

# Shell Scripts

## Licence 2

2014-2015

*Sylvain Cherrier*

*cherrier@univ-mlv.fr*

# Le shell est aussi...

- Un langage de programmation !!
- Spécialisé pour les tâches d'administration
- Offre les variables
- Les structures de controles
- Les boucles
- Les tableaux...

# Caractéristiques de la prog Shell

- Langage interprété
- Des variables
  - Non déclarées, non typées
  - \$ pour l'accès au contenu d'une variable  
(*read **nom*** mais *echo **\$nom***)
- Des structures de contrôle
  - For, if, switch, while, until
  - Très pauvre par rapport au C
- TEST (ou [ ] )
  - Très spécifique au shell

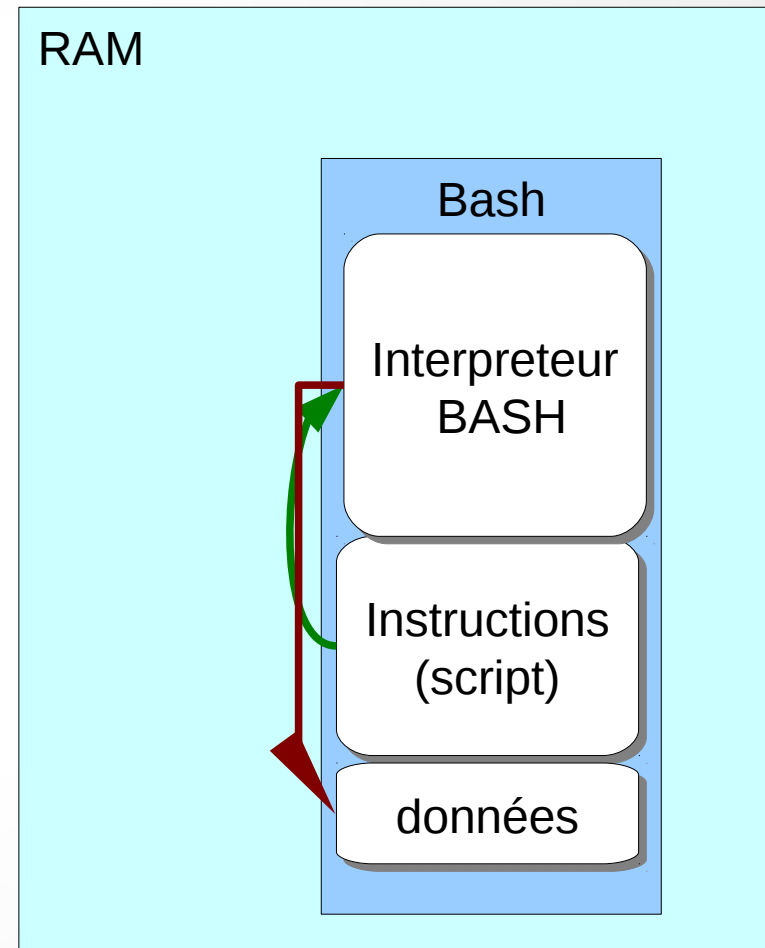
# Interpréteur

- *Interprété* s'oppose à *compilé*
- Un programme Shell (*interprété*) est une liste d'instructions, comme le source d'un programme compilé
- Par contre, la compilation analyse le source, et produit l'objet une fois pour toute. Le programme est l'objet résultant, un binaire
- L'interprétation travaille directement sur le fichier source, et analyse à chaque fois chaque instruction

# Interpréteur

Exécution d'un script en mémoire

- L'interpréteur ligne le script ligne par ligne
- Chaque ligne est interprétée, et la fonction demandée est exécutée.



# Typologie des langages de programmation

- Objectifs du langage
- Cadre d'utilisation
- Compilé/Interprété
- Variable Déclarée(O/N), typée(O/N)
- Gestion de la mémoire (O/N)
- Portabilité, rapidité, taille du code, etc...
- procédural, objet...

# Classer les langages de prog

- .....  
.....
- .....  
.....
- .....  
.....
- .....  
.....
- .....  
.....

# Script shell

- Fichier qui contient une suite de commandes...
- Droits d'exécution
- Chemin d'accès
- Source *script* (ou *. script*)
- sh script
- HashBanger ( `# !/bin/bash`) sur ligne 1



# Exemple

```
# #!/bin/bash
```

```
#####
```

```
# mon premier script
```

```
# version 1.0
```

```
#####
```

```
echo -n « Entrez votre nom : »
```

```
read rep
```

```
echo « Vous êtes $rep »
```

```
exit 0
```

# Script shell

- Pas de numéro de ligne
- Pas d'extension particulière (.sh possible)
- Pas de terminaison de ligne particulière
- ; est le séparateur de commandes sur une ligne
- # est le symbole de commentaire
- Exit 0 à la fin par (bonne) habitude

# Les variables de shell

```
var="une chaine"
```

```
var=$PATH
```

```
var='$var'
```

```
var=${var}baba
```

```
var="/tmp/$var"
```

```
l=5
```

```
noms[0]=Boule ; noms[1]=Bill
```

```
${nom[1]}
```

```
ou bien noms=( Boule Bill )
```

*pas d'espace autour de =*

Les tableaux  
doivent être  
protégés par des  
{ }

\$var permet de  
récupérer le  
contenu de var

# Les variables particulières

- Quelques variables sont générées par le fonctionnement du shell
- \$\$ contient le *PID* du shell
- \$ ? contient la valeur de retour de la dernière commande exécutée
- Pour les scripts, on peut récupérer les args de la ligne de commande (comme *argc*, et *argv*) (\$0 \$1 \$2 etc...)
- \$# contient le nb d'args de la ligne de commande
- \$\* contient uniquement les arguments (pas \$0)

# Variables

- Deux types : système ou utilisateur
- Système : générées par le système lui même
  - SHELL USERNAME LOGNAME HOME OSTYPE PWD
  - PS1 PATH
- Utilisateur
  - Nom sans espace, alphanumérique et \_
  - Pas de ? \* ou , ;

# ATTENTION

- C'est du **non déclaré non typé** !!
- Faire des opérations sur des variables inconnues entraîne automatiquement la création de la variable ! (contenant alors «») )
- Source de nombreuses erreurs !!
- Remarquez que la syntaxe est utilisée dans PHP

# Exemples...

```
#!/bin/bash
```

```
clear
```

```
echo -n "Entrez votre nom "
```

```
read nom
```

```
echo "Vous etes $Nom votre login est $LOGNAME"
```

```
echo "il y a $# args et le 3ième est $3"
```

```
echo "ce script s'appelle $0"
```

Résultat de `/tmp/test.sh *`

```
Entrez votre nom bob
```

```
Vous etes votre login est sylvain
```

```
il y a 9 args et le 3ième est pulse-  
2L9K88eMlGn7
```

```
ce script s'appelle /tmp/test.sh
```

# L'art du Quoting

- Les quotes offrent de multiples possibilités
- " \$var est remplacé par son contenu, \* aussi : c'est la substitution de var "
- ' cite très strictement ! '
- \ despécialise
- D'où " ' " et ' " '
- Le back quote exécute et substitue
- nb=`who | grep sylvain | wc -l`



# Quoting et dé-spécialisation

- `echo "$var"`
- `echo "\$var vaut $var"`
- `var=`ps ax | wc -l``
- `var=" je compte `ps ax | wc -l` "`
- `echo " la guillemet c'est \" "`
- `Echo '$var ne marche pas'`

# Quoting : Usage

- En général, on utilise `"`
- Les variables sont substituées
- On utilise les ``` (back quotes) ``` pour créer des variables contenant des résultats d'opérations
- L'usage de l'apostrophe est plus rare
- Pour éviter le problème d'une variable non affectée, on peut utiliser `"$var"` (ainsi, on a `""` au lieu de rien)

# Les opérations

- Pas très naturelles en shell
- Utilisation de **let**
  - **let** j=\$i+4
- Ou encore expr dans un sous-shell
  - j=`**expr** \$i + 4`
- ATTENTION AUX ESPACES ICI (et pas au dessus)
  - `**expr** \$i+4` donne 5+4

# Structures de contrôle

- Le Shell est un langage de programmation qui offre lui aussi des structures de contrôle du déroulement du programme
  - **for** *var* in *liste*
  - **do**
    - *Bloc*
  - **done**
  - **while** *cond*
  - **do**
    - *Bloc*
  - **done**

# For

- **For** var in *{liste}*
- Très pauvre par rapport au C
- Prévue pour itérer sur des listes (de fichiers par exemple)

```
for j in *.c
do
    gcc $j
done
```

# Et pour faire un vrai For alors ?

- Pas de sens en shell !!
- Mais on peut toujours
  - Utiliser **seq**
    - `for i in `seq 0 5``
    - `for i in `seq 5 10 55``
  - Utiliser la forme avancée du for
    - `for (( i=5 ; i<=55 ; i+=10 ))`

# Exemples de for

```
#!/bin/bash  
for i in /home/*  
do  
    cp /tmp/Fichier $i  
    echo « copie dans $i »  
done
```

```
#!/bin/bash  
cd /home/  
for i in *  
do  
    rm $i/Fichier  
    echo «suppr dans $i »  
done
```

```
#!/bin/bash  
for i in ` ps ax | grep firefox | tr -s ' ' | cut -f2 -d' ' `  
do  
    kill $i > /tmp/proc$i  
done
```

# Exemples de for

```
#!/bin/bash
clear
echo -n "Wait : "

for ((i=9;i>=0;i--))
do
    sleep 1
    echo -ne "\b$i"
done

clear
echo "Starting"
```

```
#!/bin/bash
service[1]="compta"
service[2]="R&D"
service[3]="prod"

for((i=1;i<4;i++))
do
    echo "Creation de ${service[$i]}"
    mkdir ${service[$i]}

    echo "copie des fichiers"
    cp ~reference/* ${service[$i]}
done
```



# Structures de contrôle

**if *condition***

**then**

*Bloc*

**else**

*Bloc*

**fi**

*sur une ligne, ne pas oublier le séparateur de commande*

**if *condition* ; then *bloc* else *bloc* fi**

*autre forme de la commande if*

**if *condition* ; then *bloc* elif *condition* ... else *bloc* fi**

# LE IF DU SHELL EST SPECIFIQUE

- Il sert à tester si une **commande a fonctionné**
- Il teste la valeur de retour du main de la commande (le return 0 ; que vous mettez en C, ou le exit 0 en Shell, etc etc)
- Toutes les commandes ont normalement une valeur de retour (accessible dans \$?)
- Pour le Shell, 0 veut dire « ok » et une autre valeur indique une erreur

# Valeur de retour des commandes

PING(8)

System Manager's Manual: iputils

PING(8)

## NAME

ping, ping6 - send ICMP ECHO\_REQUEST to network hosts

## SYNOPSIS

ping [-LRUbdfnqrvVaAB] [-c count] [-m mark] [-i interval] [-l preload]

[.....]

If ping does not receive any reply packets at all it will exit with code 1. If a packet count and deadline are both specified and fewer than count packets are received by the time the deadline expires it will also exit with code 1. On other errors it exits with code 1. Otherwise it exits with code 0. This makes ping a useful code to see if a host is alive or not.

```
#!/bin/bash
```

```
if ping -c1 www.free.fr
```

```
then
```

```
    echo "Le serveur répond"
```

```
else
```

```
    echo "le serveur ne répond pas"
```

```
fi
```

# Valeur de retour des commandes

SCP(1)

BSD General Commands Manual

SCP(1)

## NAME

scp — secure copy (remote file copy program)

## SYNOPSIS

scp [-12346BCpqr] [-c cipher] [-F ssh\_config] [-i identity\_file]  
[-l limit] [-o ssh\_option] [-P port] [-r] [-S program]  
[[user@]host1:]file

## DESCRIPTION

scp copies files between  
[.....]

## EXIT STATUS

The scp utility exits 0

```
#!/bin/bash
```

```
if scp -i credentials.fic $1 gilles@igm.univ.fr:/opt/tests/  
then
```

```
    echo "Copie du fichier $1 sur le serveur igm"
```

```
else
```

```
    echo "problème de copie de $1"
```

```
fi
```

# Le IF teste le succès de l'ensemble

- Erreur habituelle : passer par une variable numérique : C'est inutile pour le shell...

```
#!/bin/bash  
if who | grep root >/dev/null  
then  
    echo "Le root est connecté"  
    if ps -au root | grep java  
    then  
        echo "et il utilise java"  
    fi  
fi
```

# Condition ???

- **If condition ??**
- En shell la condition veut dire : *commande exécutée !*
- **If** *cp ...* / **if** *rm ...* / **if** *gcc...*
- **If** teste en fait la valeur de retour de la commande (**\$?**)
- Il a fallut inventer une commande pour tester (**test** ou **[ ]** )

# Test (ou [ ])

Fonctionne (*cad renvoie vrai*) si les arguments donnés sont **vérifiés**.

C'est une **commande** : Espace avant et après.

Options :

**==** ou **!=** pour les chaînes  
**-eq -lt -gt -le -ge** pour les nombres

# Exemple avec test

```
#!/bin/bash
if [ $UID -eq 0 ]
then
    echo "Bonjour Root"
fi
```

```
#!/bin/bash
if [ $LOGNAME = "root" ]
then
    echo "Bonjour Root"
fi
```

ATTENTION !!

```
if [ $nom = "morpheus" ] # DANGER
```

if [ "\$nom" = "morpheus" ] # MIEUX

```
#!/bin/bash
if [ $# -lt 3 ]
then
    echo "au moins 3 args attendus"
    exit 1
fi
```

```
#!/bin/bash
if [ `who | wc -l` -le 3 ]
then
    echo "il n'y a pas grand monde"
fi
```



# Test et le file system

- Test offre aussi des commandes spécifiques pour le file system
- Il est possible ainsi de savoir si un fichier est exécutable, vide, si c'est un répertoire, etc...

-e file : le fichier existe  
-f file : c'est un fichier régulier  
-s file : sa taille est > 0  
-d file : c'est un répertoire  
-x file : c'est un executable  
-r file : il est lisible par ce shell  
-w file : ce shell peut y écrire

file -nt file2 : file est plus récent que file2  
file -ef file2 : file et file2 sont la même inode

# Exemple avec File System

```
#!/bin/bash
if [ ! -e truc ]
then
    mkdir truc
    if [ -s bidule -a -x bidule ]
    then
        cp bidule truc
    else
        touch truc/bidule
        chmod +x truc/bidule
    fi
fi
```

```
#!/bin/bash

ln /etc/apache.conf ~/serveur.conf

if [ /etc/apache.conf -ef ~/serveur.conf ]
then
    echo "le lien est bien créé"
fi

# pouvez-vous écrire quelque chose
# de plus simple ?
```

# Structures de contrôle switch

**Case \$var in  
1)**

***Bloc***

**;;**

**Binou)**

***Bloc***

**;;**

**esac**

# Exemple Script du système

```
case $COMMAND in
status)
    echo
    echo "Since the script ...."
    echo "Upstart job, you may also ..."
    source "$JOB"
    ;;
start|stop)
    echo
    echo "Since the script you are attempting ..."
    if status "$JOB" 2>/dev/null | grep -q ' start/'
    then
        RUNNING=1
    fi
    if [ -z "$RUNNING" ] && [ "$COMMAND" = "stop" ]; then
        exit 0
    ...
```

# Boucle while

```
# Afficher les entiers de 1 à 20
```

```
(( i = 1 ))
```

```
while (( i <= 20 ))
```

```
do
```

```
    echo $i
```

```
    (( ++i ))
```

```
done
```

```
while [ $a -le "$LIMIT" ]
```

```
do
```

```
    a=$(( $a+1 ))
```

```
    if [ "$a" -eq 3 ] || [ "$a" -eq 11 ]
```

```
    # Excludes 3 and 11.
```

```
    then
```

```
        continue
```

```
        # Saute à l'étape suivante.
```

```
    fi
```

```
    echo -n "$a "
```

```
done
```

# Manipulation de chaines

- grep : Recherche de motif
- sed : remplacement de motifs

» grep root /etc/passwd

*root:x:0:0:root:/root:/bin/bash*

» date

*dimanche 9 mai 2010, 19:23:23 (UTC+0200)*

» date=`date | sed -e 's/,.\*//'`

» echo \$date

*dimanche 9 mai 2010*

# Arithmétique

Expr : calcule le résultat d'une expression

» `var=`expr $var + 1 ``

» `var = 4 ; expr $var % 2` (renvoie 0!)

» `expr 1 + 1`

2

» `expr 15 mod 4`

3

On peut aussi utiliser `let` : `let i=$j+5`

Ou encore `((i=$j+5))`

# Journaliser une action

- `logger [-t étiquette] [ message ... ]`
- » `logger -t MSG "coucou dans syslog"`