

Programmation Shell

1 ère année Licence Informatique

ISIMM

2022/2023

Exemple 1

Réaliser un script qui renvoie les messages suivants avec un décalage d'une seconde. Nommer ce script « firstday »

- - good morning Aroua
- - you're looking great today Aroua
- - you are the best student ever Aroua

Exemple 1

Shebang

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

```
echo "Good morning Aroua"
```

```
sleep 1
```

```
echo "you're looking great today Aroua"
```

```
sleep 1
```

```
echo "you are the best student Aroua"
```

Affichage du message

Décalage de l'affichage
d'une seconde

Exécuter le script

Rendre le script exécutable

```
:~$ chmod +x firstday
```

```
sana@sana-VirtualBox:~$ ./firstday  
Good morning Aroua  
you're looking great today Aroua  
you are the best student Aroua
```

Exemple 2

- Maintenant, nous allons introduire une variable : name

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

```
name="Adam"
```

```
echo "Good morning $name"
```

```
sleep 1
```

```
echo "you're looking great today $name"
```

```
sleep 1
```

```
echo "you are the best student $name"|
```

Création d'une variable

Introduire la variable manuellement

```
sana@sana-VirtualBox:~$ ./firstday  
Good morning Adam  
you're looking great today Adam  
you are the best student Adam
```

Utiliser la variable
C'est le contenu de la variable

Exemple 3

```
#!/bin/bash
echo "what is your name ?"
read name
echo "Good morning $name"
sleep 1
echo "you're looking great today $name"
sleep 1
echo "you are the best student $name"
```

```
sana@sana-VirtualBox:~$ ./firstday
what is your name ?
Myriam
Good morning Myriam
you're looking great today Myriam
you are the best student Myriam
```



Get the user input

Exemple 4

Utiliser un argument positionnel ou un paramètre positionnel

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

```
name=$1
```

```
echo "Good morning $name"
```

```
sleep 1
```

```
echo "you're looking great today $name"
```

```
sleep 1
```

```
echo "you are the best student $name"
```

```
sana@sana-VirtualBox:~$ ./firstday Ahmed
Good morning Ahmed
you're looking great today Ahmed
you are the best student Ahmed
```

Exemple 5

Utiliser deux arguments positionnels

```
#!/bin/bash  
name=$1  
adj=$2  
  
echo "Good morning $name"  
sleep 1  
echo "you're looking great today $name"  
sleep 1  
echo "you are the best student $name and you're working $adj"
```

```
sana@sana-VirtualBox:~$ ./firstday Maha hard  
Good morning Maha  
you're looking great today Maha  
you are the best student Maha and you're working hard
```

Variables d'environnement

- ➔ **HOME** : contient le chemin absolu du répertoire de connexion de l'utilisateur
- ➔ **LOGNAME** : contient le nom de connexion de l'utilisateur
- ➔ **PATH** : contient la liste des répertoires contenant des exécutable séparés par ' : '. Ces répertoires seront parcourus par ordre à la recherche d'une commande externe
- ➔ **SHELL** : contient le chemin d'accès absolu des fichiers programmes du shell

```
root@localhost:~# echo $SHELL
/bin/bash
root@localhost:~# echo $USER
root
root@localhost:~# echo $PWD
/root
```


Exemple 6

Tester quelques variables d'environnement

```
#!/bin/bash
|
echo "What is your name ?"

read name

echo "how old are you ?"

read age

echo "Hello $name, you are $age years old."

echo "$PWD, $SHELL, $USER, $HOSTNAME"
```

```
sana@sana-VirtualBox:~$ ./firstday
What is your name ?
iron man
how old are you ?
65
Hello iron man, you are 65 years old.
/home/sana, /bin/bash, sana, sana-VirtualBox
sana@sana-VirtualBox:~$
```

Exemple 7

Ajouter une variable utilisateur

Comment ajouter la variable twitter à notre script ?

```
sana@sana-VirtualBox:~$ twitter="Elon Musk"
sana@sana-VirtualBox:~$ echo $twitter
Elon Musk
```

```
sana@sana-VirtualBox:~$ export twitter
```

```
#!/bin/bash

echo "What is your name ?"
read name

echo "how old are you ?"
read age

echo "Hello $name, you are $age years old."
echo "$PWD, $SHELL, $USER, $HOSTNAME"
echo $twitter
```

```
sana@sana-VirtualBox:~$ ./firstday
What is your name ?
Flash
how old are you ?
23
Hello Flash, you are 23 years old.
/home/sana, /bin/bash, sana, sana-VirtualBox
Elon Musk
```

Sh vs bash vs ksh vs csh

sh	bash	ksh	csh	Meaning/Action
\$	\$	\$	%	Default user prompt
	>	>	>!	Force redirection
> file 2>&l	&> file or > file 2>&l	> file 2>&l	>& file	Redirect stdout and stderr to file
	{ }		{ }	Expand elements in list
`command`	`command` or \$(command)	\$(command)	`command`	Substitute output of enclosed command
\$HOME	\$HOME	\$HOME	\$home	Home directory
	~	~	~	Home directory symbol
	~+, ~-, dirs	~+, ~-	=-, =N	Access directory stack
var=value	VAR=value	var=value	set var=value	Variable assignment
export var	export VAR=value	export var=val	setenv var val	Set environment variable
	\${nnnn}	\${nn}		More than 9 arguments can be referenced
"\$@"	"\$@"	"\$@"		All arguments as separate words
\$#	\$#	\$#	\$#argv	Number of arguments
\$?	\$?	\$?	\$status	Exit status of the most recently executed command
\$!	\$!	\$!		PID of most recently backgrounded process
\$-	\$-	\$-		Current options
. file	source file or . file	. file	source file	Read commands in file
	alias x='y'	alias x=y	alias x y	Name x stands for command y
case	case	case	switch or case	Choose alternatives
done	done	done	end	End a loop statement
esac	esac	esac	endsw	End case or switch

Shell

- Langage de programmation interprété
→ Interpréteurs de commande standard des systèmes UNIX
- Utilisations :
 - ① Ouverture de session
 - ② Dialogue avec le système d'exploitation par l'intermédiaire d'un interpréteur de commandes
 - ③ Scripts
- Commentaires sur les shells :
 - Avantages : clair et facile à relire
 - Inconvénient : demande de la rigueur lors de la programmation (pas de pré-compilation)

Familles Shell

Nombreux shells répartis en 2 familles :

- Famille des Bourne Shells (Steve Bourne) : sh, bash
- Famille des C Shells : csh, tcsh

Autres:

- ksh (David Korn) : famille des Bourne Shells avec améliorations issues des C Shells

Présentation du Bourne Shell et de sa programmation

Prédicat test

test (1)

- Permet des tests de fichiers et de comparaison élaborés
- Retour :
 - 0 si la condition est vraie
 - différent de 0 si la condition est fausse

- Syntaxe :

`test expression`

ou

`[expression]`

Prédicat test

test (2)

Forme de l'expression :

- `-f name` : vrai si `name` est
 - un fichier régulier (sous System V) ;
 - un fichier au sens large (fichier régulier, fichier spécial bloc ou caractère, tube nommé) et n'est pas un répertoire
- `-d name` : vraie si `name` est un répertoire

Prédicat test

test (4)

- `-z string` : vraie si `string` est une chaîne de caractères vide
- `-n string` : vraie si `string` est une chaîne de caractères non vide
- `s1 == s2` ou `s1 = s2` : vraie si les chaînes de caractères `s1` et `s2` sont égales (inégalité : `!=`)
- `n1 -eq n2` : vraie si les chaînes de caractères `n1` et `n2` contenant des nombres entiers sont égales (utilisation aussi de `-ne`, `-gt`, `-ge`, `-lt`, `-le`)
- Opérateurs logiques : `!` (NON), `-a` (ET), `-o` (OU)

Calcul arithmétique

- Réalisation d'arithmétique élémentaire :
`expr` ou `$((expression))`
- Opérateurs : `+` `-` `*` `/` `%`
- La multiplication `*` doit être déspécialisée si l'expression n'est pas quotée
- **Exemples :**

```
$ a=10
$ a='expr $a + 1'
$ # ou
$ a=$(( $a + 1 ))
$ b='expr 10 + 5 \* 4'
$ # ou
$ b=$(( 10 + 5 \* 4 ))
$ echo "$a ; $b"
11 ; 30
```

Structure conditionnelle : if

Syntaxe :

(else facultatif)

```
if condition
then
    liste des commandes
else
    liste des commandes
fi
```

(if multiples)

```
if condition1
then
    liste des commandes
elif condition2
then
    liste des commandes
else
    liste des commandes
fi
```

Structure conditionnelle : if exemple

```
if test $# != 1
then
    echo "Zero ou plusieurs arguments"
else
    echo "Un argument : $1"
fi
```

Structure conditionnelle : case

Syntaxe :

```
case identificateur in
    motif1)
        liste de commandes
        ;;
    motif2)
        liste de commandes
        ;;
    motif3|motif4)
        liste de commandes
        ;;
    ...
    motifn)
        liste de commandes
        ;;
esac
```

Structure conditionnelle : case exemple

```
case $1 in
    Sun) echo "Dimanche";;
    Mon) echo "Lundi";;
    Tue) echo "Mardi";;
    Wed) echo "Mercredi";;
    Thu) echo "Jeudi";;
    Fri) echo "Vendredi";;
    Sat) echo "Samedi";;
    *)   echo "Jour inconnu";;
esac
```

Structure it rative : for

- Syntaxe :

```
for identificateur in liste_de_valeurs
do
    liste de commandes
done
```

```
#!/bin/bash
for i in 1 2 3 4 5
do
    echo "Hello $i"
done
```

- Exemple :

```
for i in 1 2 3 4 5
do
    echo $i
done
```

```
for i in `seq 1 5`
do
    echo $i
done
```

Structure itérative : Boucle While

- Syntaxe :

```
while condition
do
    liste de commandes
done
```

- Exemple :

```
i=1
fin=10
while [ $i -le $fin ]
do
    echo $i
    i='expr $i + 1'
done
```

Structure itérative : Boucle until

- Syntaxe :

```
until condition
do
    liste de commandes
done
```

- Exemple :

```
i=1
until [ $i -gt 10 ]
do
    echo $i
    i='expr $i + 1'
done
```