Memento Shell

Auteur : Antoine Pernot

• **Gestion des fichiers/répertoires**

Créer un répertoire (make directory) :

`-p` : Créer le répertoire parent si besoin.

```
mkdir [-p] rep
```

Créer un fichier :

```
touch fich
```

Changer de répertoire (change dir) :

```
cd rep
cd ..      # repertoire parent
cd         # repertoire personnel
cd ~alice # repertoire personnel de alice
```

Afficher répertoire courant (print working dir) :

```
pwd
```

Copier un fichier/répertoire vers un autre :

`-r` : Copier le répertoire de manière récursive.

```
cp [-r] orig dest
```

Créer un lien :

`-s` : Créer un lien symbolique.

```
ln [-s] orig lien
```

Déplacer/renommer un fichier/répertoire vers un autre (move) :

```
mv orig dest
```

Supprimer un fichier (remove) :

`-r` : Supprime un répertoire de manière récursive.

```
rm [-r] fich
```

Supprimer un répertoire vide (remove directory) :

```
rmdir rep
```

Lister le contenu d'un répertoire :

`-l` : Liste détaillée.
`-a` : Liste tous les fichiers (inclut les fichiers cachés).
`-s` : Tri par taille.
`-t` : Tri par date.
`-r` : Inverse l'ordre de tri.

```
ls [-latsr] rep
```

• **Gestion du contenu des fichiers/des flux**

Affichage brut de fichiers (non-interactif) :

```
cat [fich1 [fich2 ...]]
```

Affichage interactif de fichiers :

```
more [fich1 [fich2 ...]] # sens unique
less [fich1 [fich2 ...]] # 2 sens
```

Afficher le début d'un fichier :

`-n` : Affiche n lignes.

```
head [-n=10] [fich]
```

Afficher la fin d'un fichier :

`-n` : Affiche n lignes.

```
tail [-n=10] [fich]
```

Rechercher dans un fichier :

`-i` : Insensible à la casse.
`-v` : Affiche les lignes ne correspondant PAS à l'expression.

```
grep [-iv] expression [fich]
```

Trie les lignes d'un fichier :

`-r` : Inverse l'ordre.
`-R` : Trie dans un ordre aléatoire.

```
sort [-rR] [fich]
```

Compte les éléments d'un fichier :

`-l` : Compte le nombre de lignes.
`-w` : Compte le nombre de mots.
`-c` : Compte le nombre de caractères.

```
wc [-clw] [fich]
```

Découper un flux de texte :

`-d` : Délimiteur de découpe.
`-f` : Sélectionne les champs à renvoyer.

```
cut [-df] [fich]
```

• **Droits d'accès aux fichiers**

Changer les droits d'accès aux fichiers :

`-R` : Change les droits de manière récursive.

```
chmod [-R] {ugoa}{+-}{rwx} fich
```

Changer le propriétaire du fichier :

`-R` : Change le propriétaire de manière récursive.

```
chown [-R] nvuser[:nvgrp] fich
```

Changer le groupe du fichier :

`-R` : Change le groupe de manière récursive.

```
chgrp [-R] nvgrp fich
```

• **Recherche de fichiers**

Rechercher un fichier :

`-exec` : Exécute une commande en replaçant {} par le chemin de chacun des fichiers trouvés.

```
find rep_rech -name regex [-exec cmd {} ';']
```

• **Gestion des flux de texte**

Rediriger la sortie d'une commande vers l'entrée d'une autre :

```
cat villes.txt | grep Dijon
```

Écrire la sortie d'une commande dans un fichier (écrase le contenu) :

```
grep Dijon villes.txt > info_Dijon.txt
```

Ajoute la sortie d'une commande à un fichier :

```
grep Dijon villes.txt >> info_Dijon.txt
```

Ajoute la sortie d'erreurs d'une commande à un fichier :

```
grep Dijon villes.txt 2>> erreurs_villes.txt
```

• **Contrôle de tâches**

Affiche les processus en cours d'exécution :

```
ps aux
top     # Mode interactif
```

Envoie un signal (d'arrêt) au processus :

`-9` : Envoie un signal d'arrêt SIGKILL

```
kill [-9] pid
```

Envoie un signal (d'arrêt) aux processus appelés "nom" :

```
killall nom
```

• **Aide système**

Lister les pages de manuel contenant une chaîne de caractère :

```
apropos chaine
```

Afficher la page de manuel d'une commande :

```
man cmd
```

Memento scripts

Auteur : Antoine Pernot

- **Structure de base**

```
#!/bin/bash
# Version du script
echo "Bonjour"
exit 0
```

- **Les variables**

Affecter une variable :

```
message="Coucou !"
read rep # Stocke la reponse utilisateur
```

Appeler une variable :

```
echo $message
```

Appeler un fragment de chaîne de caractères :

```
echo ${message:offset:nbchars}
```

Variables spéciales :

`$*`	Contient tous les arguments passés à la fonction.
`$#`	Contient le nombre d'argument.
`$?`	Contient le code de retour de la dernière opération.
`$0`	Contient le nom du script.
`$n`	Contient l'argument n.
`$!`	Contient le PID de la dernière commande lancée.

- **Les tableaux**

Affectation (1ère méthode) :

```
tab=(valeur1 valeur2 ...)
```

Affectation (2ème méthode) :

```
tab[0]=John Smith
tab[1]=Jane Doe
```

Compter le nombre d'éléments du tableau :

```
len=${#tab[*]}
```

Afficher un élément :

```
echo ${tab[1]}
```

Afficher tous les éléments :

```
echo ${tab[@]}
```

- **Les structures de contrôle**

Syntaxe :

```
[ -f fichier ]
```

Opérateurs de test :

`-e fichier`	Contrôle si fichier existe.
`-d fichier`	Contrôle si fichier existe et est un répertoire.
`-f fichier`	Contrôle si fichier existe et est un fichier 'normal'.
`-w fichier`	Contrôle si fichier existe et est en écriture.
`-x fichier`	Contrôle si fichier existe et est exécutable.
`f1 -nt f2`	Contrôle si f1 est plus récent que f2.
`f1 -ot f2`	Contrôle si f1 est plus vieux que f2.

Opérateurs de comparaison numériques :

`$n1 -eq $n2`	Vérifie si les nombres sont égaux. Utiliser = pour les chaînes de caractères.
`$n1 -ne $n2`	Vérifie si les nombres sont différents. Utiliser != pour les chaînes de caractères.
`$n1 -lt $n2`	Vérifie si n1 est inférieur à n2.
`$n1 -le $n2`	Vérifie si n1 est inférieur ou égal à n2.
`$n1 -gt $n2`	Vérifie si n1 est supérieur à n2.
`$n1 -ge $n2`	Vérifie si n1 est supérieur ou égal à n2.

Les opérateurs logiques :

`expr1 -a expr2` ou `expr1 && expr2`	Opérateur ET.
`expr1 -o expr2` ou `expr1 || expr2`	Opérateur OU.
`! expr`	Opérateur NON.

- **Les conditions**

```
if [ condition1 ]
then
    Condition 1 vraie
elif [ condition2 ]
then
    Condition 2 vraie
else
    Aucune condition vraie
fi
```

- **Les boucles**

"Tant que ..." :

```
while [ condition ]
do
     Tant que la condition est vraie
done
```

"Pour ..." :

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

- **Les aiguillages**

```
case "$var" in
    val1 | val2 )  Valeur 1 ou 2;;
    val3 )         Valeur 3;;
    * )            Autres valeurs;;
esac
```

- **Les fonctions**

Déclaration de la fonction :

```
nom_fonction(){
        instructions
}
```

Appel de la fonction :

```
nom_fonction
```

- **La couleur**

Syntaxe :

```
echo -e '\033[A;B;Cm Bonjour ! \033[0m'
```

Valeurs d'effets (A) :

0	Normal
1	Gras
21	Non-gras
4	Souligné
24	Non souligné
5	Clignotant
25	Non-clignotant
7	Inversé
27	Non-inversé

Valeurs de couleur (B et C) :

Couleur	Couleur texte (B)	Couleur fond (C)
Noir	30	40
Rouge	31	41
Vert	32	42
Jaune	33	43
Bleu	34	44
Magenta	35	45
Cyan	36	46
Blanc	37	47