II – Shell, scripts & commandes

L2 informatique

INFO305 : Outils d’administration

Olivier FLAUZAC / Clément FOYER / Florent NOLOT

Shell

La ligne de commande / shell

- Moyen d’interagir avec le système
- Shell :
  - Interpréteur de commande
  - Environnement d’échange de commande
  - Langage de programmation
  - Interface avec le noyau
  - …
- Le *shell* assure l’interprétation des commandes tapées au clavier
- Terminal : environnement d’entrées / sorties

Quelques shell

- sh : *Bourne shell* shell historique
- csh / tcsh : *C Shell*
- ksh : *korn shell*
- **bash** : *Bourne Again shell linux* : le plus utilisé
- zsh : Z shell

- Shells sont accessibles par la ligne de commande : le terminal

Bash

- Créé en 1989 par Brian Fox
- Interface en ligne de commandes
- Porté sur de nombreux systèmes
- Dispose de nombreux raccourcis claviers
  - Ctrl-a / Ctrl-e : accès au début / fin de la ligne
  - Alt-f / Alt-b : avancer / reculer d'un mot
  - Ctrl-r : rechercher dans l'historique des commandes
  - Tab / Ctrl-i : complétion
  - Ctrl-z : mise en pause d'un processus en exécution
    - fg / bg : relancer le processus au premier / en arrière plan

Configuration du bash

- Exploitation de fichiers de configuration
  - .bashrc
  - .bash_profile / .profile
- Configuration
  - Du PATH
  - Du prompt
  - Des alias
  - De toutes les variables d’environnement

Fichiers de type `profile`

- Fichiers `profile` ou `.bash_profile`
- Exécuté lors d'une phase de log (une fois pas session)
- Permet de
  - Définir des alias
  - Définir des variables d'environnement
  - Modifier des variables d'environnement

Fichiers de type `bashrc`

- Exécuté à l'ouverture du shell (connexion, exécution de script …)
- `/etc/bash.bashrc`
  - Configuration de base pour tous les utilisateurs
  - Ne peut être modifié que par le super utilisateur
- `~/.bashrc`
  - Fichier local de configuration
- Permet de
  - Définir des alias
  - Définir des variables d'environnement
  - Modifier des variables d'environnement

Alias

- Substitution de commandes
- Permet de simplifier l’usage du terminal
- Intégrés dans les fichiers de configuration
- Exemple :

```
alias ll=‘ls -alF’
alias upd=‘sudo apt-get update’
```

Variables d’environnement

- Variables définies de manière générale
- Informations :
  - L’environnement
  - Les préférences utilisateur
  - Le système
  - La configuration
- Définition

```
NOM=valeur
NOM="texte de la valeur"
NOM=valeur1:valeur2:valeur3
```

- Exploitation

```
$NOM
```

Quelques variables

- HOME
- HOSTNAME
- LOGNAME
- PATH
- PWD
- USER

```
olivier@info0305:~$ echo $HOME
/home/olivier
olivier@info0305:~$ echo $HOSTNAME
info0305
olivier@info0305:~$ echo $LOGNAME
olivier
olivier@info0305:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games:/snap/bin
olivier@info0305:~$ echo $PWD
/home/olivier
olivier@info0305:~$ echo $USER
olivier
olivier@info0305:~$
```

Le prompt

- Invite de commande
- Variable d'environnement PS1

```
olivier@info0305:~/Toto$ echo $PS1
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$
```

- Définition
  - Des informations affichées
    - HOSTNAME, USER, PWD …
  - De la couleur
  - Du format
  - …

Scripts

Script shell

- Fichier texte
- Contenant une suite de commandes *shell*
- Exécutable par un interpréteur
- Retournant dans tous les cas une valeur
  - Code de bon fonctionnement (0)
  - Code d’erreur (!= 0)
  - Différenciation des erreurs !
  - Cas général des commandes linux

Scripts élémentaires

Scr1.sh

```
#!/bin/bash
# un commentaire
echo "Salut les amis"
exit 0
```

```
olivier@info0305:~/Scripts$ chmod a+x scr1.sh
olivier@info0305:~/Scripts$ ./scr1.sh
Salut les amis
olivier@info0305:~/Scripts$
```

Exécution d’une commande

- Utilisation de $(…)
- Exploitation de la chaîne retournée
  - Affichage
  - Stockage

```
#!/bin/bash
echo "nous somme le $(date)"
exit 0
```

Les variables

- Variables non typées et non déclarées
- Référencement de la valeur : $
- Variable en lecture `readonly`
- Exploitation des variables d'environnement
  - `$USER`,`$PWD`,`$HOME` …
- Liste des variables d'environnement : `env`

Utilisation des variables

```
#!/bin/bash

variable="bonjour les amis"
echo $variable

listeFic=$(ls)
echo $listeFic

echo "Vous êtes : $USER"
echo "Répertoire d'exécution : $PWD"
echo "Répertoire de base : $HOME"
exit 0
```

Variables prédéfinies

`$?`	valeur de sortie de la dernière commande
`$0`	nom du script
`$n`	Nième argument passé au script
`$#`	nombre d'arguments
`$*`	liste des arguments à partir de $1
`$$`	PID du processus courant

```
#!/bin/bash

echo "ce script s\'appelle" $0
echo "vous avez passé l'argument" $1
./var.sh
echo "la dernière commande a donné :" $?

exit 0
```

Opérations sur les variables

```
#!/bin/bash

var1=0
echo $var1  #0
let var1++
echo $var1  #1
let var1--
echo $var1  #0

var2=45
var2=$var2+359
echo $var2  #45+359

var3=4
var3=$(($var3+359))
echo $var3  #363

var4=7
var4=$[$var4+3]
echo $var4  #10
```

```
#!/bin/bash

v1="bonjour "
v2=$v1$USER
echo $v2

v3="$USER utilisateur sur $HOSTNAME"
echo $v3
exit 0
```

Tableaux

- Déclaration globale ou individuelle

```
tab1 = ("toto" "titi")
tab1[0] = "bob"
tab1[1] = "alice"
```

- Affichage

```
echo ${tab1[0]}
echo ${tab[@]}
```

- Taille

```
echo ${#tab1[*]}
len=${#tab1[*]}
```

Tests

- Utilisation dans le cadre des conditionnelles et des itérations
- Deux syntaxes
  - test expression
  - [expression]
- Tests des éléments système
  - types de fichiers
- Tests sur les valeurs
  - Chaînes de caractères
  - Valeurs numériques

Tests sur les fichiers

-e	le fichier existe
-f	c’est un fichier
-d	c’est un répertoire
-r | -w | -x	propriété du fichier
-s	non vide
f1 –nt f2	Plus récent que
f1 –ot f2	Plus vieux que

Test des valeurs

- Chaines de caractères

-z / -n	Chaîne vide / non vide
!= / =	Différentes identiques

- Valeurs numériques

-eq / -ne	Égal / différent
-lt / -gt	Strictement inférieur / strictement supérieur
-le / -ge	Inférieur ou égal/ supérieur ou égal

Expressions logiques

expr1 –a expr2	Et
expr1 –o expr2	Ou
! Expr	Négation

Structure conditionnelle : if

```
if
then
    instructions
elif test2
then
    instructions
else
    instructions
fi
```

```
#!/bin/bash

if [ $# -ne 1 ]
then
  echo usage : $0 argument
  exit 1
fi
echo Le paramètre est : $1
exit 0
```

Structures conditionnelles : case

```
case valeur in
    expr1) commandes ;;
    expr2) commandes ;;
 ...
esac
```

```
#!/bin/bash

case $# in
    0) echo aucun paramètre ;;
    1) echo 1 paramètre $1 ;;
    2) echo 2 paramètres $1 - $2 ;;
    3) echo 3 paramètres $1 - $2 - $3 ;;
esac
exit 0
```

```
#!/bin/bash


case $USER in
    root) echo hello root ;;
    olivier) echo bonjour olivier ;;
    *) echo salut autre ;;
esac
exit 0
```

Structures itératives : for

```
for variable [in liste]
do
 commandes
done
```

```
#!/bin/bash
if [ $# -le 3 ]
then
  echo au moins 4 arguments
  exit 1
fi
echo Les paramètres sont :
for var in $*
do
  echo $var
done
exit 0
```

```
#!/bin/bash
if [ $# -ne 1 ]
then
  echo usage $0 repertoire
  exit 1
fi
for f in $(ls $1)
do
  if [ -f $1/$f ]
  then
  echo $f est un fichier
elif [ -d $1/$f ]
then
  echo $f est un répertoire
fi
done
exit 0
```

Structures itératives : while

```
while [test]
do
 commandes
done
```

```
#!/bin/bash
cpt=0
while [ $cpt -le 10 ]
do
  echo "encore 1 : " $cpt
  let cpt++
done
```

La gestion des flux

- Utilisation des flux
  - < entrée standard depuis un fichier
  - > ou 1> flux standard
  - 2> flux d'erreur
  - 2>&1 redirection de la sortie d'erreur vers la sortie standard
- Redirection dans un fichier
  - > écriture et création du fichier
  - >> ajout à un fichier

Le cas du clavier

- Ecriture de scripts interactifs
- Lecture au clavier
- Utilisation de `read`
- Affectation d'une liste de variables

```
#!/bin/bash
read v1 v2 v3
echo "vous avez entré $v1"
echo "vous avez entré $v2"
echo "vous avez entré $v3"
```

Exploitation d’un fichier en sortie

- Exploitation de la redirection
- Ecriture avec `echo` dans le fichier

```
#!/bin/bash
list=$(ls)
for element in $list
do
  echo $element >> ./toto.txt
done
```

Lecture d’un fichier

- Lecture de la totalité d’un fichier dans une variable
  - Exploitation de cat
  - Exploitation de la redirection

```
#!/bin/bash
list=$(<toto.txt)
for element in $list
do
  echo $element >> ./toto.txt
done
```

```
#!/bin/bash
list=$(cat toto.txt)
for element in $list
do
  echo $element >> ./toto.txt
done
```

Lecture ligne à ligne

- Utilisation de
  - Boucle `while`
  - `read`
  - redirection

```
#!/bin/bash
cpt=1
while read LINE
do
  echo $cpt ":" $LINE
  let cpt++
done < toto.txt
```

« Exportation » des variables

- Problématique de portée des variables
- Définition par défaut pour le processus courant

```
olivier@info0305:~$ VAR1=olivier
olivier@info0305:~$ echo $VAR1
olivier
olivier@info0305:~$
```

- Pas de portabilité dans les processus fils

```
#!/bin/bash

echo  "valeur de toto est : $VAR1"
```

```
olivier@info0305:~$ ./exp.sh
valeur de toto est :
olivier@info0305:~$
```

Exportation

- Accès des variables dans les processus fils

```
olivier@info0305:~$ export VAR1=olivier
olivier@info0305:~$ echo $VAR1
olivier
olivier@info0305:~$ ./exp.sh
valeur de toto est : olivier
```

- A utiliser dans les fichiers de configuration de bash !

Fonctions

- Eléments de structuration de programmes
- Pas toujours (bien) utilisées
- Script souvent == programmation « sale »
- Exploitation standard
  - Déclaration
  - Paramétrage
  - Appel

Fonctions : déclaration et appel

```
#!/bin/bash

#déclaration
listFic(){
  echo la liste des fichiers est :
  ls -l
}

echo appel à la fonction
#appel
listFic
```

Fonction : paramètres

- Pas de déclaration dans le prototypage
- Exploitation des paramètres comme pour les scripts $1 , $2 …

```
#!/bin/bash

listFic(){
  echo la liste des fichiers dans $1 est :
  ls -l $1
}

echo Donnez le chemin à explorer :
read path
echo appel à la fonction
listFic $path
```