

Mémo console UNIX

Aide mémoire sur l'utilisation des consoles bash et quelques commandes sous Unix. Seule une utilisation simple et rudimentaire des commandes est présentée, pour plus de détails sur une commande, faire : `man commande` ou `commande --help`. Attention ' ` ` et " sont des caractères différents.

Fabien Bonnefoi - IUT de Villetaneuse - 2011

1 - Commandes de base : fichiers et répertoires

Changer de répertoire : `cd`

avec une chaîne de caractères 1 `cd /var/log/`
avec une chaîne de caractères 2 `cd '/usr/toto/Mes Docs/'`
avec une variable `cd $HOME`

Afficher le contenu d'un répertoire : `ls`

affichage simple `ls` ou `ls /var/log/`
affichage complet `ls -al`
affichage avec motif `ls *.txt`

Afficher un fichier: `cat`, `more`, `less`, `head`, `tail`

affiche tout d'un coup `cat fichier`
page par page `more fichier`
pour les gros fichiers `less fichier`
les premières lignes `head fichier`
les deux premières lignes `head -n 2 fichier`
affichage des dernières lignes `tail /var/log/kernel.log`

Afficher le répertoire courant: `pwd`

affichage simple `pwd`
résolution des liens symboliques `pwd -P`

Déplacer (ou renommer) un fichier : `mv`

vers un répertoire `mv fichier /chemin_rep/`
renommage `mv fichier fichier2`

Copier un fichier : `cp`

fichier source vers destination `cp source destination`
copie symbolique `cp -s source lien`

Supprimer un fichier ou un répertoire : `rm` et `rmdir`

suppression de plusieurs fichiers `rm fichier1 fichier2`
avec motif `rm *.tmp`
suppression d'un répertoire vide `rmdir /repertoire/`
du rep. et de son contenu `rmdir -r /repertoire/`

Créer un fichier ou un répertoire

créer un fichier vide 1 `> fichier`
créer un fichier vide 2 `touch fichier`
créer un fichier par redirection `echo "toto" > fichier`
création d'un répertoire `mkdir mon_rep`

Lecture et écriture sur les entrées-sorties standards

lecture (écrit dans la variable var) `read var`
écriture (voir aussi section 3) `echo "Bonjour $var"`

2 – Exécution, gestion des flux de contrôle et de donnée

Lancer une commande, un processus

normal `commande`
en arrière-plan `commande &`
en séquence `commande1;commande2;`
grouper les commandes `{ cmd1;cmd2; }`
dans un sous shell `(cmd1;cmd2;)`

(l'environnement est hérité mais pas modifié)

Les opérateurs logiques « et » (&&) et « ou » (||) permettent de conditionner l'exécution d'une succession de commandes (voir aussi section 5 et 7): `rm dir && echo OK || echo KO`

Redirection des entrées et sorties standards

Un processus Unix possède par défaut une entrée et une sortie standard et une sortie pour les messages d'erreurs. Les tubes | permettent de rediriger la sortie d'une commande vers l'entrée d'une autre commande.

mode tube (pipe) `commande1 | commande2`
redirection de la sortie std `commande > a.out`
ajout en fin de fichier `commande >> a.out`
les erreurs vers un fichier `commande 2> fichier.log`
rediriger l'entrée `commande << "titi toto"`
rediriger l'entrée 2 `commande < fichier`

Substitutions

Il est possible d'exécuter une commande lors de la création d'une chaîne de caractère ou d'une variable.. Par exemple pour afficher l'heure: `echo "Il est `date +%Hh%M`"`

substitution 1 ``commande arg1 arg2``
substitution 2 `$(commande arg1 arg2)`

Activité, état, terminaison

liste des processus utilisateur `ps`
liste de tous les processus `ps -A`
liste les processus et leur charge `top`
arrêter un processus par son pid `kill pid`
ou (SIGKILL) `kill -9 pid`
arrêter un processus par son nom `killall firefox`

3 – Opérations sur les variables

Pas de typage des variables, un entier peut devenir une chaîne de caractères et inversement. Des entiers, des chaînes, mais pas de réel.

Créer une variable

Créer une variable locale `local var`
en lecture seule `readonly var=123`
ou `declare -r...`
Créer un entier `declare -i var=10`
Créer une chaîne de car. `var=MonTexte`
Créer une chaîne de car.2 `var="Mon Texte"`
créer une chaîne de car.3 `var='Mon Texte'`

Utiliser une variable

afficher une variable `echo $var` ou `echo ${var}`
interprétée dans une chaîne `echo "Voici $var !"`
initialise une variable si elle n'est pas définie `${var:="toto"}`
rend toto si var n'est pas définie sans l'initialiser `${var:-"toto"}`

Supprimer une variable

Supprimer une variable `unset var`

Visibilité d'une variable

Par défaut, une variable est définie dans l'environnement d'exécution courant. Pour la rendre visible dans tout l'environnement il faut utiliser la commande `export`.

dans l'env. (en dehors du sscript) `export var`
voir aussi: `local`, `env`, `export`, `declare`, `let`, `set`

Arithmétique sur les entiers

exemple `var=$(((10 + 12) / 2))`
exemple `var=${(10 + 12) / 2}`
avec expr `var=$(expr \(10 + 12\) / 2)`
avec expr 2 `var=`expr $var1 - $var2``

Variables d'environnement à connaître

Nom de la machine `HOSTNAME`
Répertoire d'utilisateur `HOME`
(on peut aussi utiliser `~` qui est un raccourci et pas une variable)
Nom (login) de l'utilisateur `USER`
Répertoire courant `PWD`
contient la liste des répertoires où se trouvent les commandes que l'utilisateur peut exécuter `PATH`
contient le nom du shell de connexion `SHELL`
Taille de l'historique des commandes `HISTSIZE`
l'invite de commande (le prompt) `PS1`

4 – Manipulation des chaînes de caractères

Il existe différentes manières de définir une chaîne de caractère. En fonction de la méthode utilisée, les caractères spéciaux ne sont pas interprétés de la même manière. Les caractères spéciaux sont:

" , ' , ` , \ , \$, ! , ? , ~ , () , [] , { } , @ , * ...

Ils peuvent être protégés par `\` ou entre `'` ou `"`.

Créer une chaîne

chaîne simple `var1=toto\ a\ $var2\ ans`
sans interprétation `var1='toto a 10 ans'`
seul `\`` et ``` sont interprétés `var1="toto a $var2\$"`
par évaluation de commande `var=`ls`` ou `var=$(ls)`

Expansion de chaîne

taille d'une chaîne `${#var}`
extraire la sous chaîne à partir d'une position (ici 5): `${var:5}`
à partir d'une pos. pour une longueur donnée (ici 10): `${var:5:10}`
supprimer la chaîne...
- la plus courte à droite (commencent par /*) `${PWD%/*}`
- la plus longue à gauche (ici terminant par /*) `${HOME##*/}`
remplacer une occurrence `${var/rouge/vert}`

5 – Fichier de commande

Pour créer un fichier de commande bash, il suffit d'écrire une succession de commandes dans un fichier texte dont l'extension par défaut est `.sh`, et la première ligne `#!/bin/bash`.

Arguments

Le nombre d'arguments `$#`
Le premier, deuxième, etc, argument `$1, $2, etc.`
Le nom du script `$0`
La liste de tous les arguments `$* ou $@`

Il est possible d'affecter les paramètres d'un script tout en mettant à jour automatiquement les paramètres spéciaux `#, * et @`.

affecter les paramètres `$1, $2 (et $#, $*)` `set toto 21`

Variables standards

code de retour de la dernière commande `$?`
numéro de processus de la dernière commande `$!`
numéro de processus de la console `$$`

Fonctions

Pour déclarer une fonction en bash, on utilise le mot clé `function` suivi du nom de la fonction. Le code de la fonction est ensuite encadré par `{ et }`:

```
function foo {  
    echo 'Hello World !'  
}
```

Les arguments (implicites) sont contenus dans `$*`, `$1` etc.

```
function foo2 {  
    echo $1, $2...  
    echo "$# argument(s) : $*." ;  
}
```

invocation d'une fonction `foo2 toto 21 titi`

exporter la déclaration d'une fonction `declare -f foo`

Valeur de sortie (ou code de retour)

Par défaut la valeur de sortie d'un script est celle de sa dernière commande (écrite dans la variable `$?`). On peut en spécifier une autre avec la commande `exit` (zéro signifie une sortie sans erreur).

spécifier une valeur de sortie `exit 12345`

6 – Tableaux et dictionnaires

Les tableaux (dits tableaux indexés) et dictionnaires (dits associatifs) fonctionnent de manière similaire, mais un dictionnaire doit obligatoirement être déclaré.

déclaration d'un tableau `declare -a tab`
déclaration d'un dico. `declare -A dico`
initialisation `tab=(8 titi 10 toto)`
initialisation 2 `tab=([0]=8 [1]=titi)`
initialisation de dico. `dico=([titi]=8 [toto]=rien)`
afficher une valeur `echo ${tab[1]}`
afficher une valeur de dico. `echo ${dico[titi]}`
taille d'un tableau `${#tab[*]}`
taille du 2^{ème} élément `${#tab[2]}`
liste des indices définit `$tab[!tab[*]]`
afficher un dictionnaire (ou un tableau) `echo ${dico[*]}`

7 – Structures de contrôle

Il existe principalement deux techniques de gestion des flux de contrôles:

1) en utilisant le code de retour des commandes (0 = vrai sinon faux) combiné avec les opérateurs `&&` et `||` entre des commandes ou groupement de commandes (voir aussi section 2)

exemple `rm dir && echo OK || echo KO`

2) en effectuant des tests booléens avec la commande `test`, ou `[et]`, `[[et]]` ou `((et))`; combinés avec les opérateurs de contrôle `if, case, for` ou `while`.

Tests

• La commande `test` et la notation entre `[et]` permettent les opérateurs suivants (entre autres):

Test sur les fichiers (vrai si...)	
fichier existe	<code>[-e fichier]</code>
fichier est un répertoire	<code>[-d fichier]</code>
fichier est lisible	<code>[-r fichier]</code>
fichier vous appartient	<code>[-o fichier]</code>
fichier est exécutable	<code>[-x fichier]</code>
Comparaison de chaîne de caractère (vrai si...)	
\$var est vide	<code>[-z "\$var"]</code>
\$var est non vide	<code>[-n "\$var"]</code>
\$var égale toto	<code>["\$var" == "toto"]</code>
\$var diffère de toto	<code>["\$var" != "toto"]</code>
\$var et avant dans le dico	<code>["\$var" \< "toto"]</code>
Comparaison arithmétiques (vrai si...)	
\$nombre vaut 12	<code>[\$nombre -eq 12]</code>
\$nombre est différent de 12	<code>[\$nombre -ne 12]</code>
\$nombre inférieur à	<code>[\$nombre -lt 12]</code>
\$nombre inférieur ou égal à	<code>[\$nombre -le 12]</code>
\$nombre supérieur à	<code>[\$nombre -gt 12]</code>
\$nombre supérieur ou égal à	<code>[\$nombre -ge 12]</code>

exemple avec test `test "a" != "$HOME"`
ou `test 12 -eq 13;x=$?;echo $x;echo $?`

• La notation `[[...]]` évite le découpage des chaînes de caractères possédant des espaces, il n'est plus nécessaire de les protéger avec des `"`. Aussi, les caractères spéciaux ne sont plus interprétés en tant qu'extension de chemin mais en motifs de chaîne de caractère (voir les expressions rationnelles), les caractères `<` et `>` n'ont plus à être protégés par `\`. = peut être utilisé à la place de `==` pour une compatibilité POSIX.

• La notation `(((permet tests et opérations arithmétiques (voir aussi l'opérateur for):`

`(((var=3*5 == 15)) && echo $var || echo Non`
On peut combiner plusieurs tests en les encadrant avec `(et)` ou/et en utilisant les opérateurs `-a (pour et)` `-o (pour ou)`

exemple `[-d "$HOME" -a -w "$HOME"]`

exemple `[[(-d $HOME) && (-w $HOME)]]`

Opérateurs de contrôle

Exemple de structure de contrôle if:

```
if [ $# -gt 1 ] ; then  
    echo OK  
else  
    echo KO  
fi  
exemple de case:  
case $mot in  
    mot1 | mot2 ) echo bien ;;  
    mot3) echo super;;  
    *) echo bof  
esac
```

Boucles

for avec itération

```
for ((x=0; x <= 10; x++)) ; do  
    echo $x;  
done
```

for sur une liste (souvent une substitution de commande)

```
for i in element1 element2 ; do  
    echo $i  
done
```

while avec test conditionnel sur entier

```
while [ $i -lt 10 ] ; do  
    i=$(( $i + 1 )) # voir ((i++))  
    echo $i  
done
```

while avec une suite de commande, la suite_de_cmd2 est exécutée tant que suite_de_cmd1 retourne 0;

```
while suite_de_cmd1 ; do  
    suite_de_cmd2;  
done
```

8 – Compression, décompression

Seul `tar` et `gzip` sont intégrés au kernel de base.

créer une archive `tar -cf dest.tar source`
désarchiver `tar -xf archive.tar`
compresser `tar czvf archive.tar.gz source`
désarchiver et décompresser `tar xvzf archiver.tar.gz`
(voir aussi `gzip, bzip, zip, rar, par` etc.)

9 – Date, time

date style log `date +%Y-%m-%d\ %H:%M:%S.%N`
(%N pas portable sur mac. Posix ?)
sec. depuis le 01/01/1970 `date +%s`
date jolie `date +%A\ %d\ %B\ %Y\ %Hh%M`

10 – Raccourcis clavier

mettre la tâche active en arrière plan `ctrl + z puis bg`
arrêter un processus (envoi SIGTERM) `ctrl + c`
nettoyer l'écran `ctrl + l`
écrire le caractère de fin de fichier (EOF) `ctrl + d`
exécuter la dernière commande commençant par `!comm`

11 – Référence et liens

Cours de l'IUT: <http://www.lipn.fr/~cerin/SE/modele.html>

Site de bash: <http://tiswww.case.edu/php/chet/bash/bashtop.html>