Le shell : Programmation

- Le shell est doté d'un véritable langage de programmation
- Ce langage permet la réalisation de shell-scripts des plus utiles en matière d'administration
  - accès aux variables d'environnement
  - accès aux commandes du shell
  - structuration du code (conditions, boucles)
- Shell étudié : bash
- La première ligne du script permet de spécifier le shell utilisé
  - #!/bin/bash

shell-script

- Un shell-script est un fichier texte contenant une liste de commandes shell.

Exemple : UnScript.bash

```
echo bonjour
echo $LOGNAME
pwd
...
```

- Pour exécuter UnScript.bash :
  - bash UnScript.bash
  - chmod +rx UnScript.bash
  - ./UnScript.bash
  - le placer dans /usr/local/bin pour le rendre accessible

Variables d'environnement

HOME	→	le répertoire de login
LOGNAME	→	le nom de login
PS1	→	le message d'appel principal
PS2	→	le message d'appel secondaire
PATH	→	chemins de recherche des commandes

- On obtient la valeur des variables en les faisant précéder de $

```
echo $PATH
```

- Il est possible de modifier ces variables (généralement dans un fichier d'environnement)

```
PS1=" - - > "
```

- La commande env affiche les variables d'environnement

Arguments

- **Lors de l'interprétation d'une ligne de commande, les arguments sont affectés à des variables prédéfinies $0 à $9**

$cmd	arg1	arg2	arg3
↓	↓	↓	↓
$0	$1	$2	$3

- **S'il y a plus de 9 arguments, décaler : commande shift**

Variables du Shell

- pas de déclaration : toutes les variables sont de type chaînes de caractères
- l'affectation se fait par le signe =
- accès au contenu d'une variable : la faire précéder de $
- la valeur d'une variable est associée au shell qui l'a créée, et n'est pas automatiquement transmise à son fils
  - export permet d'exporter les variables vers un sous-shell
- Substitution du résultat d'une commande : var = `cmd`
  var prend pour valeur le résultat de cmd.

Le langage

- set
  - Sans argument : liste les variables (shell et environnement) et leurs valeurs.
  - Avec arguments : affectation des arguments à $1, $2...

- Variables prédéfinies

  $# → nombre d'arguments

  $* → tous les arguments

  $? → valeur de retour de la dernière commande exécutée

  $$ → n° de processus de Shell

  $! → n° de processus de la dernière commande lancée par &

Protection des caractères joker

- Ce sont tous les caractères spéciaux du Shell : (*, ?, &, <, >, !, ...).
- Il existe plusieurs procédés pour demander au Shell de ne pas interpréter les caractères génériques :
  - apostrophes : '...' → protection intégrale
  - guillemets : "..." → le shell interprète les $, `...`, et \, à l'intérieur
  - on peut utiliser \ devant chaque caractère à protéger
  - les guillemets protégent les apostrophes et vice-versa
- Pour forcer l'exécution d'une commande, utiliser les quotes inversées `cde`

```
#######################################
#                                     #
# Script illustrant l'accès aux       #
#  commandes shell                    #
#  variables d'environnement          #
#  arguments de la igne de commandes  #
#  variables shell                    #
#                                     #
#######################################

echo Bonjour
echo $LOGNAME
pwd
echo Nous sommes le `date|cut -d' ' -f1,2,3`,dossier de login : $HOME
echo "Nombre d'arguments :" $#
echo "Liste d'arguments :" $*
echo "Nom du script (\$0) :" $0
echo 'Premier argument ($1) :' $1
accueil=Bienvenue
echo $accueil, $USERNAME
set un deux trois quatre
echo "Liste d'arguments :" $*
```

```
#########################################
#                                       #
# Script illustrant l'accès aux         #
# aux arguments quand ils sont plus     #
#  que 9 via un décalage (shift)        #
#                                       #
#########################################

echo "Nombre d 'arguments :  $#"
echo Liste des arguments : $*
echo Argument 0 : $0
echo Argument 1 : $1
shift
echo Apres un shift :
echo "Nombre d 'arguments :  $#"
echo Liste des arguments : $*
echo Argument 0 : $0
echo Argument 1 : $1
```

Exécution conditionnelle

- **Instruction if**

- **Exécute des commandes en fonction du "compte-rendu" d'une autre commande**

```
if cde1
then cde2
else cde3
fi
```

- **exit n : sort du shell script en retournant un compte rendu**

```
#########################################
#                                       #
# Script illustrant l'exécution         #
#  conditionnelle (if then else)        #
#                                       #
#########################################

# Vérifie si l'argument passé est un fichier
# si oui affiche son contenu
# si non  affiche un message d'erreur

if cat $1 &>/dev/null
then
echo Contenu du fichier \<$1\>
cat $1
else
echo "<$1> n'est pas un nom de fichier"
fi
```

La commande test

- **Elle est utilisée comme prédicat dans les structures de contrôle.**

- **Exemples**
  - **test –f nom** — **renvoie vrai si nom est un fichier**
  - **test –d nom** — **renvoie vrai si nom est un répertoire**
  - **test "$a"** — **renvoie vrai si la variable a n'est pas la chaîne vide**
  - **test –z "$a"** — **renvoie vrai si la variable a est la chaîne vide**
  - **test ! –f nom** — **renvoie vrai si nom n'est pas un fichier**
  - **test $a op $b** — **op : -eq , -ne , -gt , -lt , … comparaison numérique**

    : , = , != comparaison littérale

- **Autre syntaxe […] (shell fils) ou [[…]] (même shell)**
  - **[-f nom]**
  - **[[$# = = 3]]**

```
#######################################
#                                     #
# Script illustrant l'expression      #
#  des conditions                     #
#                                     #
#######################################

# Test imbriqués (elif)
if test $# -eq 0
then
echo -e "\t\t\tPas d'argument"
elif test -f $1
then
echo $1 est un fichier
else
echo Au moins un argument : $*
fi

# comparaison littérale
if test $1 = toto
then
echo "C'est toto le premier"
fi

# autre syntaxe
if [[ $# > 2 ]]
then
echo "Trop d'arguments"
fi
```

Conditions multiples

- **Instruction case**

  case évalue une expression

  ```
  case mot in
  exp) commandes ;;
  exp) commandes ;;
  ...
  esac
  ```

  - *** est le cas par défaut**
  - **exp est une expression régulière**

```
#########################################
#                                       #
# Script illustrant l'exécution         #
# d'une condition multiple (case)       #
#                                       #
#########################################

# Affiche la date sous trois formes selon l'argument
# date : affiche la date
# heure : affiche l'heure
# affiche la date et l'heure sinon

case $1 in
heure) echo Il est : ;date|cut -d' ' -f5;;
date) echo Nous sommes le : ;date|cut -d' ' -f1,2,3;;
*)date
esac
```

Instruction for

```
for i in liste de mots
    do
        commandes
    done
```

$i prend une à une les valeurs de "liste de mots".

la liste est facultative :

```
for i
    do
        commandes
    done
```

$i prend les valeurs des arguments de la ligne de commande

```
#########################################
#                                       #
# Script illustrant l'exécution         #
#  d'une boucle de répétition for       #
#                                       #
#########################################

# Affiche le contenu des fichiers passés en arguments
# en ajoutant des lignes de séparations

for i
do
echo "----------Début du Fichier : $i ------------"
cat $i
echo "---------- Fin du Fichier : $i ------------"
done
```

Instructions while et until

Tant que

```
while cde 1
do
  cde2
done
```

cde2 est réalisée tant que cde1 retourne vrai

Jusqu'à ce que

```
until cde1
do
  cde2
done
```

cde2 est réalisée jusqu'à ce que cde1 retourne vrai

```
#########################################
#                                       #
# Script illustrant l'exécution         #
# d'une boucle de répétition while      #
#                                       #
#########################################

# Affiche tous les arguments passés sur la ligne
   de commande

echo Liste des arguments de la ligne de commande
while test $1
do shift
echo $1
done
```

Le langage

- **Conditions simples : && et ||**

  cde1 && cde2 → cde2 est exécutée si cde1 retourne vrai (et logique)

  cde1 || cde2 → cde2 est exécutée si cde1 retourne faux (ou logique)

- **exit n → sort d'un shell script et retourne un compte rendu**
- **break → sort d'une boucle**
- **continue → retourne en début de boucle, à l'itération suivante.**

- **Evaluation des variables**

  ${VAR} → $VAR
  ${VAR-val} → valeur de VAR si définie ; val dans le cas contraire
  ${VAR=val} → idem + $VAR devient val si indéfinie
  ${VAR?mes} → valeur de VAR si définie ; impression de"mes" sinon ; le shell se termine

La commande expr

- expr permet l'évaluation d'une expression arithmétique
- Syntaxe
  - expr var1 op var2 #op : +, -, *, :, %

- Exemple

```
$ a=3
$ b=8
$ expr $a + $b
11
$ c=`expr $a + $b`     ← c=$a+$b incorrect
$ echo $c
11
$ c=`expr $c + 1`
$ echo $c
12
```

Utilisation de Fonctions

- **Script essai**

```
function mafonc ()
{
local var1=hello

echo "mafonc - Liste des arguments :
$*"
echo "mafonc - Var1 local : $var1"
echo "mafonc - Var2 : $var2"
}

echo "Liste des arguments : $*"
var1=bonjour
var2=$1

mafonc bonsoir

echo "Liste des arguments : $*"
echo "Var1 : $var1"
echo "Var2 : $var2"
```

- **Sortie**

```
$ ./essai chien

Liste des arguments : chien

mafonc - Liste des arguments : bonsoir
mafonc - Var1 local : hello
mafonc - Var2 : chien

Liste des arguments : chien
Var1 : bonjour
Var2 : chien
```

Gestion des signaux

- **trap : gestion des signaux**
  → le but de trap est de piéger certains signaux précisés par leurs numéros
  → masquage : TRAP " n
  le signal n est ignoré
  → interruption : TRAP "cde" n
  l'événement est ignoré
  l'action à exécuter est précisée par "cde"
  → rétablir : TRAP n
  le signal n est rétabli

- **Quelques signaux**

  2 arrêt (interrupt : Ctrl + C)

  9 destruction (kill) → non masquable