

3. Shell Bash

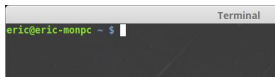
Eléments de base
Script Bash

Shell Bash

Bash

► Shell, comportement d'une boucle infinie :

- affichage du prompt (\$) d'attente du clavier
- lecture d'une commande (RETURN)
- analyse syntaxique (découpe en mots)
- substitution (caractères spéciaux)
- exécution de la commande
- écriture des résultats sur son terminal d'attache



► Bash définit un langage : le langage bash

- Variables
- Structures algorithmiques classiques (if, while, for, etc.)

► Accès rapide aux mécanismes offerts par le niveau exécutif du système d'exploitation (tube, fichiers, redirections, ...)

Programmation shell

Texte Bash

- ▶ Un **texte** bash est formé de **mots** bash

Attention : bash est sensible à la casse

- ▶ Un mot bash
 - ▶ Formé de caractères séparés par des **délimiteurs** (espace, tabulation, retour à la ligne)

Exemple :



- ▶ Exceptions :
 - ▶ ; & && | || () `
 - sont des mots ne nécessitant pas de délimiteurs
 - ▶ si une chaîne de caractères est entourée de " " ou ' '
 - alors bash voit un unique mot

Shell Bash

Illustration 1 :

Ici nous avons 5 mots

" Alors que nous avons un unique "mot" y compris le mot milieu"

voici, trois, mots

" charabia "@é§à' mais toujours un unique mot'

nous|avons;neuf&&MOTS&ici

Shell Bash

Variables Bash



Déclaration/affectation directe d'une valeur (numérique, chaîne de caractères). Attention : aucun espace autorisé !!



Accès à la valeur contenue dans la variable



Affiche la liste des variables d'environnement prédéfinies

Exemple de variables d'environnement :

USER	: nom de l'utilisateur
PWD	: chemin et nom du répertoire courant
PS1	: chaîne apparaissant à l'invite du Shell
SHELL	: chemin et nom du programme Shell
HOME	: chemin du <i>home directory</i>
PATH	: liste des répertoires où chercher les exécutable des commandes externes

Shell Bash

Variables Bash



Saisie interactive d'une valeur affectée à variable



Affichage au terminal de la valeur d'une variable

Programmation shell

Illustration 2 :

```
$ a=42
```

```
$ echo $a
```

```
$ s='Hello world!!!'
```

```
$ echo $s
```

```
$ read x
```

```
Ceci est une phrase
```

```
$ echo $x
```

```
$ read x y
```

```
Ceci est une phrase
```

```
$ echo $x
```

```
$ echo $y
```

Programmation shell

Caractères spéciaux de Bash



caractères spéciaux ; en particulier :

	Description
#	# s'appelle <i>hashtag</i> définit un commentaire ; tout ce qui suit dans une ligne est ignoré par le shell
\	\ s'appelle <i>backslash</i> Déspecialise le caractère qui suit
'...'	' s'appelle <i>quote</i> Permet de définir une chaîne de caractères qui ne sera pas évaluée par le shell
"..."	" s'appelle <i>double quote</i> Définit une chaîne de caractères dont les variables seront évaluées par le shell
`...`	` s'appelle <i>backquote</i> Définit une chaîne de caractères qui sera interprétée comme une commande et remplacée par la chaîne qui serait renvoyée à l'exécution de la dite commande

Illustration 3 :

```
eMail=Xavier.Gandibleux@univ-nantes.fr  
moi=Xavier  
vous=L1  
phrase1="$moi dit «attentif les $vous»"  
phrase2='$moi dit «attentif les $vous»'  
echo $phrase1
```

```
echo $phrase2
```

```
rep='pwd'  
echo $rep
```

Shell Bash

Script bash – première partie : script simple

Un script bash, c'est :

- ▶ un **texte bash** dans un **fichier texte**
- ▶ première ligne doit obligatoirement être `#!/bin/bash`

Exemple :

```
#!/bin/bash  
commande1  
commande2
```

Enregistré sous le nom (par convention, extension .sh) :

`mon_script.sh`

- ▶ fichier exécutable (droit x) : `chmod_u+x_mon_script.sh`
- ▶ sortie implicite du script à la fin du fichier
 - ▶ sortie explicite avec la commande `exit_[numero]`


Shell Bash

Exemple

```
#!/bin/bash
#
# Premier shell-script
# Fevrier 2019
#
message='bonjour'
echo $message
exit 0
```

Invoquer le script nommé par exemple `mon_script.sh` avec :

```
./mon_script.sh
```



```
bonjour
```

Programmation shell

Script bash – seconde partie : script avec arguments

Exécuter un script en lui passant un certain nombre de paramètres (ou arguments) :

```
nomScript arg1 arg2 ... argN
```

Arguments stockés dans des variables spéciales :

Variable	Description
\$0	Nom de la commande
\$#	Nombre d'arguments
\$*	Liste des arguments (en 1 mot)
@	Liste des arguments (en n mots)
\$1	Valeur du premier paramètre
:	
\$9	Valeur du neuvième paramètre

Programmation shell

Script bash – seconde partie : script avec arguments

exemple 1 : **echo L1 fonctionnement des ordinateurs**
 commande argument1 argument2 argument3 argument4

commande ← echo
argument1 ← L1
argument2 ← fonctionnement
argument3 ← des
argument4 ← ordinateurs

```
$ ./testParam.sh cmi_opt/im cmi_is maths-info info
```

exemple 2 : **echo "L1 fonctionnement des ordinateurs"**
 commande argument1

commande ← echo
argument1 ← L1 fonctionnement des ordinateurs

```
$ ./testParam.sh "cmi_opt/im cmi_is maths-info" info
```

Shell Bash

Imbrication de commandes



Récupère le texte écrit sur le terminal par la commande `cmd` dans une chaîne de caractère

Ne pas confondre avec `$cmd` qui accède à la valeur de la variable `cmd`

Exemple :

```
$ date
```

```
Mer 23 jan 2019 14:48:45 CET
```

```
$ echo "Nous sommes le $(date)."
```

```
$ echo "Nous sommes le $date."
```

```
$
```

Shell Bash

Opérateurs et expressions arithmétiques :

► Opérateurs arithmétiques

- + : addition
- : soustraction
- * : multiplication
- / : division
- ** : puissance
- % : modulo

Expressions arithmétiques :



méthode utilisant la commande `let` pour effectuer des calculs

Exemple

```
a=1
$ let "b=$a + 1"
$ echo $b
```

Expressions et conditions de test (1/3)



Ecriture d'une **condition**

► Expression / opérateurs de tests pour les nombres

`n1 -eq n2` : vrai si `n1` est égal à `n2`
`n1 -ne n2` : vrai si `n1` est différent de `n2`
`n1 -gt n2` : vrai si `n1` supérieur strictement à `n2`
`n1 -ge n2` : vrai si `n1` supérieur ou égal à `n2`
`n1 -lt n2` : vrai si `n1` inférieur strictement à `n2`
`n1 -le n2` : vrai si `n1` est inférieur ou égal à `n2`

► Expression / opérateurs de tests pour les chaînes de caractères

`chaîne1 = chaîne2` : vrai si `chaîne1` est égale à `chaîne2`
`chaîne1 != chaîne2` : vrai si `chaîne1` n'est pas égale à `chaîne2`
`-z chaîne` : vrai si `chaîne` est une chaîne vide
`-n chaîne` : vrai si `chaîne` n'est pas la chaîne vide

Shell Bash

Expressions et conditions de test (2/3)

► Expression / opérateurs de tests logiques

exp1 **-a** exp2 : ET logique (vrai si e1 et e2 sont vrais)

exp1 **-o** exp2 : OU logique (vrai si e1 ou e2 est vrai.)

! exp : NON logique (vrai si e est faux)

► Exemple :

```
#!/bin/bash
read -p "Si vous etes d'accord entrez o ou oui : " reponse
```

```
:
```

on pourra écrire des schémas alternatifs du style suivant :

```
si [ ! "$reponse" = "o" -a ! "$reponse" = "oui" ] alors
    echo "Non, je ne suis pas d'accord !"
sinon
    echo "Oui, je suis d'accord"
fsi
```

Expressions et conditions de test (3/3)

► Expression / opérateurs de tests pour les fichiers

-e fname	:	vrai si fname existe
-s fname	:	vrai si fname est vide
-d fname	:	vrai si fname est un répertoire
-f fname	:	vrai si fname est un fichier ordinaire
-r fname	:	vrai si fname est lisible (r)
-w fname	:	vrai si fname est modifiable (w)
-x fname	:	vrai si fname est exécutable (x)
fname1 -nt fname2	:	vrai si fname1 plus récent que fname2
fname1 -ot fname2	:	vrai si fname1 plus ancien que fname2

Shell Bash

Schéma alternatif (1/2) :

if / then :

Définition :

```
if condition ; then
    actions
fi
```

Exemple :

```
if [ $L -eq 1 ] ; then
    echo "licence 1"
fi
```

if / then / else :

Définition :

```
if condition ; then
    actions1
else
    actions2
fi
```

Exemple :

```
x=1
y=2
if [ $x -eq $y ] ; then
    echo "valeurs égales"
else
    echo "valeurs différentes"
fi
```

NB : espaces et ; sont indispensables !!

Shell Bash

Schéma alternatif (2/2) :

if / then / else-if / else :

Définition :

```
if    condition1  ; then
    actions1
elif  condition2  ; then
    actions2
elif  condition3  ; then
    actions3
else
    actions4
fi
```

Exemple :

```
x=1 ; y=2
if  [ $x -eq $y ] ; then
    echo "$x = $y"
elif [ $x -gt $y ] ; then
    echo "$x > $y"
else
    echo "$x < $y"
fi
```

case :

Définition :

```
case $variable in
    valeur1)  action(s)1  ;;
    valeur2)  action(s)2  ;;
    * )       action(s)N+1 ;;
esac
```

Exemple :

```
lang="fr"
case $lang in
    "fr")  echo "Bonjour"  ;;
    "nl")  echo "Goededag" ;;
    * )    echo "Hello"    ;;
esac
```

Shell Bash

Schémas itératifs :

for :

Définition :

```
for var in liste ; do
    actions
done
```

Exemple :

```
for var in 1 2 3 4 ; do
    echo $var
done
```

while :

Définition :

```
while condition ; do
    actions
done
```

Exemple :

```
x=10
while [ $x -ge 0 ] ; do
    read x
    echo $x
done
```

Exercice

Ecrire un script qui affiche le nombre d'arguments passés en paramètre et séparément chacun de ces arguments

Ressources complémentaires

Sites

- ▶ **GNU Bash manual**

<https://www.gnu.org/software/bash/manual/>

- ▶ **UNIX - Programmation en Bourne-shell**

[https://download.tuxfamily.org/linuxvillage/
Informatique/LeBourneShell/prog_sh.html](https://download.tuxfamily.org/linuxvillage/Informatique/LeBourneShell/prog_sh.html)

- ▶ **Guide avancé d'écriture des scripts Bash**

<https://abs.traduc.org/abs-fr/>

- ▶ **Vérifier son script**

<https://www.shellcheck.net/>

Suite...

Chapitre 4 : Fondements des aspects matériel de l'ordinateur