Programmation avancée SHELL

1. Login/non login shell, interactive/non interactive shell

Un shell de login est un shell qui vous demande de vous loguer : (Username ? Password ?)

Si celui-ci ne demande pas de vous loguer (par exemple en ouvrant gnome-terminal) alors c'est un non-login shell.

Voici les cas où vous avez accès à un shell de login :

- Accès via ssh
- Simulation d'un login shell avec « bash –l »
- Simulation d'un login root : « sudo –i »
- Où « sudo –u username –i» pour un utilisateur non root
- Utilisation de su – username
- Utilisation de sudo login (pour changer d'utilisateur)

Un shell est *interactif* si son entrée standard et sa sortie standard sont toutes deux connectées à un terminal, ou s'il est invoqué avec l'option **-i**. Le paramètre **PS1** est positionné, et le paramètre **$-** contient la lettre **i** si **bash** est interactif, ce qui permet à un script ou à un fichier de démarrage de vérifier l'état du shell.

Le paragraphe suivant décrit comment **bash** exécute ses fichiers d'initialisation. Si l'un de ces fichiers existe mais n'est pas accessible en lecture, **bash** signale une erreur.

Lorsque **bash** est lancé comme shell de login interactif, ou comme shell non-interactif avec l'option **--login**, il lit et exécute tout d'abord les commandes se trouvant dans le fichier */etc/profile* s'il existe. Après lecture de ce fichier, il recherche*~/.bash_profile*, *~/.bash_login*, et *~/.profile*, dans cet ordre, et exécute les commande se trouvant dans le premier fichier existant et accessible en lecture.

L'option **--noprofile** peut être utilisée au démarrage du shell pour empêcher ce comportement.

Lorsqu'un shell de login se termine, **bash** lit et exécute les commandes du fichier *~/.bash_logout*, s'il existe.

Quand un shell interactif démarre sans être un shell de login, **bash** lit et exécute les commandes se trouvant dans *~/.bashrc* s'il existe. Ce comportement peut être inhibé à l'aide de l'option **--norc**. L'option **--rcfile** *fichier* forcera **bash** à exécuter les commandes dans *fichier* plutôt que dans *~/.bashrc*.

Quand **bash** est démarré de manière non-interactive, pour lancer un script shell par exemple, il recherche la variable **BASH_ENV** dans l'environnement, développe son contenu si elle existe, et considère cette valeur comme le nom d'un fichier à lire et exécuter. **Bash** se comporte comme si la commande suivante se trouvait en début de script :

if [-n "$BASH_ENV"]; then . "$BASH_ENV"; fi

mais la valeur de la variable **PATH** n'est pas utilisée pour rechercher le fichier.

2. Chaines de caractères

Protection de caractères

Le shell utilise différents caractères particuliers pour effectuer ses propres traitements ($ pour la substitution, > pour la redirection de la sortie standard, * comme caractère générique, etc.). Pour utiliser ces caractères particuliers comme de simples caractères, il est nécessaire de les protéger de l'interprétation du shell. Trois mécanismes sont utilisables :

- Protection d'un caractère à l'aide du caractère \

Ce caractère protège le caractère qui suit immédiatement le caractère \.

```
$ echo \*    => le caractère * perd sa signification de caractère générique
*
$ echo *
tata toto
$
$ echo \\   => le deuxième caractère \ perd sa signification de caractère
               de protection
\
$ echo N\'oublie pas !
N'oublie pas !
$
```

Le caractère \ permet également d'ôter la signification de la touche Entrée. Cela a pour effet d'aller à la ligne sans qu'il y ait exécution de la commande. En effet, après saisie d'une commande, l'utilisateur demande au shell l'exécution de celle-ci en appuyant sur cette touche. Annuler l'interprétation de la touche Entrée autorise l'utilisateur à écrire une longue commande sur plusieurs lignes.

Dans l'exemple ci-dessous, le shell détecte que la commande interne echo n'est pas terminée ; par conséquent, bash affiche une chaîne d'appel différente matérialisée par un caractère > suivi d'un caractère espace invitant l'utilisateur à continuer la saisie de sa commande.

```
$ echo coucou \Entrée
> salut Entrée   => terminaison de la commande : le shell l'exécute !
coucou salut
$
```

- Protection de caractères à l'aide d'une paire de guillemets *"chaîne"*

Tous les caractères de chaîne sauf $ \ ` " sont protégés de l'interprétation du shell. Cela signifie, par exemple, qu'à l'intérieur d'une paire de guillemets le caractère $ sera quand même interprété comme une substitution, etc.

```
$ echo "< * $PWD * >"
< * /home/sanchis * >
$
$ echo "< * \"$PWD\" * > "
< * "/home/sanchis" * >
$
```

- Protection totale *'chaîne'*

Entre une paire d'apostrophes ('), aucun caractère de *chaîne* (sauf le caractère ') n'est interprété.

```
$ echo '< * $PWD * >'
< * $PWD * >
$
```

Longueur d'une chaîne de caractères

Syntaxe : ${*#paramètre*}

Cette syntaxe est remplacée par la longueur de la chaîne de caractères contenue dans *paramètre*. Ce dernier peut être une variable, un paramètre spécial ou un paramètre de position.

```
$ echo $PWD
/home/sanchis
$ echo ${#PWD}
13   => longueur de la chaîne /home/sanchis
$
$ set "au revoir"
$ echo ${#1}
9   => la valeur de $1 étant au revoir, sa longueur est 9
$
$ ls >/dev/null
$
$ echo ${#?}
1   => contenue dans $?, la valeur du code de retour de ls
$   => est 0, par conséquent la longueur est 1
```

Modificateurs de chaînes

Les modificateurs de chaînes permettent la suppression d'une sous-chaîne de caractères correspondant à un modèle exprimé à l'aide de caractères ou d'expressions génériques.

- Suppression de la plus courte sous-chaîne à gauche

Syntaxe : ${*paramètre#modèle*}

```
$ echo $PWD
/home/sanchis
$
$ echo ${PWD#*/}
home/sanchis      => le premier caractère / a été supprimé
$
$ set "12a34a"
$
$ echo ${1#*a}
34a               => suppression de la sous-chaîne 12a
$
```

- Suppression de la plus longue sous-chaîne à gauche

Syntaxe : ${*paramètre##modèle*}

```
$ echo $PWD
/home/sanchis
$
$ echo ${PWD##*/}
sanchis   => suppression de la plus longue sous-chaîne à gauche se
             terminant par le caractère /
$
$ set 12a34ab
$
$ echo ${1##*a}
b
$
```

- Suppression de la plus courte sous-chaîne à droite

Syntaxe : ${*paramètre*%*modèle*}

```
$ echo $PWD
/home/sanchis
$ echo ${PWD%/*}
/home
$
```

- Suppression de la plus longue sous-chaîne à droite

Syntaxe : ${*paramètre*%%*modèle*}

```
$ eleve="Pierre Dupont::12:10::15:9"
$
$ echo ${eleve%%:*}
Pierre Dupont
$
```

La variable *eleve* contient les prénom, nom et diverses notes d'un élève. Les différents champs sont séparés par un caractère deux-points. Il peut manquer des notes à un élève (cela se caractérise par un champ vide).

Extraction de sous-chaînes

- ${*paramètre*:*ind*} : extrait de la valeur de paramètre la sous-chaîne débutant à l'indice *ind*. La valeur de paramètre n'est pas modifiée.

Attention : l'indice du premier caractère d'une chaîne est 0

```
ch="abcdefghijk"
      01234567..10

echo ${ch:3} affiche defghijk
```

- ${*paramètre*:*ind*:*nb*} : extrait *nb* caractères à partir de l'indice *ind*.

```
ch="abcdefghijk"
    01234567..10

$ echo ${ch:8:2}
ij
$ set ABCDEFGH
$
$ echo ${1:4:3}
EFG
$
```

Remplacement de sous-chaînes

- ${*paramètre*/*mod*/*ch*}

Bash recherche dans la valeur de *paramètre* la plus longue sous-chaîne satisfaisant le modèle *mod* puis remplace cette sous-chaîne par la chaîne *ch*. Seule la première sous-chaîne trouvée est remplacée. La valeur de paramètre n'est pas modifiée. Caractères et expressions génériques peuvent être présents dans *mod*.

Ce mécanisme de remplacement comprend plusieurs aspects :

1. Remplacement de la première occurrence

```
$ v=totito
$ echo ${v/to/lo}
lotito
$
```

La valeur de la variable *v* (*totito*) contient deux occurrences du modèle *to*. Seule la première occurrence est remplacée par la chaîne *lo*.

```
$ set topo
$ echo $1
topo
$
$ echo ${1/o/i}
tipo
$
```

2. Remplacement de la plus longue sous-chaîne

```
$ v=abcfefg
$ v1=${v/b*f/toto}    => utilisation du caractère générique *
$ echo $v1
atotog
$
```

Deux sous-chaînes de v satisfont le modèle *b*f* : *bcf* et *bcfef*

C'est la plus longue qui est remplacée par *toto*.

- ${*paramètre*//*mod*/*ch*}

Contrairement à la syntaxe précédente, toutes les occurrences (et non seulement la première) satisfaisant le modèle *mod* sont remplacées par la chaîne *ch*

```
$ var=tobatoba
$ echo ${var//to/tou}
toubatouba
$
$ set topo
$ echo $1
topo
$
$ echo ${1//o/i}
tipi
$
```

- ${*paramètre*//*mod*/}

Lorsque la chaîne *ch* est absente, la première ou toutes les occurrences (suivant la syntaxe utilisée) sont supprimées

```
$ v=123azer45ty
$ shopt -s extglob
$ echo ${v//+([[:lower:]])/}
12345
$
```

L'expression générique *+([[:lower:]]* désigne la plus longue suite non vide de minuscules. La syntaxe utilisée signifie que toutes les occurrences doivent être traitées : la variable v contient deux occurrences (*azer* et *ty*). Le traitement à effectuer est la suppression.

Il est possible de préciser si l'on souhaite l'occurrence cherchée en début de chaîne de *paramètre* (syntaxe à utiliser : *#mod*) ou bien en fin de chaîne (*%mod*).

```
$ v=automoto
$ echo ${v/#aut/vel}
velomoto
$
$ v=automoto
$ echo ${v/%to/teur}
automoteur
$
```

3. Les tableaux

Moins utilisés que les *chaînes de caractères* ou les *entiers*, bash intègre également les tableaux monodimensionnels.

Définition et initialisation d'un tableau

Pour créer un tableau, on utilise généralement l'option -a (comme *array*) de la commande interne declare :

declare -a *nomtab ...*

Le tableau *nomtab* est simplement créé mais ne contient aucune valeur : le tableau est défini mais n'est pas initialisé.

- Pour connaître les tableaux définis : declare -a

```
$ declare -a
declare -a BASH_ARGC='()'
declare -a BASH_ARGV='()'
declare -a BASH_LINENO='()'
declare -a BASH_SOURCE='()'
declare -ar BASH_VERSINFO='([0]="3" [1]="1" [2]="17" [3]="1" [4]="release"
[5]="i486-pc-linux-gnu")'
declare -a DIRSTACK='()'
declare -a FUNCNAME='()'
declare -a GROUPS='()'
declare -a PIPESTATUS='([0]="0")'
$
```

- Pour définir et initialiser un tableau : declare -a *nomtab*=(*val0 val1 ...*)

Comme en langage C, l'indice d'un tableau débute toujours à 0 et sa valeur maximale est celle du plus grand entier positif représentable dans ce langage (bash a été écrit en C). L'indice peut être une expression arithmétique.

- Pour désigner un élément d'un tableau, on utilise la syntaxe : *nomtab*[*indice*]

```
$ declare -a tab => définition du tableau tab
$
$ read tab[1] tab[3]
coucou bonjour
$
$ tab[0]=hello
$
```

Il n'est pas obligatoire d'utiliser la commande interne declare pour créer un tableau, il suffit d'initialiser un de ses éléments :

```
$ array[3]=bonsoir    => création du tableau array avec
$                     => initialisation de l'élément d'indice 3
```

Trois autres syntaxes sont également utilisables pour initialiser globalement un tableau :

nomtab=(*val0 val1 ...*)

nomtab=([*indice*]=*val ...*)

```
$ arr=([1]=coucou [3]=hello)
$
```

l'option -a de la commande interne read ou readonly :

```
$ read -a tabmess
bonjour tout le monde
$
```

Valeur d'un élément d'un tableau

On obtient la valeur d'un élément d'un tableau à l'aide la syntaxe : ${*nomtab*[*indice*]}

bash calcule d'abord la valeur de l'indice puis l'élément du tableau est remplacé par sa valeur.

Il est possible d'utiliser toute expression arithmétique valide de la commande interne ((pour calculer l'indice d'un élément.

```
$ echo ${tabmess[1]}
tout
$
$ echo ${tabmess[RANDOM%4]} # ou bien ${tabmess[$((RANDOM%4))]}
monde
$
$ echo ${tabmess[1**2+1]}
le
$
```

Pour obtenir la longueur d'un élément d'un tableau : ${*#nomtab*[*indice*]}

```
$ echo ${tabmess[0]}
bonjour
$
$ echo ${#tabmess[0]}
7   => longueur de la chaîne bonjour
$
```

Lorsqu'un tableau sans indice est présent dans une chaîne de caractères ou une expression, bash utilise l'élément d'indice 0.

```
$ echo $tabmess
bonjour
$
```

Réciproquement, une variable non préalablement définie comme tableau peut être interprétée comme un tableau.

```
$ var=bonjour
$ echo ${var[0]}     => var est interprétée comme un tableau à un seul
élément
bonjour              => d'indice 0
$ var=( coucou ${var[0]} )
$ echo ${var[1]}     => var est devenu un véritable tableau
bonjour
$
```

Caractéristiques d'un tableau

Le nombre d'éléments d'un tableau est désigné par : ${*#nomtab*[*]}

Seuls les éléments initialisés sont comptés.

```
$ echo ${#arr[*]}
2
$
```

Tous les éléments d'un tableau sont accessibles à l'aide de la notation : ${*nomtab*[*]}

Seuls les éléments initialisés sont affichés.

```
$ echo ${arr[*]}
coucou hello
$
```

Pour obtenir la liste de tous les indices conduisant à un élément défini d'un tableau, on utilise la syntaxe : ${!*nomtab*[*]}

```
$ arr=([1]=coucou bonjour [5]=hello)
$
$ echo ${!arr[*]}
1 2 5
$
```

L'intérêt d'utiliser cette syntaxe est qu'elle permet de ne traiter que les éléments définis d'un tableau « à trous ».

```
$ for i in ${!arr[*]}
> do
>   echo "$i  => ${arr[i]}" => dans l'expression ${arr[i]}, bash interprète
> done   => directement i comme un entier
1  => coucou
2  => bonjour
5  => hello
$
```

Pour ajouter un élément val à un tableau tab :

à la fin : *tab*[${#*tab*[*]}]=*val*

en tête : *tab*=(*val* ${*tab*[*]})

```
$ tab=( un deux trois )
$ echo ${#tab[*]}
3
$ tab[${#tab[*]}]=fin      =>ajout en fin de tableau
$ echo ${tab[*]}
un deux trois fin
$ tab=( debut ${tab[*]} )  =>ajout en tête de tableau
$ echo ${tab[*]}
debut un deux trois fin
$
```

Si l'on souhaite créer un tableau d'entiers on utilise la commande declare −ai. L'ordre des options n'a aucune importance.

```
$ declare -ai tabent=( 2 45 -2 )
$
```

Pour créer un tableau en lecture seule, on utilise les options −ra :

```
$ declare -ra tabconst=( bonjour coucou salut )    => tableau en lecture
seule
$
$ tabconst[1]=ciao
-bash: tabconst: readonly variable
$
$ declare -air tabInt=( 34 56 )   => tableau (d'entiers) en lecture seule
$ echo $(( tabInt[1] +10 ))
66
$ (( tabInt[1] += 10 ))
-bash: tabInt: readonly variable
```

Suppression d'un tableau

Pour supprimer un tableau ou élément d'un tableau, on utilise la commande interne unset.

Suppression d'un tableau : unset *nomtab* … Suppression d'un élément d'un tableau : unset *nomtab*[*indice*] …