Système d'exploitation
Scripting Bash
(2)

- Variables d'environnement
- Variables positionnelles
- Variables locales

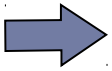
Rappel

- Edition du script *MonScript.sh* :

```
#!/bin/bash
date +%d/%m
```

- Exécution de *MonScript.sh* :

```
source MonScript.sh   #shell courant
bash MonScript.sh     #nouveau shell
```



14/09

- Rendre exécutable :

```
chmod a+rx MonScript.sh
./MonScript.sh
```

le droit r est indispensable en plus du x appliqué ici pour tous UGO

indispensable de faire précéder d'un chemin pour l'invocation (./ si script local)

- Placer un script dans /usr/bin ou /usr/local/bin le rend accessible depuis toute l'arborescence

Variables **d'environnement**

- Variables :
  - pour configurer l'environnement,
  - pour obtenir des informations sur l'environnement,
  - transmises aux commandes lancées depuis le shell,
  - par convention identifiants en majuscules.

- **printenv** ou **env** : LOGNAME, UID, GROUPS , HOME, HOSTNAME...

- Certaines variables sont renseignées dynamiquement : PWD, OLDPWD

- La variable **RANDOM** retourne un entier différent lors de chaque appel.
  (entier compris entre 0 et 32767)

Accès aux variables

- **Accéder** à la valeur d'une variable : faire précéder d'un **$**

```
echo $PATH     # affiche les chemins de recherche des exécutables

ls –lR /home |cut -d" " -f3- | grep "^$LOGNAME "
      # liste les fichiers sous /home de l'utilisateur connecté,
```

- **Affecter** une valeur à une variable : pas de $ *(pas d'espace autour du =)*

```
PATH=/usr/local/bin  # écrase la valeur contenue
```

- **Modifier**

```
PATH=$PATH:/usr/local/bin       # ajoute un chemin
```

Persistance des modifications

- Pour que les variables conservent leur valeur après déconnexion :

  les redéfinir dans un des fichiers d'environnement

- Exemple : redéfinition des invites **PS#**

```
user@jessie$PS1="$LOGNAME>"

user>PS2="?"
  ...
user>ls \
?
```

Attention : pas d'espace autour du =

nouvelle invite niveau 1

commande non terminée
le \ inhibe le retour à la ligne

nouvelle invite niveau 2

Invites (PS#)

- Séquences d'échappement *(man bash + rechercher le mot INVITE)*
  - **\d** : la date au format "Jour Mois Quantième" (ex : "Tue May 26")
  - **\H** : le nom d'hôte complet de la machine
  - **\t** : l'heure actuelle au format HH:MM:SS sur 24 heures
  - **\u** : le nom de l'utilisateur
  - **\w** : le répertoire de travail en cours
  - ...

Exemple :

- remplacement du $ par un triangle (code UTF-8 ici en décimal, 3 chiffres) :
  PS1=" \u : \H **\342\226\270**"

- ajout de la date au début de l'invite : PS1="**\d:**$PS1"

Lecture du flux d'entrée

1) **read** lit une ligne depuis l'entrée standard pour renseigner des variables

2) **read** *mot1 mot2 fin* : *mot1* : 1er mot , *mot2* : 2ème, et *fin* : le reste de la ligne

3) **read** sans paramètre : la ligne réponse est conservée dans la variable ***REPLY***

- echo "OK?" ; read ; echo "Vous avez répondu **$REPLY**"

- Options : *(man bash)*
  - -t délai : faux si aucune réponse n'est fournie avant le délai
  - -p message : affiche un message avant la saisie
  - -s : saisie invisible
  - -n ## : saisie d'un nombre de caractères fixé (-n 1 : touche pressée)

- Attention :
  - pour la lecture d'un fichier : **read ligne < fichier** # *ce qui doit être utilisé*
  - ~~cat fichier | read ligne~~ # *ne fonctionne pas*

    read est exécuté par un shell fils, la variable ligne est perdue

Variables **positionnelles**

- Le script et ses arguments sont affectés à des **variables positionnelles** :

  ./script.sh arg1arg2arg3 ...

  $0 $1 $2 $3 ... $9 ${10} ${11}

- Exemple : **concatenation.sh**

```
#!/bin/bash
cat $1 $2 >$3
echo "le script $0 a concaténé $1 et $2 dans $3"
```

./concatenation.sh fic1 fic2 fic3 : concatène fic1 et fic2 dans fic3
affiche : *le script concatenation.sh a concaténé fic1 et fic2 dans fic3*

./concatenation.sh * : concatène les premiers fichiers dans le dernier
(expansion des meta-caractères avant transmission au script)

./concatenation.sh : erreur d'exécution *(pas de paramètres)*

Arguments : décalage

- **shift** permet d'accéder aux arguments suivants en réalisant un décalage

  ./script.sh arg1arg2arg3

  $0 $1 ← $2 ← $3

  - **shift** => $2 est affecté à $1 , $3 à $2 ... *($0 ne change pas)*

    la valeur initiale de $1 est perdue *(si elle n'a pas été mémorisée)*

  - **shift** *x* *# effectue x décalages*

- Remarque :
  - **$0** : retourne le chemin qui a permis de lancer le script,
  - **basename $0** : retourne le nom du script (sans le chemin)

Exemple

- Le script qui affiche les droits sur les 2 premiers arguments:

```
echo -n "Droits de $1 : "
ls –l $1 |cut -c2-10
echo -n " Droits de $2 : "
ls –l $2 |cut -c2-10
```

./droits.sh titi toto tata

- Le même script mais en utilisant un décalage:

```
echo -n " Droits de $1 : "
ls –l $1 |cut -c2-10
shift
echo -n " Droits de $1 : "
ls –l $1 |cut -c2-10
```

(utile pour l'écriture de boucles)

Variables utilisateurs

- Variables et constantes **locales** à l'interpréteur

- Pas de déclaration de type préalable :
  - les variables sont de type chaînes de caractères,
  - les identifiants de variables sont en minuscules
  - les identifiants de constante en majuscules..

- Affectation **=** *(pas de $) :* message="Bonjour"

- Accès au contenu d'une variable : Faire précéder la variable de **$**
  echo "**$**message et bienvenue" *=> Bonjour et bienvenue*

  - **{ }** pour délimiter l'identifiant si nécessaire :
    echo "Texte du message **${**message**}**_1"

- Bonne pratique : définir des constantes en début de scripts
  - maintenance facilitée et lisibilité accrue

Portée des variables

- Une variable locale est associée au shell qui l'a créée
  - un script ne pourra modifier les variables du shell qui l'a lancé, que s'il a été lancé via la commande *source*
  - **export variable** (*# pas de $*) : exporte la variable (*accessible aux shells fils*)
    - *uniquement si réellement utile*
    - *pas d'export du fils vers le père, le père crée la variable, le fils la renseigne*

- **set** sans argument *(en ligne de commandes)*
  - liste les variables (shell et environnement) et leurs valeurs
- **set** avec arguments *(dans un script)*
  - affectation des arguments aux variables positionnelles $1, $2, ...
- **unset** *variable :* détruit la variable

./script.sh un deux trois

```
echo "Argument 1 : $1"          => Argument 1 : un
set one two three               ré-affectation
echo "Argument 1 : $1"          nouvelle valeur pou r $1 => Argument 1 :  one
```

Variables prédéfinies

- Variables prédéfinies

  $# => nombre d'arguments *(sans compter le script)*

  $* => tous les arguments : en une seule chaîne

  $@ => tous les arguments : en un tableau de valeurs chaînes

  $? => valeur de retour de la dernière commande exécutée

  $$ => n°de processus de Shell

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

  $_ => arguments de la dernière commande

Exemple

./script.sh un deux trois

```
echo "Argument 1 : $1"          => un
echo "Nombre d'arguments : $#"  => 3
echo "Liste des arguments : $*" => un deux trois
echo "Script : $0"              => script.sh
shift
echo "Après un shift :"
echo "Argument 1 : $1"          l'ex 2ème argument => deux
echo "Nombre d'arguments : $#" diminué de 1 => 2
echo "Liste des arguments : $*" 1er argument perdu => deux trois
echo "Script : $0"              le nom du script => script.sh
```

Protection des meta-caractères

- Forcer l'exécution d'une commande : `` `...` `` ou **$(...)**

```
echo "Nous sommes le `date` "
echo "Nous sommes le $(date)"
ladate=`date`
ladate=$(date)
```

- Inhiber l'interprétation des meta-caractères dans une chaîne :
  - **** inhibe l'interprétation du méta-caractère qui suit
    en fin de ligne permet d'écrire une commande sur plusieurs lignes
  - apostrophes : **'...'** => protection intégrale
  - guillemets : **"..."** => protection partielle : interprète les $, `` `...` ``,et \
  - les guillemets protègent les apostrophes et vice-versa

Exemple

évaluer la commande

pour pouvoir afficher $

```
echo "Bonjour, $LOGNAME"
echo "Nous sommes le `date|cut -d' ' -f1-3`"
echo "Nombre d'arguments passés au script: $#"
echo "Nom du script (\$0) : $0"
echo "Le premier argument ($1) est :' $1
accueil=$(date)
echo "$USER (le $accueil)"
```

pour pouvoir afficher $

création d'une variable shell accueil

évalue la commande

Conditions

- Toute commande peut être interprétée comme une condition d'exécution, elle retourne une valeur rendant compte de son exécution :
  - code d'erreur (ou compte-rendu) indiquant s'il y a eu erreur ou non

- Tout script doit retourner un compte-rendu via la commande ***exit*** *n*
  - stoppe l'interpréteur => sort du script en retournant un compte rendu
    - exit 0 : exécution correcte
    - exit *nnn* : entre 1 et 255, indique une erreur
    - sans exit explicite : retourne le compte-rendu de la dernière commande

- *Attention : si le script a été lancé par source (pas de shell fils),*
  - *exit ferme l'interpréteur (et éventuellement le terminal!).*

Exécution conditionnelle

- **if** *commande :* exécute des commandes en fonction du compte de *commande*

```
if commande
then        ...commandes exécutées en cas de succès de commande...
else        ...commandes exécutées en cas d'échec...
fi
```

elif équivalent à else if, mais un seul fi suffit

```
if cat $1 &>/dev/null
    then
        echo "Contenu du fichier <$1>"
        cat $1
    else
        echo "<$1> n'est pas un nom de fichier"
    exit 1
fi
exit 0
```

cat propre : Vérifie si l'argument est un fichier si oui affiche son contenu ,sinon affiche un message d'erreur

Attention

- **if then else elif fi** sont des commandes
  - en début de ligne ou après un ";"

```
if grep -q "$1" $2
then
        echo "$1 est dans $2"
    else
        echo "$1 n'est pas dans $2"
fi
exit 0
```

*option **-q** (quiet) :*
pas d'affichage
uniquement un compte-rendu
0 : trouvé,
1 : non trouvé,
2 : erreur

écriture équivalente :

```
if grep -q "$1" $2; then echo "$1 est dans $2"
else  echo "$1 n'est pas dans $2" ; fi
exit 0
```

Commande test

- La commande **test** permet d'exprimer des conditions

```
if test $1   # teste l'existence de la variable $1
then
   echo "Paramètres :$*"
else
   echo "Pas de paramètre"
fi
```

- Conditions sur les fichiers *(man bash) :*
  - test **-f** *nom :* vrai si nom est un fichier
  - test **-r** *nom :* vrai si nom est un fichier/répertoire lisible droit r
  - test **-O** *nom :* vrai si nom est un fichier/répertoire appartenant à nom
  - test *fichier1* **-ot** *fichier* : vrai si *fichier1* est plus vieux *fichier2*
  - ...

    Une seule option : ~~test -f -r fic~~

- Négation : **!** *(encadré par des espaces)*
  - **!** test -f *nom :* vrai si *nom* n'est pas un fichier

Commande test : **variables**

- Conditions sur des variables : test $a opérateur $b
  - les éléments suivant la commande *test* sont des options et arguments

- **Valeurs entières,** opérateurs : **-eq , -ne , -gt , -ge , -lt , -le**

  if test *$entier* –gt *0*

- **Chaînes de caractères** opérateurs : **= , != , <, >**
  - espaces autour du symbole **=** , comparaison selon l'ordre ASCII

    test $chaine = "Bonjour"
  - **>** est aussi le symbole de redirection il faut le protéger !

    if test $chaine **\>** "texte"

- Options : **-z** : chaîne vide, **-n** : chaîne non vide
  - test –z *$chaine*

Tests imbriqués

```
# comparaison numérique
if test $# -eq 0
    then
        echo -e "\t\t\tPas d'argument"
    elif test -f $1
        then
            echo "$1 est un fichier"
        else
            echo "Le premier argument doit être un fichier"
            exit 1
fi

# comparaison littérale
if test $1 = "toto"
    then
        echo "C'est toto le premier"
fi
exit 0
```

Combinaisons d'exécutions

- Exécutions conditionnées avec && et ||
  - cde1 **&&** cde2 : *cde2* est exécutée si *cde1* retourne vrai *(et logique)*
  - cde1 **||** cde2 : *cde2* est exécutée si *cde1* retourne faux *(ou logique)*

  - Exemple :

    test $UID –gt 999 && echo "usager standard"

    affiche usager standard si $UID>999

- Négation : **!** ne porte que sur la commande qui suit directement => **( )** :

  1) ! test -r fic && test -f fic && echo "fichier non lisible"

  2) ! (test -r fic && test -f fic) && echo "n'est pas un fichier non lisible"

  (est soit lisible , soit pas un fichier, voire les deux)

Conditions sans if

```
if test -f fic ; then
if test -r fic ; then
cat fic
fi
<=>
test -f fic && test -r fic && cat fic


if grep -q "$1" $2
then
    echo "$1 est dans $2"
 else
    echo "$1 n'est pas dans $2"
fi
<=>
grep -q "$1" $2 && echo "$1 est dans $2" || echo "$1 n'est pas dans $2"
```

Conditions multiples

- **case** évalue une expression comme condition:

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

- exp) : expression meta-caractères Unix *(pas une expression régulière)*
- *) : cas par défaut, mot ne correspond à aucune des expressions

- NPO : **;;** termine les commandes, *s'il est omis, continue avec le traitement pour le cas suivant*

Exemple

```
# affichage fonction du paramètre ("./quelle.sh heure" ou "./quelle.sh date")
case $1 in
    heure) echo -n "Heure :" ; date|cut -d' ' -f5;;
    date) echo -n "Date :"; date|cut -d' ' -f1-3;;
    *)date
esac

# meta-caractères  Linux
read -p "Debian c'est le top ?" reponse
case $reponse in
    O*|Y*) echo "Bonne réponse";;
    N*) echo "Vous serez bientôt convaincu";;
    *)echo "Il faut se décider" ;;
esac
```