

CH.4 LE LANGAGE SHELL

- 4.1 Les langages de commandes
- 4.2 Les caractères spéciaux
- 4.3 Les variables du Shell
- 4.4 Les fichiers de commandes
- 4.5 Les variables des fichiers de commandes
- 4.6 Les opérations
- 4.7 Les tests
- 4.8 Les itérations
- 4.9 Quelques commandes utiles

Info L1-1 Systèmes ch6 1

Les divers langages existants sont très semblables : sh, csh, ksh, tesh ou bash (utilisé ici).

Le même mot désigne l'interpréteur de commandes (shell) et le langage dans lequel les commandes sont écrites (Shell).

Info L1-1 Systèmes ch6 3

4.1 Les langages de commandes

Les *langages de commandes* réalisent l'interface entre le système et l'utilisateur.

Ce sont tous des langages *interprétés* permettant de transmettre des commandes avec une (plus ou moins) grande souplesse :

- usage de métacaractères (*);
- contrôle de l'environnement ;
- présence de nombreuses commandes prédéfinies ;
- possibilité d'écrire des commandes complexes ;
- possibilité d'écrire des fichiers de commandes (script-shell) .

La plupart des problèmes courants de gestion du travail peuvent être résolus par des commandes Shell.

Info L1-1 Systèmes ch6 2

4.2 Les caractères spéciaux

Caractères d'abréviation : * ?

(Le point en début de chaîne n'est pas reconnu par * ou ?) :

```
$ ls *.c
$ ls toto.?
```

Caractères liés au lancement d'une commande :

```
 ; ( ) > >> 2> 2>> < & |
# \ ' ' " " ` `
```

Désécialisation des métacaractères :

En les faisant précéder de la contre-oblique :

\\

ou en les incluant dans des délimiteurs : ' ' ou " " :

```
$ echo "bonjour > toto"
bonjour > toto
```

Info L1-1 Systèmes ch6 4

4.3 Les variables du Shell

On peut affecter des variables. Si a est une variable, \$a représente sa valeur et echo permet d'en prendre connaissance :

```
$ x=gh
$ echo $x
gh
$ echo $xijk

$ echo ${x}ijk
ghijk
```

Les caractères spéciaux ' et " permettent une délimitation des chaînes de caractères. Dans le premier cas \$nom n'est pas évalué, mais il l'est dans le second :

Info L1-1 Systèmes ch6 5

```
$ a=truc
$ x='machin$a'
$ y="machin$a"
$ echo $x
machin$a
$ echo $y
machintruc
```

Les accents graves permettent d'affecter à une variable le résultat qui serait produit par la commande placée entre accents :

```
$ a=`whoami`
$ echo $a
desar
```

Enfin la commande read permet de lire des valeurs entrées directement :

Info L1-1 Systèmes ch6 6

```
$ read prenom nom adresse
Jacques Desarmenien bureau 4B089
$ echo $prenom
Jacques
$ echo $nom
Desarmenien
$ echo $adresse
bureau 4B087
```

Un certain nombre de variables sont prédéfinies, suivant le contenu de fichiers du genre .bashrc ou .profile :

```
$ echo $HOME
/users/ens/desar
$ echo $PATH
/bin:/usr/bin:/usr/local/bin:/users/ens/desar/bin:
$ echo $PS1
$
$ PS1='& '
&
```

Info L1-1 Systèmes ch6 7

La liste des variables et de leurs valeurs peut être obtenue par la commande set. Une variable peut être effacée de l'environnement par la commande unset :

```
$ a=bonjour
$ set
HOME=/users/ens/desar
IFS=

LOGNAME=desar
PATH=/bin:/usr/bin:/usr/local/bin:/users/ens/desar/bin:
PS1=$
PS2=>
TERM=xterm
a=bonjour
$ echo $a
bonjour
$ unset a
$ echo $a
```

Info L1-1 Systèmes ch6 8

4.4 Les fichiers de commandes

On peut fabriquer un fichier de commandes (ou script-shell). On rassemble des commandes dans un fichier qu'on peut exécuter de plusieurs façons.

```
$ cat bonjour
echo Bonjour, `whoami` !
$ bash bonjour      (crée un nouveau shell)
Bonjour, desar !
$ . bonjour         (redirige l'entrée du shell)
Bonjour, desar !
```

La façon "classique" de procéder est de rendre le fichier exécutable :

```
$ chmod 744 bonjour
```

de la sorte le nom devient une commande "de plein droit" :

```
$ bonjour          (comme bash bonjour)
Bonjour, desar !
```

Info L1-1 Systèmes ch6 9

Attention de ne mettre qu'une commande par ligne, ou d'utiliser des séparateurs ;

Il existe la possibilité d'utiliser un formalisme de fonction au lieu de scripts-shell. Le résultat est le même.

Les commandes écrites par l'utilisateur peuvent s'appeler l'une l'autre ou même récursivement.

Le mieux est de placer les commandes fréquemment utilisées dans le répertoire bin sous son répertoire de travail (et vérifier que celui-ci est indiqué dans PATH).

Pour créer assurer une copie de sauvegarde :

```
chmod 755 ~/sauvegarde
mv -f $i ~/sauvegarde
chmod 555 ~/sauvegarde
```

Info L1-1 Systèmes ch6 10

4.5 Les variables des fichiers de commandes

Les fichiers de commandes peuvent comporter des paramètres, qui sont entrés sur la ligne de commande à l'exécution :

```
$ cat echange
ln -f $1 _tempo
ln -f $2 $1
ln -f _tempo $2
rm _tempo
$ cat tintin
Je suis Tintin !
$ cat milou
Et moi Milou !
$ echange tintin milou
$ cat tintin
Et moi Milou !
$ cat milou
Je suis Tintin !
```

Info L1-1 Systèmes ch6 11

* est la liste des arguments ;

est le nombre d'arguments ;

o est le nom de la commande ;

1, ..., 9 sont les 9 premiers arguments.

Bien entendu, il est possible de définir des variables locales dans un fichier de commandes. Ces variables ne sont pas passées aux autres processus (elles restent locales).

Il est possible de les passer aux processus fils (variables exportables).

Voir les détails de la commande export.

Info L1-1 Systèmes ch6 12

4.6 Les opérations

La commande `expr` permet de faire des opérations entre ses arguments.
Attention à tous les blancs entre arguments ! Se souvenir que les variables ont comme valeur des chaînes de caractères.
Penser à déspecialiser les métacaractères.

```
$ expr 4 + 5
9
$ a=5
$ expr 4$a + 5
50
$ expr $a \< < 7
1
$ expr 1 \< < z
1
$ expr tintin \< < milou
0
```

Info L1-1 Systèmes ch6 13

4.7 Les tests

Deux formes possibles : `test expression` ou `[expression]`. Attention aux espaces !

Tests sur les chaînes de caractères :

```
test -z chaîne est vrai ssi chaîne est vide
test -n chaîne est vrai ssi chaîne est non vide
test chaîne1 = chaîne2
test chaîne1 != chaîne2
```

Info L1-1 Systèmes ch6 15

Les opérations possibles sont :

+ - * / % (reste) sur des opérandes entiers ;
< > = >= <= != (différent de) sur des opérandes quelconques, avec comparaisons faites suivant l'ordre lexicographique entre chaînes de caractères ;
| & opérations booléennes.

La priorité est la priorité usuelle. On peut grouper avec des parenthèses, qu'il faut penser à déspecialiser ; l'arithmétique est assez pénible.

```
$ a=`expr 2 \* \( 3 + 4 \)` (affectation)
$ echo $a
14
$ a=`expr $a + 1` (incrémentatation)
$ echo $a
15
```

Info L1-1 Systèmes ch6 14

Exemple :

```
if [ `pwd` = $HOME ]
then echo "le catalogue est bien le catalogue de travail"
fi
```

Tests sur les nombres **entiers** :

```
test chaîne1 -eq chaîne2
test chaîne1 -ne chaîne2
test chaîne1 -lt chaîne2
test chaîne1 -le chaîne2
test chaîne1 -gt chaîne2
test chaîne1 -ge chaîne2
```

Info L1-1 Systèmes ch6 16

Tests sur les fichiers (entre autres...) :

```
test -f nom est vrai ssi nom est un fichier
test -d nom est vrai ssi nom est un répertoire
test -r nom est vrai ssi nom est autorisé en lecture
test -w nom est vrai ssi nom est autorisé en écriture
test -x nom est vrai ssi nom est autorisé en exécution
test -s nom est vrai ssi nom est de taille non nulle
```

Connecteurs : `!` (non), `-a` (et), `-o` (ou). L'ordre d'évaluation peut être forcé par des parenthèses déspecialisées `\(\)`.

Les tests sont rarement utilisés seuls (bien que cela soit possible). Ils le sont en général dans une conditionnelle.

Info L1-1 Systèmes ch6 17

Quelques exemples :

```
$ cat rencontre
ls $1 > /dev/null 2> /dev/null
if test $? -eq 0
then echo "Oui, $1 existe, je l'ai rencontré."
else echo "Pour $2, repassez plus tard."
fi
$ ls
adieu affiche rencontre tintin
$ rencontre tintin
Oui, tintin existe, je l'ai rencontré.
$ rencontre Dieu
Pour Dieu, repassez plus tard.
$ cat affiche
if [ $# -ne 1 ] ; then echo "syntaxe : $0 nom_de_fichier"; exit 1; fi
if [ -f $1 ]; then cat $1
elif [ -d $1 ]; then ls $1
else echo "$1 est un fichier special ou n'existe pas."
fi
```

Info L1-1 Systèmes ch6 19

Les formes possibles sont :

```
if commande1      si commande1 est vrai
then commande2    commande2 est exécuté,
else commande3    sinon c'est commande3,
fi
```

Attention : les mots-clés then, else et fi doivent être en **début de ligne** ou précédés de ;

La clause else est facultative.

Si on enchaîne des tests, on peut utiliser la syntaxe
`if ... then ... elif ... then ... else ... fi`

Info L1-1 Systèmes ch6 18

```
$ ls
adieu affiche rencontre tintin
$ affiche adieu tintin
syntaxe : ./affiche nom_de_fichier
$ affiche adieu
Bonsoir !
$ affiche .
adieu affiche rencontre tintin
$ affiche Dieu
Dieu est un fichier special ou n'existe pas.
```

Il existe aussi un aiguillage case ... esac

```
case chaîne in
motif1) commande1 ;;
motif2) commande2 ;;
...
esac
```

Info L1-1 Systèmes ch6 20

Comme exemple, la commande suivante appelle le bon compilateur selon l'extension du fichier. Le nom de celui-ci peut être entré en argument ou entré en ligne :

```
case $# in
0) echo -n "Fichier a compiler : " (pas de passage à la ligne)
  read reference
  set $reference;;          (affecte reference à 1)
esac
case $1 in
*.c) cc $1 ;;
*.p) pc $1 ;;
*.f) ff $1 ;;
*) echo "Compilation de $1 impossible" ;;
esac
```

Info L1-1 Systèmes ch6 21

Autres itérations :

```
while ... ; do ... ; done Cf until ... ; do ... ; done
```

Exemple :

```
$ cat invite
echo Repondre par oui ou non :
read reponse
while [ "$reponse" != oui -a "$reponse" != non ]
do echo Repondre par oui ou non :
read reponse
echo "Votre reponse est : $reponse."
done
$ invite
Repondre par oui ou non :
oui
Votre reponse est : oui.
$ invite
Repondre par oui ou non :
zut
Repondre par oui ou non :
non
Votre reponse est : non.
```

Info L1-1 Systèmes ch6 23

4.8 Les itérations

L'itération bornée :

```
for variable in chaîne1 chaîne2 ...
do commande
done
```

La liste de chaînes peut être explicite, la valeur d'une variable, souvent \$*, le résultat d'une commande entre accents graves, le répertoire de travail avec *.

Les mots-clés do et done doivent être en **début de ligne** ou précédés de ;

Exemple : Pour afficher le contenu de tous les fichiers dont l'extension est .txt :

```
for i in `ls *.txt`; do cat $i; done
```

Info L1-1 Systèmes ch6 22

Création dynamique de fichiers :

```
$ cat cree
i=1
until [ $i -gt $1 ]
do echo "valeur $i" > xx$i
i=`expr $i + 1`
done
$ cree 3
$ ls xx*
xx1 xx2 xx3
$ cat xx2
valeur 2
```

Il est déconseillé de se livrer à un tel jeu sans précautions !

Cela risque de saturer le disque rapidement...

Info L1-1 Systèmes ch6 24

4.9 Quelques commandes utiles

Faire man bash, pour avoir une liste de ce que comprend le shell.

La commande alias *nom*="*chaîne*" crée un alias *nom* pour une *chaîne*.
La commande alias permet de connaître la liste des alias actifs.

La commande cut pour sélectionner des champs.

```
$ echo "ab.cd.ef" | cut -d . -f 2
cd
$ echo "ab.cd.ef" | cut -d . -f 2-
cd.ef
$ echo "ab.cd.ef" | cut -d . -f -2
ab.cd
```

Info L1-1 Systèmes ch6 25

Les premières et les dernières lignes head et tail :

```
head -n garde les n premières lignes ;
tail -n garde les n dernières lignes ;
tail +n garde les lignes à partir de la ligne n ;
```

La commande suivante écrit toto1, puis ses lignes sauf la première, etc.

```
$ while [ -s toto1 ]; do cat toto1; tail +2 < toto1 > toto2;
mv -f toto2 toto1; done
```

Remplacement de caractères tr :

Pour mettre en minuscules tous les noms de fichiers du répertoire,

```
$ for i in `ls`; do j=`echo $i | tr "[A-Z]" "[a-z]";
mv -f $i $j; done
```

Édition de texte sed :

Pour remplacer dans les fichiers d'extension .txt toutes les occurrences de Pierre par Jacques,

```
$ for i in `ls *.txt`; do sed s/Pierre/Jacques/g < $i > $i.new;
mv -f $i.new $i; done
```

Info L1-1 Systèmes ch6 27

Les *filtres* lisent sur l'entrée standard, la modifient et renvoient le résultat sur la sortie standard. L'entrée standard peut souvent être remplacée par une liste de fichiers spécifiés en paramètres. On peut les utiliser dans des tubes. Pour chacune de ces commandes, voir le man correspondant.

Identité cat .

Décompte de lignes, mots et caractères wc .

```
$ echo "ab.cd.ef" | wc -c
9
$ wc -l < toto1
4
```

Sélection de motifs grep.

Mise en ordre alphabétique sort.

Info L1-1 Systèmes ch6 26