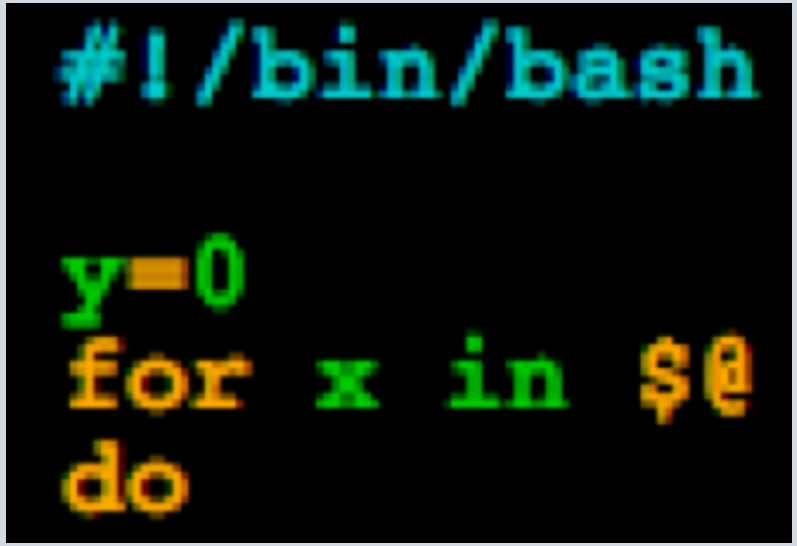

Introduction aux scripts Bash

A111 - SYSTÈMES

Langage

Variables

Instructions de contrôle : if, for, while, …

Opérateurs de combinaison : ';', '||', ' && '

Redirections : '<', '>', '|'

Script

Programme (fichier texte) qui regroupe des commandes de l'interpréteur

Exécuter des tâches d'administration répétitives

Portabilité et pérennité

Squelette de script bash

FICHIER SCRIPT.SH	DESCRIPTION
#! /bin/bash	sha-bang chemin vers l'interpréteur
# commentaire	
commandes …	et instructions de contrôle
	Le fichier doit avoir les droits d'exécution pour l'utilisateur et il faut préciser le PATH
	chmod u+x script.sh
	./script.sh [arguments…]

Hello World!

```
#! /bin/bash
# script bash hello.sh
# A111 - Novembre 2015
echo "Hello World !"
```

Commande echo : exemples

```
echo "Affiche une ligne"
echo -n "Ne passe pas à la ligne"
```

Commande echo

```
$ echo -e toto \\c
toto $ echo -e "toto \\c"
toto $ echo -e "toto \c"
toto $ echo -e toto \c
toto c
$ echo -e "toto \btiti"
totot iti
$
```

Définition de variables

Définition d'une variable

$ mavar=toto

Attention! Pas d’espaces autour du symbole =

$ echo mavar

mavar

Substitution du contenu avec le caractère $

$ echo $mavar

toto

Destruction d'une variable : unset

Commande read

```
echo -e "Entrez quelque chose : \c"
read reponse
echo "Vous avez entré : $reponse"
echo "Entrez plusieurs mots : "
read var1 var2
echo "Premier mot : $var1"
echo "Le reste : $var2"
read -s mdp
```

Variables d'environnement

Commandes set / unset

Variables :

nom / contenu

PATH / $PATH

Manipulation :

echo $PATH

export PATH=~/bin:$PATH

Code retour d'une commande : $?

Toutes les commandes retournent un code au processus appelant : 0 ou code erreur.

0 tout s’est bien passé, un autre entier sinon

ls ; echo $?

mkdir ; echo $?

mkdir /tmp ; echo $?

Chaînes de caractères

Type par défaut

- toto=bonjour

Protection avec des " "

- echo "$toto $USER"

>bonjour bilancin

Protection avec des ' '

- echo '$toto $USER'

>$toto $USER

**Utilisation de **

- echo "$toto \$USER"

> bonjour $USER

combinaison "et '

- echo "'$toto \$USER'"
- >'bonjour $USER'

my_name="Fred Smith" # Set a variable
echo "$my_name" # Will output - Fred Smith
echo '$my_name' # Will output - $my_name

Expressions numériques

Par défaut : variable = chaîne de caractères

Déclaration inutile (mais attention : pas d'espace entre le nom et =)

Initialement vide

Variables entières

typeset –i var1 [var2 ...] : var1 et var2 de type entier (integer)

i=4; j=i+3; echo $j

typeset –i i j

i=4; j=i+3; echo $j

Expressions numériques

Par défaut : variable = chaîne de caractères

Déclaration inutile (mais attention : pas d'espace entre le nom et =)

Initialement vide

Variables entières

typeset –i var1 [var2 ...] : var1 et var2 de type entier (integer)

i=4; j=i+3; echo $j

typeset –i i j

i=4; j=i+3; echo $j

```
vagrant@vagrant:/$ i=4;j=i+3;echo $j
i+3
vagrant@vagrant:/$ typeset -i i j
vagrant@vagrant:/$ i=4;j=i+3;echo $j
7
```

Commande expr ou (())

Évaluation d'expressions arithmétiques : délimitées par des doubles parenthèses ou expr

Doit être un seul mot pour le SHELL

- `((i1 = $i2 +1))`
- `i1=$(expr $i2 + 1)# ici pas d'espace autour du 1er = mais des espaces obligatoires autour de +`
- `let i1=i2+6 # pas d’espaces`

http://abs.traduc.org/abs-fr/ch08.html#ops

Substitution de commande

De la forme $(CMD) ou `CMD`

La forme est substituée par le résultat de CMD

Exemples :

- `echo "Répertoire courant : `pwd`"`
- `echo "Votre groupe est : $(id -gn )"`
- `mv foo foo_$(date +%d_%m_%Y)`

Les éventuels sauts de lignes en fin de résultat sont éliminés

Tableaux

Déclaration implicite lors de l'initialisation :

- `tab=( "Un" "Deux" "Trois" "Quatre")`

Indices : de 0 à N-1,

- `echo ${tab[0]} => "Un"`
- `echo ${tab[4]} =>`

Autres manipulations

- `echo ${tab[@]}  => Un Deux Trois Quatre`
- `echo ${!tab[@]} => 0 1 2 3`
- `echo ${#tab[@]} => 4`

Commande test

TEST(1) Commandes TEST(1)

NOM

test - Vérifier le type d'un fichier, et comparer des valeurs

SYNOPSIS

test EXPRESSION

test

[EXPRESSION]

[ ]

[OPTION

test

```
$ ls -l
total 20
drwxr-xr-x 2 syska personnel 4096 oct.  20 17:39 chapitre-01
drwxr-xr-x 2 syska personnel 4096 oct.  20 17:39 chapitre-02
drwxr-xr-x 2 syska personnel 4096 oct.  14 11:39 chapitre-03
drwxr-xr-x 2 syska personnel 4096 oct.  14 11:39 chapitre-04
-rwxr-xr-x 1 syska personnel  923 oct.  12 14:36 FICHIERS
```

```
$ test -f chapitre-01
$ echo $?
1
$ test -f FICHIERS
$ echo $?
0
$ test -d FICHIERS
$ echo $?
1
$
```

Instruction de contrôle if then else fi

Toutes les commandes admettent un résultat vrai ou faux:

```
if mkdir /foo
then
  echo 'réussite!'
else
  echo 'échec!'
fi
if mkdir /foo ; then echo 'réussite!' ; else echo 'échec!' ; fi
```

ou mieux :

```
if mkdir /foo 2> /dev/null; then echo 'réussite!' ; else
echo 'échec!' ; fi
```

test if [...]

```
if test ...
then ...
fi
```

est équivalent à :

```
if [ ... ]
then ...
fi
```

Documentation : man test

Opérateurs pour test / [ ]

Tests logiques :

-a : AND, -o : OR, ! : NOT, (EXPR) ...

Tests arithmétiques :

-le, -lt, -gt, -ge, -ne, -eq

Tests de fichiers :

-f, -x, -d, -l, -s, ...

Tests de chaînes :

STR1 = STR2, STR1 != STR2, -z STR (vide), -n STR (non vide)

Les opérateurs de tests disponibles sont, pour les les objets du système de fichiers :

- `[ -e $FILE ]`

vrai si l'objet désigné par $FILE existe dans le répertoire courant,

- `[ -s $FILE ]`

vrai si l'objet désigné par $FILE existe dans le répertoire courant et si sa taille est supérieure à zéro,

- `[ -f $FILE ]`

vrai si l'objet désigné par $FILE est un fichier dans le répertoire courant,

- `[ -r $FILE ]`

vrai si l'objet désigné par $FILE est un fichier lisible dans le répertoire courant,

- `[ -w $FILE ]`

vrai si l'objet désigné par $FILE est un fichier inscriptible dans le répertoire courant,

- `[ -x $FILE ]`

vrai si l'objet désigné par $FILE est un fichier exécutable dans le répertoire courant,

- `[ -d $FILE ]`

vrai si l'objet désigné par $FILE est un répertoire dans le répertoire courant.

Exemples test et [ ]

```
if [ -f java.tar ]
then
      echo "OUI"
fi
```

```
if [ -f java.tar -o -d home ]
then
echo "OUI"
fi
```

```
A="Toto"; B="Toto"
if [ "$A" = "$B" ]
then
      echo "OUI"
fi

if [ -z $B ]
then
      echo "OUI"
else
      echo "NON"
fi
```

Test en bash ou zsh [[…]]

Syntaxe proche du langage C

STR == PATTERN

STR < PATTERN (ordre alphanumérique)

STR > PATTERN

EXP1 && EXP2 : ET logique

EXP1 || EXP2 : OU logique

Tests supplémentaires

-o OPTION : option zsh positionnée

Attention : quelques différences avec test

-a FILE : fichier existant (au lieu du ET logique)

Exemples [[ ]]

```
if [[ -d home && -e java.tar ]]
then
        echo 'OUI'
fi
```

ou

```
[[ -d home && -e java.tar ]] && echo 'OUI!'
```

```
if [ -d home -a -e java.tar ]
then
        echo 'OUI'
fi
```

ou

```
[ -d home -a -e java.tar ] && echo 'OUI!'
```

Exemples [[ ]] (suite)

```
A="Toto"
if [[ $A == T* ]]
then
        echo "OUI"
else
        echo "NON"
fi
OUI
```

```
if [[ $A == t* ]]
then
        echo "OUI"
else
        echo "NON"
fi
NON
```

Un exemple Wikipédia

```
#!/bin/bash
read -p "Si vous etes d'accord entrez o ou oui : " reponse
if [ ! "$reponse" = "o" -a ! "$reponse" = "oui" ]; then
    echo "Non, je ne suis pas d'accord !"
else
    echo "Oui, je suis d'accord"
fi
```