Système d'Exploitation Linux et Services Réseaux

(LP Métiers du Net)

R.J.

Automne 2013

Notions et mécanismes de bases

- Le shell est l'interface de commande du système.
- Il peut travailler en deux modes:
  – en mode interactif, i.e., l'utilisateur tape une commande Linux au niveau du terminal en attendant un résultat donné,
  – en mode programmé, i.e., l'utilisateur exécute un fichier appelé "script" qui contient un ensemble de commandes Linux.
- Lors du lancement d'une commande, le shell effectue les opérations suivantes:
  – analyse de la ligne de commande (il faut faire attention aux caractères spéciaux, aux espaces et à la casse),
  – recherche du fichier contenant le programme de la commande,
  – création du processus qui va exécuter ce programme,
  – chargement du programme et passage des arguments.

Les différents Shells

- **Le Bourne Shell** : C'était le premier Shell qui a été écrit pour Unix par S.R. Bourne, d'où son nom. C'est le Shell de base d'Unix.
- **Le C-shell** : Le C-shell est issu de l'Université de Berkeley. Une partie de sa syntaxe a été influencée par le langage C, à qui il doit son nom. Il est le précurseur de nouveautés qui ont été ensuite intégrées au Korn Shell. Dans la même famille, on trouve le Tenex C shell (tcsh) qui est le Shell par défaut des systèmes du monde BSD.
- **Le Korn Shell** : Le Korn Shell est un interpréteur de commandes qui a été disponible en standard à partir d'UNIX System V.4. Il est doté d'un langage de programmation compatible avec le Bourne Shell. Il a repris et amélioré les innovations du C-Shell, et dispose de nombreuses extensions, permettant d'accélérer la saisie des commandes. On trouve sous Gnu/Linux une version libre du Korn Shell appelée pdksh.

Les différents Shells

- **Le Bourne Again Shell (Bash)** : Le Bash est la version GNU du Bourne Shell. Il incorpore de nombreuses fonctionnalités présentes dans d'autres Shells, comme le Korn Shell ou le C-shell. C'est le Shell par défaut de GNU/Linux.
- Il existe d'autres Shells qui ont été développés, mais le Bash reste le Shell le plus répandu.

La ligne de commande

- La ligne de commande va du prompt du Shell, jusqu'au retour chariot.
- Le premier mot est le nom d'une commande Linux ou d'un fichier exécutable.
- La commande peut être suivie par des options et/ou des arguments, qui sont séparés par des espaces.
- La ligne de commande ne s'exécute qu'une fois qu'on a appuyé sur la touche entrée (retour chariot).
- État de sortie d'une commande:
  – en cas d'une terminaison correcte, la sortie est 0,
  – en cas d'une terminaison incorrecte, la sortie est différente de 0.

Les alias

- Les alias sont des pseudo-commandes qui servent à redéfinir le comportement de certaines commandes ou à en créer des nouvelles.
- On peut créer des alias avec la commande Linux alias.
- Exemples:
  $**alias rm='rm -i'** : On redéfinit la commande rm en lui ajoutant l'option "-i" qui signifie interactif (l'utilisateur doit ainsi valider la suppression d'un fichier avant son exécution).
  $**alias my-ls='ls -rl'** : On crée une commande my-ls qui fait un ls avec les options "-rl", pour une exécution récursive avec un affichage ligne par ligne.
  $**alias count-users='who | wc -l'** : On crée une commande count-users qui affiche le nombre d'utilisateurs système connéctés.

Les variables Shell

- Il y a deux types de variables:
  – Les variables créées et maintenues par le système,
  – les variables créées par l'utilisateur.
- Création d'une variable : (au niveau du terminal) VAR=valeur
  Création d'une variable qui s'appelle VAR et qui contient la chaîne de caractères valeur.
- Attention : Il n'y a pas d'espaces avant et après le signe d'égalité.
- Notons bien que la valeur d'une variable est enregistrée sous format d'une chaîne de caractères.
- L'interprétation numérique d'une chaîne de caractères numériques se fait via les commande test et expr.

- Pour créer une chaîne vide : x= ou x=" ou x="".
- L'accès à la valeur d'une variable se fait via la commande echo : echo $VAR.
- Faites attention à ne pas oublier de mettre le caractères $, sinon ça sera le nom de la variable qui sera affichée, non sa valeur.
  Exemple : echo la valeur de VAR est $VAR
  => affichage du message : la valeur de "le nom de la variable" est "la valeur de la variable".

Les variables Shell prédéfinies

- Ces variable sont appelées des ”variables d’environnement”.
- La commande set permet d’afficher ces variables. Elle permet aussi de modifier les paramètres du Shell.
- Une des variables d’environnement est la variable PATH.
  Elle regroupe les noms des répertoires qui contiennent l’ensemble des commandes qu’on peut exécuter.

Variables Shell locales et globales

- Toute variable créée est par défaut une variable locale, i.e., elle n'est connue que dans le Shell où elle a été créée.
- Une variable de même nom qui sera créée dans un sous-Shell (du Shell actuel) sera une variable distincte. La valeur de la même variable dans le Shell père ne sera pas modifiée.
- => Tout Shell a ses propres variables locales.
- Exemple :

```
$chiffre=1              # création d'une variable chiffre qui contient la valeur 1
$echo $chiffre          # affichage de la valeur de la variable chiffre
1
$sh                     # lancement d'un nouveau Shell
        $echo $chiffre
                        # la variable chiffre n'est pas définie dans le sous-Shell
        $chiffre=9
        $echo $chiffre
        9
        $ctrl+D         # fermeture du Shell dernièrement créé
$echo $chiffre
1
```

Variables Shell locales et globales

- La commande export permet de rendre une variable locale une variable globale, i.e., tous les sous-Shell vont posséder une copie de cette variable.
- Attention : Les variables peuvent être exportées vers les sous-Shell, mais ne peuvent pas être renvoyées vers le Shell parent.
- Exemple :

```
$chiffre=1
$echo $chiffre
1
$export chiffre
1
$sh
        $echo $chiffre
        1
        $chiffre=9
        $echo $chiffre
        9
        $ctrl+D
$echo $chiffre

1
```

Les scripts Shell

- Un script Shell est un fichier exécutable qui regroupe un ensemble de commandes Linux ainsi qu'éventuellement des variables.
- Un script Shell permet d'automatiser une tâche donnée.
- Les scripts Shell font souvent appel aux commandes echo et read pour interagir avec l'utilisateur ; echo pour afficher des messages à l'écran et read pour lire des entrées à partir du clavier.
  Exemple :
  #!/bin/bash
  echo Entrez un chiffre
  read chiffre
  echo La valeur qui a été saisie est $chiffre

- Notons bien qu'on peut lire une liste de variables d'un seul coup.
  Exemple : read nom prenom
- Faites toujours attention à ajouter le droit d'exécution aux scripts Shell créés, pour les rendre exécutables.

Arguments

- Un script Shell peut être lancé avec des arguments : $./mon-script argument1 argument2 argument3 . . .
- Dans les fichiers scripts, les arguments sont référencés et utilisables via leur position dans la liste des arguments donnés :
  $1 <=> 1er argument,
  $2 <=> 2ème argument,
  . . .
  $N <=> Nième argument.
- Remarque : La commande set permet d'attribuer des valeurs aux paramètres positionnels $1 $2 . . .
  Exemple :
  $set Maroc Malaisie Canada
  $echo $1 $2 $3
  Maroc Malaisie Canada

Variables spéciales

- $0 : c'est le nom de la commande ou du script Shell qui fait appel aux arguments,
- $* : contient la liste des arguments donnés,
- $# : c'est le nombre d'arguments (le nombre de paramètres positionnels),
- $? : c'est l'état de sortie de la dernière commande exécutée ;
  la valeur retournée est 0 si la commande s'est bien terminée, et elle est différente de 0 sinon,
- $$: c'est le PID du Shell courant,
- $! : c'est le PID du dernier processus lancé en background.

- Les pipes et les redirections peuvent être utilisés naturellement à l'intérieur des scripts Shell, mais ils peuvent aussi être utilisés à l'extérieur des scripts Shell.
- Lorsqu'une chaîne de caractères est entourée par ` ` elle est interprétée comme étant une commande et elle est remplacée par son résultat.
  Exemple:
  $echo date
  date
  $echo `date`
  Wed Dec 11 11:05:58 UTC 2013
  Attention : Il s'agit du caractère ` non du caractère ' .
- On peut ajouter des commentaires dans le script Shell, en les précédant par le caractère # .

Mécanismes d'échappement

- \ négation d'un métacaractère, i.e., son interprétation comme étant un simple caractère textuel,
- '...' négation de tous les métacaractères qui se trouvent entre les simples quottes,
- "..." interprétation des métacaractères (comme caractères spéciaux).

La programmation Shell

Boucle for

Syntaxe:
for variable in valeur1 valeur2 . . .
do
commandes
done

Boucle for

Exemple1:
for i in Linux Réseaux-Informatiques Imagerie-Numérique
do
echo On a ensemble une séance de $i.
done

$./Exemple1
On a ensemble une séance de Linux.
On a ensemble une séance de Réseaux-Informatiques.
On a ensemble une séance de Imagerie-Numérique.

Boucle for

Exemple2 (passage d'arguments):
for i in $*
do
echo On a ensemble une séance de $i.
done

$./Exemple2 Linux Réseaux-Informatiques Imagerie-Numérique
On a ensemble une séance de Linux.
On a ensemble une séance de Réseaux-Informatiques.
On a ensemble une séance de Imagerie-Numérique.

Exemple3 (exécution de la boucle for directement depuis l'invité de commandes):
$for i in 1 2 3 ; do echo $i ; done
1
2
3

Boucle for

- L'instruction shift permet d'effectuer un décalage à gauche des paramètres positionnels:

$./script	arg1	arg2	arg3		arg1	arg2	arg3
$0	$1	$2	$3	shift		$1	$2

- shift peut être combinée avec for.
- Exemple4 (utilisation de for avec shift):

```
# envoi d'un message à plusieurs contacts
message=$1
shift
for contact in $*
do
mail $contact < $message
done
```

$./Exemple4 contenu-message Hassan Salma Aziz Layla

Boucle for

Exemple5 (utilisation de for avec génération de noms) :

```
# copie de fichiers .txt
for F in *.txt
do
cp $F repertoire-fichiers-txt
done
```

Boucle for

Exemple6 (utilisation de for en interaction avec l'utilisateur):

```
echo Entrez les noms des invités :
read invites
echo Il est convié à la réception :
for i in $invites
do
echo $i
done
```

```
$./Exemple6
Entrez les noms des invités :
Hassan Salma Aziz Layla
Il est convié à la réception :
Hassan
Salma
Aziz
Layla
```

Boucle for

Exemple7 (utilisation de for en combinaison avec la substitution de commandes):

```
for ligne in `cat F`
do
echo $ligne
done
```

Boucle while

- Syntaxe:
  while commande-de-controle
  do
  commandes
  done
- Au début de chaque itération, la commande ”commande-de-controle” est exécutée:
  – si elle retourne au Shell la valeur 0 (exit(0)), i.e., la commande s’est bien executée, alors le corps de la boucle sera exécuté,
  – sinon, on sort de la boucle.

  Ainsi, la boucle while s’exécute jusqu’à ce que commande-de-controle ait un état de sortie différent de 0, i.e., arrêt sur une erreur d’exécution de commande-de-controle.
- Remarque : True peut être utilisée comme commande-de-controle, et dans ce cas la on peut quitter la boucle par la commande break.

Boucle while

Exemple:

```
# recherche du premier fichier ne contenant pas un mot donné
# utilisation : ./Exemple mot liste de fichiers
mot=$1
shift
while grep $mot $1 > F
do
shift
done
echo $1 est le premier fichier qui ne contient pas $mot
```

Boucle until

- Syntaxe:
  until commande-de-controle
  do
  commandes
  done
- Au début de chaque itération, la commande ”commande-de-controle” est exécutée:
  – si elle retourne au Shell une valeur différente de 0, alors le corps de la boucle sera exécuté,
  – sinon, on sort de la boucle.

  Ainsi, la boucle until s'exécute tant que commande-de-controle a un état de sortie différent de 0, i.e., arrêt sur un succès de commande-de-controle.

Boucle until

Exemple:

```
# recherche du premier fichier contenant un mot donné
# utilisation : ./Exemple mot liste de fichiers
mot=$1
shift
until grep $mot $1 > F
do
shift
done
echo $1 est le premier fichier qui contient $mot
```

L'instruction case

- Syntaxe:
  case valeur in
  choix1) commande1 ; commande2 . . . ;;
  choix2) commande1 ; commande2 . . . ;;
  :
  choixN) commande1 ; commande2 . . . ;;
  ∗) commande1 ; commande2 . . . ;; # le cas par défaut
  esac
- – Notons bien qu'on sépare les commandes par un seul ";" et qu'à la fin il y a deux ";",
  – notons bien aussi qu'on met un ")" après la valeur du choix,
  – notons bien également que le cas par défaut est optionnel.
- Remarque : Les commandes peuvent faire appel aux pipes et aux redirections.

L'instruction case

Exemple1:

```
echo Où se trouve les meilleures plages du Maroc
echo A - Mehdia
echo B - Oued Law
echo C - Taghazout
echo D - Dakhla
read reponse
case $reponse in
        A) echo Bonne réponse ;;
        B|C|D) echo Mauvaise réponse;;
# notons bien qu'on peut regrouper différents choix avec le symbole | qui désigne ici le OU logique
esac
```

L'instruction case

Exemple2:

```
for F in $*
do
case $F in
        *.c) rm $F ;;
        *.java) cp $F codes-sources/projet4/JAVA ;;
# notons bien qu'on peut faire appel dans les choix aux jokers
esac
done
```

L'instruction if

- Syntaxe:
  if commande-de-controle
  then
  commandes-A
  else
  commandes-B
  fi
- – Si commande-de-controle s'est bien exécutée alors on exécutera commandes-A,
  – sinon, on exécutera commandes-B.
- Remarques: Le else est optionnel.

L'instruction if

Exemple:

```
# suppression d'un fichier copie
# utilisation : ./Exemple fichier1 fichier2
if diff $1 $2
then
        rm $1 ; echo $1 est une copie de $2
else
        echo $1 et $2 sont différents
fi
```

L'instruction if

Remarque: On peut imbriquer des tests if

```
if commande-de-controle1
then
        commandes-A # exécution si succès de commande-de-controle1
elif commande-de-controle2
then
        commandes-B # exécution si échec de commande-de-controle1, et
succès après de commande-de-controle2
else
        commandes-C # exécution si échec de commande-de-controle1 et
échec après de commande-de-controle2
fi
```

L'instruction test

- L'instruction test permet d'effectuer des tests d'existence de variables et des tests de comparaison de chaînes de caractères et de comparaison numérique. L'instruction test permet aussi de tester par rapport aux caractéristiques des fichiers et des répertoires.
- Exemple de tests d'existence de variables:

```
$module=CRSW
$test $module
$echo $?            # l'état de sortie de la commande test
0                   # existence de la variable => 0
$test $matière
$echo $?
1                   # inexistence de la variable => 1
```

L'instruction test

- Tests de comparaison de chaînes de caractères:
  test $chaine vrai si la variable chaine n'est pas vide
  test -z $chaine vrai si la variable chaine est vide
  test chaine1 = chaine2 vrai si les deux chaînes sont égales
  test chaine1 ! = chaine2 vrai si les deux chaînes sont différentes

  Attention : Il faut mettre un espace avant et après les signes = et ! = .
- Exemple:
  $echo Devinez le jour de la semaine auquel je pense
  $read reponse
  $until test $reponse = jeudi
  $do
  $echo Raté
  $read reponse
  $done

  $echo Plein dans le mille

L'instruction test

- Tests de comparaison numérique:

test n1 -eq n2	vrai si les 2 valeurs numériques sont égales
test n1 -ne n2	vrai si les 2 valeurs numériques sont différentes
test n1 -gt n2	vrai si la valeur numérique de n1 est supérieure à celle de n2
test n1 -ge n2	vrai si la valeur numérique de n1 est supérieure ou égale à celle de n2
test n1 -lt n2	vrai si la valeur numérique de n1 est inférieure à celle de n2
test n1 -le n2	vrai si la valeur numérique de n1 est inférieure ou égale à celle de n2

- Exemples:

```
$test 7 = 7          # vrai (comparaison de chaînes de caractères)
$test 7 = 007        # faux (comparaison de chaînes de caractères)
$test 7 -eq 007      # vrai (comparaison numérique)
```

L'instruction test

- Tests par rapport aux caractéristiques des fichiers et des répertoires

  test -f F vrai si F est un fichier
  test -d F vrai si F est un répertoire
  test -r F vrai s'il y a le droit de lecture sur F
  test -w F vrai s'il y a le droit d'écriture sur F
  test -x F vrai s'il y a le droit x sur F
  test -s F vrai si F a une taille supérieur à 0

- Exemples:

```
$test -w .profile ; echo $?
0          # vrai
$test -w /etc/passwd ; echo $?
1          # faux
```

L'instruction test

- Tests et opérations logiques:
  ! négation
  -a ET logique
  -o OU logique
- Exemples:
  $test ! -r F -a -w F # vrai s'il n'y a pas le droit de lecture sur F et s'il y a le droit en écriture sur F
  $test N -gt M -o T -lt M # vrai si la valeur numérique de N est supérieure à celle de M ou si la valeur numérique de T est inférieure à celle de M

L'instruction test

- La forme réduite du test:

  On peut écrire les tests sous une forme réduite, en utilisant les caractères "[" "]" et espace.

- Exemples:
  [$OS = Linux] est équivalente à test $OS = Linux
  [-f fich] est équivalente à test -f fich
  [$age -gt 20 -a $age -lt 30] est équivalente à test $age -gt 20 -a $age -lt 30

- Attention : Il faut mettre un espace après "[", et un autre avant "]".
  – [$OS = Linux] est fausse,

  – [$OS = Linux] est correcte.

Calcul arithmétique avec la commande expr

- La commande expr interprète évalue et donne le résultat d'une expression arithmétique.
- Exemples:
  $expr 27 + 31
  58
  $expr 17 − 5
  12
  $expr 3 * 6
  18
  # notons bien l'utilisation du caractère \, vu que le caractère ∗ est un caractère spécial
  $expr 32 / 16
  2
- Notons bien qu'il y a des espaces avant et après les signes des opérations arithmétiques.

Évaluation d'expressions

- La commande eval sert à évaluer une expression donnée en argument.
- Exemple:
  $OS=Linux
  $SE=OS
  $echo $SE
  OS
  $eval "echo \$$SE" # \$$SE ≡ Linux
  Linux

Autres instructions

- break : pour sortir d'une boucle (par exemple pour sortir d'une boucle while True).
- continue : pour passer à l'itération suivante d'une boucle.
- exit : pour sortir d'un script Shell.

Appels récursifs

- Un script peut exécuter un autre script ; comme il peut s'appeler lui même, ce qui permet des appels récursifs.
- Exemple:

```
# calcul du factoriel N
# utilisation ./Exemple N
case $1 in
        1) echo 1 ;;
        *) P=`./Exemple \`expr $1 - 1\` ` ; echo `expr $1 \* $P` ;;
# notons bien l'utilisation du caractère \ pour interpréter les
métacaractères comme de simples caractères
esac
```

Exercices

- Créez un script Shell (SS1) qui rend exécutables ses arguments , tout en affichant des messages indiquant qu'ils sont à présent des fichiers exécutables.
- Créez un script Shell (SS2) qui affiche l'heure (sans la date) toutes les N secondes, pendant une durée totale (d'exécution du script) de M secondes. Les valeurs N et M doivent être saisies par l'utilisateur.
- Créez un script Shell (SS3) qui indique si son argument commence par une voyelle, ou par une consonne ou par un chiffre.
- Créez un script Shell (SS4) qui affiche ses arguments dans l'ordre inverse de leur écriture.
- Créez un script Shell (SS5) qui permet de supprimer des répertoires vides entrés comme arguments. Le script doit vérifier si l'argument est un répertoire et dans le cas contraire avertir l'utilisateur, puis il doit vérifier que le répertoire est vide avant de procéder à sa suppression.
- Créez un script Shell (SS6) qui calcule $f(x) = 1 + x + x^2 + \ldots + x^N$, où x et N sont deux entiers positifs saisis par l'utilisateur.

Solution - SS1

```
for i in $*
do
chmod +x $i
echo $i est a present un fichier executable.
done
```

Solution - SS2

```
echo Entrez l\'intervalle d\'affichage :
read N
echo Entrez la duree totale de l\'execution du script:
read M
iter=`expr $M / $N`
while test $iter -ge 1
do
date +%T
sleep $N
iter=`expr $iter - 1`
done
```

Solution - SS3

```
case $1 in
        [aeiou]*|[AEIOU]*) echo $1 commence par une voyelle ;; # faites attention à la présence du signe du OU logique
        [0-9]*) echo $1 commence par un chiffre ;;
        *) echo $1 commence par une consonne ;;
esac
```

Solution - SS4

```
iter=1
while test $iter -le $#
do
eval "echo $`expr $# - $iter + 1`"
iter=`expr $iter + 1`
done
```

Solution - SS5

```
for i in $*
do
if test -d $i
then
        if test `ls $i | wc -l` -eq 0
        then
                rmdir $i ; echo $i vient d\'etre supprime
        else
                echo Attention : $i n\'est pas vide.
        fi
else
        echo Attention : $i est un fichier.
fi
done
```

Solution - SS6

```
case $2 in
        0) echo 1 ;;
        1) echo `expr $1 + 1` ;;
        *) P=1
        iter=1
        while test $iter -le $2
        do
        P=`expr $P \* $1`
        iter=`expr $iter + 1`
        done
        S=`./SS6 $1 \`expr $2 - 1\` `
        echo `expr $S + $P` ;;
esac
```