Shell(s) et langage(s) de script : Éléments de syntaxe et de grammaire

Formation continue — Université Lille 1
Pour toutes remarques : Alexandre.Sedoglavic@univ-lille1.fr

Licence trimestre 3 — 2008-09

Terminologie

Les termes suivants seront utilisés dans ce cours :

- un *blanc* correspond aux caractères ASCII 32 (espace) et 9 (tabulation) ;
- un *identificateur* est un nom composé de caractères alphanumériques ;
- un *métacaractère* du shell sert de séparateur aux identificateurs ; c'est soit un blanc, soit un des caractères ASCII suivants : `| & ; ( ) < >` ;
- un *opérateur de contrôle* est soit un retour chariot, soit un des symboles suivants :`|| & && ; ;; ( ) |` ;
- un lexème (token en anglais) est une suite fini de caractères formant une unité lexicale par le shell ;
- un *mot réservé* est un des mots suivants :

```
! case  do done elif else esac fi for function
if in select then until  while  {  } time [[ ]]
```

Typage et affectation

Le shell n'utilise que des chaînes de caractères (suite finies d'octets se terminant par l'octet 0). (Par exemple, les entiers sont stockés comme des chaînes de caractères).

On peut affecter une chaîne à une variable par l'opérateur = : `NOMVAR=<chaine>`. (Attention, il n'y a pas d'espace séparateur autour du = et bien que le bash soit sensible à la casse, on utilise souvent les majuscules pour les variables).

L'interpréteur substitute la chaîne de caractères correspondante à chaque évaluation de `NOMVAR`.

```
% a=hello ; b=UNIX ; c=a ; d = b
bash: d: command not found
% a ; $a ; echo $a $b $c d
bash: a: command not found
bash: hello: command not found
hello UNIX a d
```

Évaluation d'expression

Une évaluation de paramètre est de la forme : `${ expression }` dont la forme la plus simple est `${parameter}` (dans ce cas, les accolades sont optionnelles si le paramètre n'est pas un entier supérieur à 9) ; la valeur de `parameter` se substitue à `${parameter}` dans le processus d'interprétation du shell.

L'expression `${#expr}` retourne le nombre de caractères de l'évaluation de `expr`. Si l'expression est `*` ou `@`, le résultat n'est pas spécifié.

Pour illustrer ces points, notons que le bash est un filtre qui retourne les évaluations des expressions sur ses sorties standards :

```
% bash 2> /tmp/foo
$LANG ; quit
% cat /tmp/foo
bash: line 1: en_US.UTF-8: command not found
```

L'évaluation de paramètre peut être modifiée en utilisant un des formats suivant où `word` représente une expression :

- `${parameter:=word}` : si `parameter` est non affecté ou null, l'évaluation de `word` est affectée au paramètre (utilisable uniquement sur les variables). Sinon, la valeur du paramètre est retournée sur la sortie standard ;
- `${parameter:-word}` : si `parameter` est non affecté ou null, l'évaluation de `word` est retournée sur la sortie standard. Sinon, la valeur du paramètre est retournée sur la sortie standard ;
- `${parameter:+word}` : si `parameter` est non affecté ou null, null est retourné. Sinon l'évaluation de `word` est retournée sur la sortie standard.
- `${parameter:?[word]}` : si `parameter` est non affecté ou null, l'évaluation de `word` est retournée sur la sortie d'erreur (dans ce cas l'expression provoque la terminaison d'un script la contenant). Sinon l'évaluation du paramètre est retournée sur la sortie standard.

L'usage du : provoque un test sur l'affectation ou la nullité du paramètre. Sans :, seule l'affectation du paramètre est testée. La table suivante permet de résumer ce point :

prm	affecté et non null	affecté mais null	non affecté
`${prm:-wrd}`	retourne prm	retourne wrd	retourne wrd
`${prm-wrd}`	retourne prm	retourne null	retourne wrd
`${prm:=wrd}`	retourne prm	affecte wrd	affecte wrd
`${prm=wrd}`	retourne prm	retourne null	affecte wrd
`${prm:?wrd}`	retourne prm	erreur, exit	erreur, exit
`${prm?wrd}`	retourne prm	retourne null	erreur, exit
`${prm:+wrd}`	retourne wrd	retourne null	retourne null
`${prm+wrd}`	retourne wrd	retourne wrd	retourne null

Étant donné `regexp` une expression régulière et un paramètre `param` s'évaluant en une chaîne de caractères, `regexp` définit dans `${param}` des suffixes et des préfixes.

- `${parameter%regexp}` enlève le plus court suffixe définit par `regexp` dans l'évaluation de `parameter` ;
- `${parameter%%regexp}` enlève le plus long suffixe définit par `regexp` dans l'évaluation de `parameter` ;
- `${parameter#regexp}` enlève le plus court préfixe définit par `regexp` dans l'évaluation de `parameter` ;
- `${parameter##regexp}` enlève le plus long préfixe définit par `regexp` dans l'évaluation de `parameter` ;

Illustrons ces points par des exemples :

```
% MAVAR=LaVieEstVraimentBelle
% echo ${MAVAR%a*}
LaVieEstVr
% echo ${MAVAR%%a*}
L
% echo ${MAVAR#*a}
VieEstVraimentBelle
% echo ${MAVAR##*a}
imentBelle
%  echo ${MAVAR#a*} # ne definit aucun pr\'efixe
LaVieEstVraimentBelle
%  echo ${MAVAR%*a} # ne definit aucun suffixe
LaVieEstVraimentBelle
```

Calcul numérique

Le shell dispose d'un mécanisme premettant :

- de convertir une chaîne de caractères en entier ;
- d'effectuer des opérations numériques classiques ; et
- de reconvertir le résultat en chaîne de caractères.

Pour ce faire, on utilise un double-paranthésage comme suit : `$((expression))`

Par exemple :

```
% MONINT=10
% MONINT=$(( (100*MONINT-10)/100 ))
% echo $MONINT
9
```

Substitution d'une commande par sa sortie standard

La substitution de commande à la sortie d'une commande de se substituer au nom de commande. Pour ce faire, on utilise

`$(expression)` (resp. `‘expression‘`)

La commande `expression` est exécutée dans un sous-shell et la chaîne de caractères qui en résulte sur la sortie standard est substituée à `$(expression)` (resp. `‘expression‘`). (Seule la partie précédant un retour chariot est gardée).

```
% date
Sun Jan 14 10:53:25 CET 2007
% ‘date‘
bash: Sun: command not found
% ls /bin | grep date
date
% ‘ls /bin | grep date‘
Sun Jan 14 10:54:07 CET 2007
```

L'évaluation de la variable `$HOME` se substitue au tilde (~).

Les accents aigus (' ') bloquent l'évaluation de l'expression les entourant (cette expression ne peut contenir d'accent aigu).

Les guillements (" ") bloquent l'évaluation de l'expression les entourant à l'exception des symboles dollar, accents graves et contre-oblique :

```
% VAR=date; echo 'la $VAR est $(`ls /bin|grep date`)'
la $VAR est $(`ls /bin | grep date`)
% VAR=date ;  echo "la $VAR est $(ls /bin | grep date)"
la date est date
 % VAR=date ;  echo "la $VAR est ``ls /bin | grep date``"
la date est ls /bin | grep date
 % VAR=date ;  echo "la $VAR est $(`ls /bin | grep date`)"
la date est Sun Jan 14 11:06:39 CET 2007
% VAR=date ;  echo "la $VAR est `$(ls /bin | grep date)`"
la date est Sun Jan 14 11:06:39 CET 2007
% VAR=date; echo "la $VAR est $($(ls /bin|grep date))"
la date est Sun Jan 14 10:59:31 CET 2007
```

La commande alias

Un alias établit une correspondance entre un nom de commande et une chaîne de caractères de remplacement.
Bash maintient une table de correspondance d'alias et déclenche la substitution lorsqu'il repère une des entrées sur la ligne de commande.
Sans argument, la commande alias retourne la liste des alias déclarés.

```
% alias
alias gcc='gcc -Wall -ansi -pedantic'
alias h='history'
```

La commande alias (resp. unalias) ajoute (resp. supprime) un alias de la table de correspondance.

```
% alias monos='uname -sr'
% monos
Linux 2.6.11-1.1369_FC4
% unalias monos
% monos
bash: monos: command not found
```

La commande test

Le filtre %test expr permet d'obtenir de savoir si l'expression expr est vrai ou fausse.

La valeur de retour est :

- 0 si l'évaluation de l'expression est vraie ;
- 1 sinon.

Cette convention est l'inverse de celle utilisée en C.

Cette commande possède une forme équivalente [expr].

Voir la documentation en ligne pour une description complète.

```
% echo $LANG ; test $LANG = fr  ; echo $?
en_US.UTF-8
1
%  echo $USER ; [ $USER = sedoglavic ]  ; echo $?
sedoglavic
0
```

Instruction

Une *instruction simple* est une expression constituée d'affectation, de commande, de redirection terminée par un opérateur de contrôle (`;` `&` `&&` `||`).

Le shell dispose de plusieurs *instruction composée* permettant de contrôler l'exécution d'une suite d'instructions simples. Ainsi,

- chaque instruction composée est définie par un mot réservé ou un opérateur à son début et un autre à sa fin ;
- ces instructions peuvent être suivies de redirection(s) sur la même ligne que le mot réservé ou l'opérateur les terminant. Ces redirections s'appliquent alors à l'ensemble des commandes constituant l'instruction composée qui ne modifient pas explicitement ces redirections.

Une instruction simple est la plus simple des instructions composée :-)
Ainsi, les instructions composées les plus simples sont des groupements d'instructions simples :

- entre paranthèses, celles-ci sont exécutées dans un sous-shell :

```
% (ls /nexistepas ; exit )
ls: /nexistepas: No such file or directory
```

- entre accolades, celles-ci sont exécutées dans le shell courant :

```
% { ls /nexistepas ; echo $0 ;}
ls: /nexistepas: No such file or directory
bash
```

  Remarquez bien le séparateur après l'accolade ouvrante et le point-virgule avant l'accolade fermante.

Conditionnelle simple

Le mot-clef if exécute une instruction composée et utilise son code de retour pour contrôler l'exécution d'autres instructions composées. On l'utilise comme suit :

```
if      instruction composée test 0    then
            instruction composée 1
                <optionnel>
elif    instruction composée test 1    then
            instruction composée 2
                ...
else    instruction composée finale
                < \optionnel>
fi
```

- Si le code de retour de l'instruction composée test est 0, l'instruction composée 1 est exécuté et notre conditionnelle simple se termine.
- Sinon les instructions composées servant de test aux elif son exécutées jusqu'à ce que la première d'entre elles retourne 0 ; l'instruction composée suivant le then correspondant est exécutée et la conditionnelle se termine.
- Le code de retour de la conditionnelle est celui de l'instruction composée (suivant un then) exécutée ou 0 si aucune n'a été exécutée.

Conditionnelle à choix multiples

La syntaxe de cette instructions est :

```
case          expr                              in
regexpr 1)    instruction composée 1 ; ;
              <optionnel>
regexpr n)    instruction composée n ; ;
              < \optionnel>
esac
```

L'instruction conditionnelle à choix multiples exécute l'instruction composée correspondant à la première expression régulière satisfaite par la chaîne de caractères résultant de l'évaluation de l'expression expr.

```
% echo $LANG
en_US.UTF-8
%  case $LANG in fr) echo "francais" ;; \
>  *US*) echo "US english" ;; esac
US english
```

Une fois une expression régulière satisfaite, les instructions composées correpondantes aux expressions régulières suivantes ne sont pas exécutées (contrairement au C).

Le code de retour d'une instruction conditionnelle à choix multiple est 0 si une expression régulière est satisfaite ; dans le cas contraire, le code de retour de cette instruction est celui de la dernière instruction composée exécutée.

Itération énumérative

La boucle for exécute une instruction composée pour chaque membre d'une liste d'expressions. Cette instruction composée est délimitée par les mots-clefs do et done.

for var <optionnel>in expr < \optionnel> do
instruction composée
done

- L'expression expr produit une liste d'expression. La variable var est affectée à chacune de ces expressions et l'instruction composée est chaque fois exécutée.
- Si la liste d'expression est vide, l'instruction composée n'est pas exécutée.
- Ne pas utiliser in expr est équivalent à in "$@".
- Le code de retour ce type instruction d'itération est celui de la dernière instruction composée exécutée. S'il n'y a pas de membre dans la liste que for énumère, le code de retour est 0.

```
% for MAVAR in un deux trois ;\
>  do \
> echo $MAVAR ; \
> done ; \
> echo "je sais compter jusqu'\'a trois" ;
un
deux
trois
je sais compter jusqu''a trois
% for VAR in `ls -d /etc/r*` ; \
> do \
> if [ -d $VAR ] ; then \
> echo "$VAR est un repertoire" ;\
> else \
> echo "$VAR est un fichier" ;\
> fi ;\
> done  >> /tmp/foo
% head -2 /tmp/foo
/etc/racoon est un repertoire
/etc/rc est un fichier
```

Itération conditionnelle

La syntaxe de cette instruction composée est :

```
while   instruction composée 1   do
        instruction composée 2
done
```

Elle exécute l'instruction composée 2 tant que l'instruction composée 1 retourne le code de retour 0.

Le code de retour de cette instruction composée est celui de la dernière instruction composée 2 exécutée et 0 si aucune ne l'est.

Itération conditionnelle (inverse)

La syntaxe de cette instruction composée est :

```
until   instruction composée 1   do
        instruction composée 2
done
```

Elle exécute l'instruction composée 2 tant que l'instruction composée 1 retourne le code de retour 1 (comportement inverse du while).

Le code de retour de cette instruction composée est celui de la dernière instruction composée 2 exécutée et 0 si aucune ne l'est.

```
% pwd
/local/home/sedoglavic/Enseignement/Shell/Cours
% while [ `pwd` != / ] ; do cd .. ; done ;
% # on peut bien sur faire directement % cd /
% pwd
/
% until [ `pwd` = / ] ; do cd .. ; done ;
cassandre % pwd
/
```

Une fonction est une instruction composée portant un nom et définie par l'utilisateur. Elle peut être utilisée par l'invocation de son nom et dispose de ses propres paramètres de position. Pour définir une fonction, on utilise la syntaxe : NomDeFonction() Instruction composée redirection.

L'évaluation des instructions (ainsi que les redirections et l'affectation de variable) ne se fait pas au moment de la définition de la fonction mais à celui de son exécution.

La fonction est exécutée à l'invocation de son nom dans le shell en tant que commande. Les arguments de cette commande deviennent les paramètres de position de la fonction.

Si la commande interne return est exécutée dans la fonction, celle-ci se termine.

Le code de retour de la definition d'une fonction est 0 si sa déclaration est correcte et non nulle sinon. Le code de retour de l'exécution de la fonction est celui de la dernière instruction exécutée en son sein.

```
% moncat(){ if [ -r $1 ]; then cat $1; fi } 1> /tmp/foo
% moncat 02/02.tex
```

Un exemple plus compliqué :

```
tarview() {
    echo -n "Displaying contents of $1 "
    if [ ${1##*.} = tar ]
    then
        echo "(uncompressed tar)"
        tar tvf $1
    elif [ ${1##*.} = gz ]
    then
        echo "(gzip-compressed tar)"
        tar tzvf $1
    elif [ ${1##*.} = bz2 ]
    then
        echo "(bzip2-compressed tar)"
        cat $1 | bzip2 -d | tar tvf -
    fi
}
% tarview shorten.tar.gz
```