avec les options *-Nrud*. Par exemple, on entrera :

```
diff -Nrud ancien nouveau > fichier.patch
```

1.5 pour appliquer des modifications à un fichier

syntaxe : `patch` -p*nombre* < *fichier_patch*

paramètres :

-p*nombre* spécifie le chemin des fichiers à patcher (à comparer avec le chemin indiqué dans *fichier_patch*). Incrémenter *nombre* de 1 revient à supprimer un ensemble répertoire + /.

1.6 pour rechercher un fichier, ... sur le disque

syntaxe : `find` *chemin expression*

paramètres : *chemin* spécifie le chemin à partir duquel s'effectue la recherche. *expression* est un ensemble d'options, de tests et d'actions. Pour la liste des options, cf. la page *man*. De nombreux tests existent, les plus courants sont :

-amin *n* dernier accès au fichier il y a *n* minutes

-group *groupe* fichier appartenant au groupe *groupe*, ceci peut être le nom ou le GID

-user *utilisateur* fichier appartenant à l'utilisateur *utilisateur*, ceci peut être le nom ou l'UID

-name *motif* fichier dont le nom correspond au motif (accepte les caractères jokers et est case sensitive, pour une version non case sensitive, utiliser -iname)

-regex *motif* fichier dont le nom correspond à la regex *motif*

-type *c* fichier du type *c* où *c* peut être (entre autres) :

d répertoire

f fichier régulier

l lien symbolique

1.7 pour créer des "scripts" d'automatisation

syntaxe : `tee` *fichier* tee se contente de lire depuis l'entrée standard et de rediriger la sortie à la fois vers la sortie standard et vers *fichier*. Ce programme est très utile en conjonction avec la redirection d'entrée pour créer des scripts automatisant le fonctionnement de certains programmes.

2 *Pipe* et filtres

Les *pipe* permettent d'enchaîner plusieurs commandes à la suite sur une même ligne de commande. Ils permettent d'utiliser les concepts à la base d'Unix (cf. http ://www.faqs.org/docs/artu/ch01s06.html), à savoir utiliser des outils simples et les lier entre eux par des *pipe* plutôt que de créer un unique gros programme. Par exemple :

```
$ gunzip -c pipo.gz | less
$ cat /etc/passwd | grep root | cut -d : -f 7
```

3 Redirections

Les symboles < et > ont pour tâche de rediriger respectivement les entrées et sorties vers un programme ou un fichier.

3.1 les redirections d'entrée

Elles permettent d'éviter d'avoir à constamment taper les mêmes lignes en entrée d'un script. Par exemple :

```
$ cat script.sh
#!/bin/bash
if [ $# ==  0 ]
then
        read in
        echo "salut $in"
else
        echo "salut $1"
fi
exit

$ cat data
toto

$ ./script.sh toto
salut toto

$ ./script.sh < data
salut toto
```

3.2 les redirections de sortie

Elles permettent de rediriger la ou les sorties standards vers un fichier. Il en existe deux types : > ou >>. Dans le premie cas, on écrase le fichier de sortie. Dans le second cas, on écrit à la fin du fichier. On peut rediriger au choix, STDOUT, STDERR ou les 2. Les redirections se basent sur la numérotation[1] des IO propres à Unix. Ainsi, si vous voulez :

rediriger STDOUT `$ ./script > fichier`

rediriger STDERR `$ ./script 2> fichier`

rediriger STDOUT et STDERR `$ ./script 2>fichier >fichier2`

rediriger STDERR vers STDOUT `$ ./script &>fichier`

Ce principe de numérotation des IO est aussi utile lorsque l'on programme pour faciliter le tri dans les logs produits par un programme. Il est ainsi

[1] 0 pour STDIN, 1 pour STDOUT, 2 pour STDERR

préférable de ne pas utiliser `printf` mais plutôt `fprintf` qui permet de spécifier sur quel descripteur on veut écrire.

4 Les variables du SHELL

4.1 Les variables d'environnement

$HOME : Répertoire principale de l'utilisateur.

$PATH : Liste de répertoires séparés par deux points, où chercher les commande.

$PS1 : Invite du shell.

$PS2 : Invite secondaire lors de la saisie de commande.

$IFS : Séparateur de champs lors d'un saisie.

$0 : Nom du script de shell en cours d'exécution.

$# : Nombre de paramètres transmis à un script.

$$: Numéro d'identification du processus du script de shell.

$? : Code de retour du script ou de la dernière commande exécutée. En général il vaut 0 si tout s'est bien passé.

4.2 Les variables de paramètres

$0, $2, ... : Paramètres transmis au script.

$* : Liste de tous les paramètres.

$@ : Identique à $* sans utiliser $IFS comme séparateur.

5 Les opérateurs de comparaison et de condition

5.1 Comparaison de chaînes

chaine1 = chaine2 : Vrai (True) si les chaînes sont identiques.

chaine1 != chaine2 : Vrai si les chaînes sont diffétents.

-n chaine : Vrai si la chaîne n'est pas vide (not null).

-z chaine : Vrai si la chaîne est vide (null).

5.2 Comparaison arithmétique

expression1 -eq expression2 : Vrai si les expressions sont égales.

expression1 -ne expression2 : Vrai si les expressions sont différentes.

expression1 -gt expression2 : Vrai si expression1 est supérieure à expression2.

expression1 -ge expression2 : Vrai si expression1 est supérieure ou égale à expression2.

expression1 -lt expression2 : Vrai si expression1 est inférieure à expression2.

expression1 -le expression2 : Vrai si expression1 est inférieure ou égale à expression2.

! expression : le caractère ! créé l'opposé de cette expression, renvoyant True si l'expression est False, et inversement.

5.3 Conditions de fichier

-d fichier : Vrai si le fichier est un répertoire.

-e fichier : Vrai si le fichier existe.

-f fichier : Vrai si le fichier est un fichier normal.

-g fichier : Vrai si set-group-id est activé pour ce fichier.

-r fichier : Vrai si le fichier peut être lu.

-s fichier : Vrai si la taille du fichier est ¿ 0.

-u fichier : Vrai si set-user-id est activé pour ce fichier.

-w fichier : Vrai si le fichier peut être écrit.

-x fichier : Vrai si le fichier est exécutable.

6 Les structures de contrôles

6.1 If

Syntaxe :

```
if condition
then
   instructions
else
```

instructions
`fi`

6.2 For

Syntaxe :

`for` variable `in` valeurs
`do`
instructions
`done`

6.3 While

Syntaxe :

`while` condition
`do`
instructions
`done`

6.4 Until

Syntaxe :

`until` condition
`do`
instructions
`done`

6.5 Case

Syntaxe :

`case` variable `in`
modèle `[` | modèle`]` ...)instructions ; ;
modèle `[` | modèle`]` ...)instructions ; ;

```
    ...
esac
```