Système d'exploitation
Scripts Bash
(4)

- Les boucles
  - for *(pour)*
  - while *(tant que)*
  - until *(jusqu'à ce que)*
  - select *(choisir)*

Instruction **for**

- Répéter une suite d'instructions pour tous les éléments d'une **liste** :

pas de $, "affectation"

```
for nom in une suite de chaînes de caractères
do
    ...commandes...
done
```

$nom prend pour valeur les chaînes une à une

Exemple :
pour chaque chaîne afficher s'il s'agit d'un nom d'usager :
soit "usager:UID", soit "usager:Inconnu"

ou "u1 u2 u3" ou avec tab ou retour à la ligne

```
for usager in "u1" "u2" "u3"
    do
        echo "$usager:$(id -u $usager 2>/dev/null||echo 'Inconnu')"
    done
```

```
u1:1000
u2:Inconnu
u3:Inconnu
```

Instruction for

- **Globbing** *(méta-caractères)* utilisable *(substitution a priori)*

```
echo "Répertoires commençant par un . :"
for   rep   in   ~/.*
do
  test -d $rep && echo -e "\t$(basename $rep)"
done
```

pour ne lister que le nom

Répertoires commençant par un . :"
.
..
.cache
.config
.dbus
.gconf
.local

- La liste peut résulter de l'exécution d'une **commande** :

```
echo "Répertoires silencieux :"
for   rep   in   $(ls -d ~/.*)
do
        test -d $rep && echo -e "\t$(basename $rep)"
done
```

Répertoires silencieux :"
.cache
.config
.dbus
.gconf
.local

Instruction for

- Sans liste : la variable prend les valeurs des variables positionnelles

```
for nom
do
    commandes
done
```

Ex : ./script.sh **t u v** => $nom prend successivement les valeurs **t u v**

- Ex : Créer un fichier *nb_lignes.txt* contenant les nombres de lignes des fichiers en argument

```
>nb_lignes.txt      #vide le fichier
for fichier
do
      echo -n "$fichier : " >>nb_lignes.txt
      wc -l $fichier | cut -d' ' -f1 >>nb_lignes.txt
done
```

titi.txt :10
toto.txt :0
script.sh :100

Instruction for

- Utilisation avec affectation des variables positionnelles :

```
# Afficher ligne à ligne les éléments de la date
set $(date | cut -d" " -f1-3)          # Lundi 14 Octobre
                                        $1    $2    $3
for i
do                          Lundi
                            14
   echo $i                  Octobre
done
```

  - **do** est une instruction => en début de ligne ou après un "; "
  - **do** doit suivre directement le for (pas d'instruction intercalée)

  - Autre écriture :

```
set $(date|cut -d, -f1)
for i ; do echo $i ; done
```

  - ou encore :

```
for i in $(date|cut -d, -f1) ; do echo $i ; done
```

Instruction **seq**

- Boucle numérique

```
for i in 1 2 3
do
      echo $i
done
echo Go!
```

```
1
2
3
Go!
```

- Avantageusement réécrit avec l'instruction **seq**

```
for i in $(seq 3) ;do echo $i; done
echo Go!
```

  - *possibilité de fixer le début (à 1 p. défaut), le pas et la fin*

```
seq 5 10      => 5 6 7 8 9 10
seq 10 5 25   => 10 15 20 25
seq 10 5 23   => 10 15 20
```

Instruction **while**

- Répéter une suite d'instructions tant qu'une condition est vérifiée

```
while commande  # tant que la commande condition s'exécute avec succès
    do
        ...commandes...
    done
```

- Exemple : *Afficher les arguments de la ligne de commande (1 par ligne)*

```
echo "Arguments :"
while test $1    # si l'argument existe
do
    echo -e "\t$1"
    shift          # décalage
done
```

```
Arguments :
   un.txt
   deux.txt
```

Instruction while

- **while** attend une commande comme condition => **$(...)** inutile

- Ex : *Pour surveiller la déconnexion de l'utilisateur passé en paramètre*

```
while who| grep $1 &>/dev/null  (l'affichage n'est pas utile)
do
   echo "$(date +%H:%M:%S) : $1 connecté..."
   sleep 5 # pause de 5s
done
echo "\n$1 déconnecté!!!"
```

```
10:00:15 : user connecté…
10:00:20 : user connecté…
10:00:25 : user connecté…

user déconnecté!!!
```

- Si l'instruction condition est toujours réalisable *(exit 0)* => boucle infinie
- Exemple : read *(lecture clavier => toujours possible)*

```
while read saisie # boucle infinie
do
   ...   # une des instructions doit permettre la sortie de boucle
done
```

Sortie de boucle

- **break** : sort d'une boucle, *pas de valeur de retour, un seul niveau*

  (read -n1 c : lit un seul caractère, attention ne pas saisir "Enter")

- Exemple : *trouver la lettre*

```
reponse=k                # lettre à deviner
echo "Trouves la lettre : "
while true               # true:mot réservé, boucle infinie
do
   read -n1 lettre       # lire un seul caractère
   if test $lettre = $reponse
   then break            # sortie de boucle si trouvée
   fi
   echo -n "\nEssaies encore!"
done
echo -n "\nBravo!"
```

```
Trouves la lettre :
a
Essaies encore!
b
Essaies encore!
k
Bravo!
```

- Rmq : exit # : sort de l'interpréteur courant et donc de la boucle

Itération suivante

- **continue** : passe directement à l'itération suivante
  (sans exécuter les instructions qui suivent)

- Ex : lister les droits des répertoires de connexion des **autres** utilisateurs

```
rwxr-xr-x /home/ens1
rwxr-xr-x /home/etu1
rwxr-xr-x /home/exemple
```

```
for rep in /home/*
do
   if test $rep = $HOME   # exclure l'utilisateur courant
         then continue
  fi
   ls -ld $rep |cut -c2- |tr -s " "|cut -d" " -f1,9
done
```

Autres formes de boucles infinies

- Opération vide (nop) :

```
while :
do
   echo .
done
```

- Attention à la place du do :

```
while
echo x # Instruction toujours vraie => boucle infinie
do
   echo .
done
```

Redirection d'entrée : lecture fichier

- La condition peut être issue de la lecture d'un fichier
  - la redirection est spécifiée au niveau du done
    - boucle tant que la lecture est possible *(i.e. non fin de fichier)*

Ex : Afficher le contenu d'un fichier

```
while read
do
   echo $REPLY
done < data.txt
```

spécification de la redirection d'entrée

Lecture/Ecriture fichier

- Ex : Modifier le fichier en paramètre en encadrant les lignes par des balises ** ... **

  (=> passer par un fichier temporaire)

```
>tampon.tmp              # créer/vider le fichier tampon

while read ligne         # lecture depuis le canal d'entrée
do

   echo "<p>$ligne</p>" >>tampon.tmp

done  <$1                # redirection d'entrée

mv tampon.tmp $1         # modification du fichier
```

```
un titre
un texte
une phrase
```

```
<p>un titre</p>
<p>un texte</p>
<p>une phrase</p>
```

Redirection d'entrée

- lire le résultat d'une autre commande : tube au niveau du while

```
commande |while read
do
   …
done
```

```
exemple  tty1         2015-09-24 10:24
user     :0           2015-09-24 08:50
(:0)
user     pts/0        2015-09-24 09:34
(:0.0)
user     pts/1        2015-09-24 10:24
(:0.0)
```

Ex : afficher les noms des usagers connectés avec leur uid

```
who |cut -d' ' -f1 |sort |uniq |while read nom
do
   echo $nom a pour id : $(id -u $nom)
done
```

```
exemple a pour id : 1007
user a pour id : 1000
```

Instruction **until**

- **until** : Les *commandes* sont exécutées jusqu'à ce que la *commande condition* s'exécute avec succès
  - condition est spécifiée en début de boucle

```
until commande condition
do
        ...commandes...
done
```

Ex : lire et afficher les lignes du fichier paramètre jusqu'au mot "fin"

```
ligne=""
until test "$ligne" = "fin"
do
   read ligne
   echo $ligne
done <$1
```

Protection des chaînes

- Le script précédent produit une erreur si le fichier contient une ligne vide
  => la condition devient dans ce cas **until** test = "fin"
  erreur opérande attendu 

- Solution : encadrer la chaîne lue par un caractère arbitraire

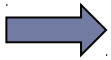
  Ex : encadrement des chaînes par un x (choix arbitraire)

```
ligne="##"        # initialisation pour entrer dans la boucle
until test "#${ligne}#" = "#fin#"
do
   read ligne
   echo $ligne
done <$1
```

Proposer un menu : **select**

- **select** permet de proposer à l'utilisateur un choix parmi des chaînes

```
select reponse in "pause" "tp" "interro"
do
    echo "$REPLY($reponse)"
done
```



```
1) pause
2) tp
3) interro
#? 1
1(pause)
#? 2
2(tp)
#?
```

L'invite en attente de réponse est $PS3 (#? par défaut)

- Le choix, un **chiffre**, est enregistré dans la variable **$REPLY**
  - si le chiffre fait partie des choix proposés, $reponse contient la **chaîne** correspondante, sinon $reponse est vide
- select est une **boucle infinie** => pour sortir il faut exécuter un break (ou exit)

Exemple

```
PS3="Entrez votre choix : "          # configuration du prompt

select systeme in "Linux" "Windows"    # choix possibles
do
    if test $REPLY -eq 1 || test $REPLY -eq 2
    then
        echo "Vous avez choisi $REPLY ($systeme)"
        break                          # sortie du select
    else
        echo "Mauvais choix, veuillez recommencer"
    fi
done
```

```
1) Linux
2) Windows
Entrez votre choix : 3
Mauvais choix, veuillez recommencer
Entrez votre choix : 1
Vous avez choisi 1 (Linux)
```