Shell Scripts

Philippe Roose

C’ est quoi ?

- Interpréteur de commandes
- Deux fonctionnalités
  - Mode Interactif
    - Attend que l’on tape une ligne de commandes ;
  - Mode Scripts
    - Exécution de programmes écrits en shell.

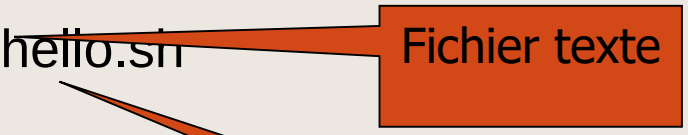
Ca contient quoi ?

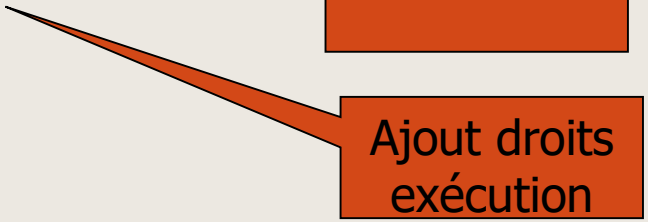
- Shell = fichier texte avec des instructions + éventuellement l'appel à des exécutables.

```
% cat > hello.sh <<MY_PROGRAM
#!/bin/sh
echo 'Hello, world'

MY_PROGRAM
% chmod +x hello.sh
% ./hello.sh

Hello, world
```





Fonctionnement

- Une fois lancé, un shell est inactif
  - attend qu'on lui donne des ordres
  - qand on lui en donne un, il l'exécute
  - quand il a terminé, il retourne à son état d'inactivité
- Quand le shell est inactif, il affiche un *invite* (*prompt* en anglais)

Fonctionnement

- Lorsque l'on tape une commande, le shel lit le contenu de la variable PATH qui indique les « chemins » où trouver les commandes.
  - PATH=/usr/bin;/bin;/home/toto/bin,
- Si l'on tape bonjour, le shell va chercher successivement les commandes :
  - /usr/bin/bonjour,
  - /bin/bonjour et
  - /home/toto/bin/bonjour ;
- S'il ne trouve la commande dans aucun des répertoires référencés par le PATH, il va renvoyer un message d'erreur
  - >~ $ bonjour
  - bonjour: Command not found

Programmation en Shell Script

- Un fichier contenant des commandes pour le shell est appelé un *script*.
- C'est un programme écrit dans le langage du shell.
- Un script shell est un fichier en mode texte, non compilé, directement exécutable : il est interprété.
- Il existe plusieurs shell
  - sh : le standard, à utiliser pour chaque script
  - Autres shells : bash (Bourne-Again Shell), ksh (Korn shell)

Exemple de script

```
#!/bin/sh
echo "Êtes-vous favorable au remplacement du NIR par le VIR ?"
select opinion in Pour Contre
do case $opinion in
# Laisser passer ceux qui répondent correctement à la question
"Pour"|"Contre") break;;
# Au cas où des zozos tapent sur autre chose que 1 ou 2
"*") continue;;
esac
done
# M'envoyer le résultat par mail echo "$opinion" | mail bourdieu
```

Illisible ????

Exemple de script

```
#!/bin/sh
echo "Êtes-vous favorable au remplacement du NIR par le VIR ?"

select opinion in Pour Contre
    do
            case $opinion in
            # Laisser passer ceux qui répondent correctement à la question
            "Pour"|"Contre") break;;
            # Au cas où des zozos tapent sur autre chose que 1 ou 2
            "*") continue;;
            esac
    done
# M'envoyer le résultat par mail
echo "$opinion" | mail bourdieu
```

- C'est quand même mieux quand c'est indenté…

Commandes

- # : Commentaires
- echo : ecriture sur sortie standard

Premier script, l'éternel…

```
#!/bin/sh
# Fichier "helloworld"
echo "Hello world"
```

- Pour exécuter un script, il doit avoir les droits en exécution…(+x)

echo...encore

```
#!/bin/sh #
echo "Bonjour... "
echo "Comment allez-vous ?"
```

- Affiche les lignes suivantes :
  - Bonjour…
  - Comment allez-vous ?

  et non :
  - Bonjour… Comment allez-vous ?

- Annuler le retour chariot : option -n.

```
#!/bin/sh
echo -n "Bonjour..."
echo "Comment allez-vous ?"
```

  - Bonjour… Comment allez-vous ?

echo...toujours

```
% echo ‘$USER’
$USER
% echo “$USER”
roose
% echo “\””
”
% echo “deacnet\\sct”
deacnet\sct
% echo ‘\”’
\”
```

Lecture entrée standard

- Read

```
#!/bin/sh
echo "Bonjour... Comment vous-appelez-vous ?"
read nom prenom
echo "Je vous souhaite, $prenom $nom, de passer une
  bonne journée.«
read # lecture ‘à blanc’
```

- L’espace est le séparateur
- La commande read doit être suivie du seul nom de la variable, **non précédé du signe dollar**.
- Le signe dollar ne doit précéder le nom de la variable que lorsque l'on cite son contenu.

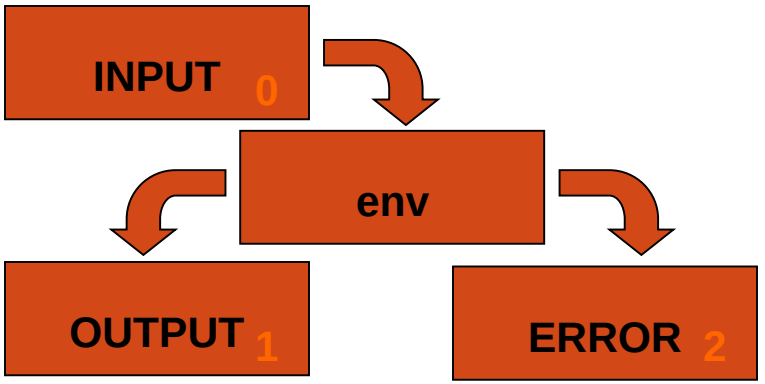
Redirections…rappel

- > Redirection de sortie dans un fichier
  - Avec écrasement
- >> Redirection de sortie dans un fichier
  - Avec ajout à la suite
- < Redirection de l'entrée

```
./saisie < fic_entrée >> listing
```

Redirections...rappel

- `cat > /tmp/myfile`
- `cat >> /tmp/myfile`
- `cat 2> /tmp/myerr`
- `cat < /tmp/myinput`
- `cat <<INPUT`
  `Some input`
  `INPUT`



Redirections

- ls foo || echo "Je n'ai pas de fichier foo."
  - Exécute ls, si erreur, exécute le echo
- grep gag /usr/dict/words | tail
  - Affiche les 10 derniers mots du dictionnaire qui contiennent la chaîne gag.

La panoplie complète des redirections est la suivante :

- > : change la sortie standard de la commande pour la placer dans un fichier.
- < : change l'entrée standard de la commande pour la prendre dans un fichier.
- | : branche la sortie standard de la commande de gauche sur l'entrée standard de la commande de droite.
- >> : change la sortie standard pour l'ajouter à la fin d'un fichier existant.
- || : exécuter la commande suivante si la première a échoué.
- **&&** : n'exécuter la commande suivante que si la première a réussi.

Gestion des E/S

- **more** (ou **less**, bien plus évolué)
  - affiche le résultat page par page, pour laisser le temps de le lire.
- **wc**
  - compte le nombre de caractères, de mots et de lignes de son entrée.
- **grep**
  - cherche dans son entre les lignes contenant un mot donné, et les écrit sur sa sortie.
- **sort**
  - lit toutes les lignes de son entrée, les trie, et les écrit dans l'ordre sur sa sortie. Par défaut l'ordre est alphabétique.
- **tail**
  - écrit sur sa sortie les dernières lignes de son entrée.
- **head**
  - écrit sur sa sortie les premières lignes de son entrée.
- **cat**
  - copie plusieurs fichiers sur sa sortie.
- **fold**
  - coupe les lignes de son entrée à 80 caractères et écrit le résultat sur sa sortie.

- Ces commandes sont fréquemment associées à un *pipe*

Les tests

- Le shell propose deux principales façons de réaliser un test ; ces deux méthodes sont équivalentes :
  - test *expression*
  - [*expression*]
- Un test renvoie un *code de retour* **(0 ou autre)**
  - 0 correspond à la réponse « vrai ».
  - Pour répondre « faux », le programme répond… autre chose (ce peut être 1, 2, -1 ou autre).

```
test -f foo || echo "Le fichier foo n'existe pas."
```

Principaux tests

- *test -f monFichier*
  - vrai si monFichier existe
- *test -r monFichier*
  - vrai si monFichier est accessible en lecture
- *test -w monFichier*
  - vrai si monFichier est accessible en écriture
- *test -d monFichier*
  - vrai si monFichier est un répertoire

La totale

Tests sur les fichiers (et sur les répertoires)

- ***-e fichier*** : Vrai si le fichier/répertoire existe.
- ***-s fichier*** : Vrai si le fichier à une taille supérieure à 0.
- ***-r fichier*** : Vrai si le fichier/répertoire est lisible.
- ***-w fichier*** : Vrai si le fichier/répertoire est modifiable.
- ***-x fichier*** : Vrai si le fichier est exécutable ou si le répertoire est accessible.
- ***-O fichier*** : Vrai si le fichier/répertoire appartient à l'utilisateur.
- ***-G fichier*** : Vrai si le fichier/répertoire appartient au groupe de l'utilisateur.
- ***-b nom*** : Vrai si nom représente un périphérique (pseudo-fichier) de type bloc (disques et partitions de disques généralement).
- ***-c nom*** : Vrai si nom représente un périphérique (pseudo-fichier) de type caractère (terminaux, modems et port parallèles par exemple).
- ***-d nom*** : Vrai si nom représente un répertoire.
- ***-f nom*** : Vrai si nom représente un fichier.
- ***-L nom*** : Vrai si nom représente un lien symbolique.
- ***-p nom*** : Vrai si nom représente un tube nommé.
- ***fichier1 -nt fichier2*** : Vrai si les deux fichiers existent et si fichier1 est plus récent que fichier2.
- ***fichier1 -ot fichier2 :*** Vrai si les deux fichiers existent et si fichier1 est plus ancien que fichier2.
- ***fichier1 -ef fichier2*** : Vrai si les deux fichiers représentent un seul et même fichier.

La totale : Tests sur les entiers

- ***entier1 -eq entier2*** : Vrai si entier1 est égal à entier2.
- ***entier1 -ge entier2*** : Vrai si entier1 est supérieur ou égal à entier2.
- ***entier1 -gt entier2*** : Vrai si entier1 est strictement supérieur à entier2.
- ***entier1 -le entier2*** : Vrai si entier1 est inférieur ou égal à entier2.
- ***entier1 -lt entier2*** : Vrai si entier1 est strictement inférieur à entier2.
- ***entier1 -ne entier2*** : Vrai si entier1 est différent de entier2.

La totale : Tests sur les chaînes

- ***-n "chaîne"***
  - Vrai si la chaîne n'est pas vide.
- ***-z "chaîne"***
  - Vrai si la chaîne est vide.
- ***"chaine1" = "chaine2"***
  - Vrai si les deux chaînes sont identiques.
- ***"chaine1" != "chaine2"***
  - Vrai si les deux chaînes sont différentes.

La totale combinaison de tests

- NON expression
  - ***test ! expr***
  - ***[! expr]***
- expr1 OU expr2
  - ***test expr1 -o expr2***
  - ***[expr1 -o expr2]***
- expr1 ET expr2
  - ***test expr1 -a expr2***
  - ***[expr1 -a expr2]***

Exemples

touch foo

rm bar

[-f foo] = vrai si le fichier foo existe

[! -f bar] = vrai si bar n'existe pas

à n'exécuter que si foo existe ET que bar n'existe pas.

[-f foo **-a** ! -f bar] && mv foo bar

Valeurs de retours

Exemple 1

```
ls /does/not/exist
% echo $?
1
% echo $?
0
```

Exemple 2

```
% cat > test.sh <<_TEST_
exit 3
_TEST_
% chmod +x test.sh
% ./test.sh
% echo $?
3
```

Math...

- `$(( ))`

```
% echo $(( 1 + 2 ))
3
% echo $(( 2 * 3 ))
6
% echo $(( 1 / 3 ))
0
```

Conditions
if

Syntaxe générale

if *condition*
then *commandes*
elif *condition*
then *commandes*
else *commandes*
fi

```
#!/bin/sh
if test $PWD = $HOME
    then echo "Vous êtes dans votre répertoire d'accueil."
elif test $PWD = $HOME/bin
    then echo "Vous êtes dans votre répertoire d'exécutables."
elif test $PWD = $HOME/bin/solaris then echo "Vous êtes dans votre répertoire d'exécutables pour Solaris."
else echo 'Vous êtes dans un répertoire quelconque, qui n'est ni $HOME,'
    echo 'ni $HOME/bin, ni $HOME/bin/solaris. '"Vous êtes dans $PWD."
fi

# Test d'existence de fichier
if [ -e /etc/passwd ]
then
  echo "/etc/passwd existe"
else
  echo "/etc/passwd pas trouvé!"
fi
```

Conditions case

- La boucle if est très pratique, mais devient rapidement illisible lorsqu'un aiguillage offre plusieurs sorties, et que l'on doit répéter une condition plusieurs fois sous des formes à peine différentes.

- Syntaxe générale

  case *chaîne* in

  motif) *commandes* ;;

  motif) *commandes* ;;

  esac

- Un *motif* (*pattern*) est un mot contenant éventuellement les constructions *, ?, [a-d], avec la même signification que pour les raccourcis dans les noms de fichiers (les *jokers*).
- Exemple :

```
case $var in
   [0-9]*) echo "$var est un nombre.";;
   [a-zA-Z]*) echo "$var est un mot.";;
   *) echo "$var n'est ni un nombre ni un mot.";;
esac
```

```
case "$PWD"
   "$HOME " | "$HOME/bin") echo "Vous êtes dans votre répertoire d'accueil ou d'exé.";;
   "$HOME/bin/solaris") echo "Vous êtes dans votre répertoire d'exécutables pour Solaris.";;
   "$HOME/prive") echo "Vous êtes dans votre répertoire privé.";;
   "*") echo 'Vous êtes dans un répertoire quelconque, qui n'est ni $HOME,'
         echo 'ni $HOME/bin, ni $HOME/bin/solaris. '"Vous êtes dans $PWD.";;
esac
```

Boucles while

- Syntaxe générale

while *condition*
    do *commandes*
done

```
numero=1
while test $numero != 51
   do
   touch fichier"$numero"  # crée un fichier# vide
   numero=$(($numero + 1))
done

# incrément d'un compteur tant que le fichier.compteur existe
COUNTER=0
while [ -e "$FILE.COUNTER" ]
do
  COUNTER=$(( COUNTER + 1))
done
```

Boucles until

- Syntaxe générale

until *condition*
  do *commandes*
done

```
echo "Tapez le mot de passe : "
read password
until test $password = mot de passe correct
 do
  echo "Mot de passe incorrect."
  echo "Tapez le mot de passe"
  read password
done
echo "Mot de passe correct."
echo "Bienvenue sur les ordinateurs secrets de la
  NASA."
```

Boucles for

- Syntaxe générale

for *var* in *liste de chaînes*
    do *commandes*
done

```
#!/bin/sh
for i in 1 2 3
do
  echo $i
done

for element in *
  do echo "$element"
done
```

L'* signifie tous les fichiers du répertoire courant

C'est la commande ls !

For...version « C »

```
LIMIT=10
for (( a=1          ;
       a<=LIMIT     ;
       a++          ))
do
  echo -n "$a "
done
```

Boucles select

- Syntaxe générale

select *variable* in *valeurs possibles*
    do *commandes*
done

```
#!/bin/sh
echo "Êtes-vous favorable au remplacement du NIR par le VIR ?"
select opinion in Pour Contre
    do
      case
            $opinion in "Pour"|"Contre") break;;  # Si choix 1 ou 2 => On sort
            "*") continue;; # Si autre choix, on reboucle 'continue' ne fait…rien.
      esac
done
```

- Êtes-vous favorable au remplacement du NIR par le VIR ?
- 1) Pour
- 2) Contre

Fonctions

- Syntaxe générale

```
nom () {
        instruction1
        instruction2
        …
}
```

```
#!/bin/sh
effacer_fichier ()  {
        echo "$1"
        echo "Voulez-vous vraiment l'effacer ? (o/n)"
        read reponse
        if [ $reponse = "o" ]
                then rm -f $1
        fi
}
for fichier in *.bak
do
        effacer_fichier $fichier
done
for fichier in *.old
do
        effacer_fichier $fichier
done
```

$1 = param1
$2 = param2
…

Tous les fichiers du répertoire courant *.old*

Remarque : Le passage de paramètre est identique à un script. $1, $2, etc. Le nom $0 représente le nom du script.

Variables d'environnement

- Pour **définir** le contenu d'une variable
  - on écrit simplement son nom, **sans le signe $**
- Pour **utiliser** le contenu d'une variable
  - on fait précéder son nom du **signe $**.

- **PWD** : Répertoire courant
- **USER** : nom de l'utilisateur courant
- **PATH** : chemins où aller chercher les commandes
- **DISPLAY** : L'écran sur lequel les programmes X travaillent. Cette variable est souvent de la forme : machine.somehost.somewhere:0.0 Si cette variable est vide, c'est qu'il n'y a pas d'affichage graphique possible.
- **SHELL** : contient le nom de votre shell.
- **HOME** : contient le nom de votre répertoire personnel.
- **USER** : contient votre nom de login.
- **LOGNAME** : la même chose que USER.

Variables pré-définies

- ***$0***
  - Le nom de la commande (i.e. : du script)
- ***$1, $2, etc.***
  - **$1** à **$9** : contient les neufs premiers paramètres; s'il y en a plus de neuf, il faut utiliser la commande *shift* qui décale sur la droite les arguments : $1 reçoit $2, …, $9 reçoit le dixième argument;
- ***$****
  - La liste de tous les arguments passés au script.
- ***$#***
  - Le nombre d'arguments passés au script.
- ***$?***
  - Le code de retour de la dernière commande lancée.
- ***$!***
  - Le numéro de processus de la dernière commande lancée en tâche de fond.
- ***$$***
  - Le numéro de process du shell lui-même.

FIN