**Programmation Shell script**

## Introduction à la programmation shell

Un script bash est en général une suite de commandes Linux, exécutables dès l'invocation de son nom. Mais en plus, il s'agit d'un langage de programmation interprété et complet, comprenant notamment

* la gestion des entrées-sorties et de leur redirection
* des variables définies par le programmeur et des variables systèmes
* le passage de paramètres
* des structures conditionnelles et itératives
* des fonctions internes

#### Saisie du script

* utiliser vi de préférence à mc qui ne traite pas les accents
* Les lignes commençant par le caractère dièse # sont des commentaires. En insérer abondamment !
* Un script est un fichier de type texte regroupant des commandes shell et exécutable par l'interpréteur **(un shell)**, par exemple le programme /bin/bash.  
  Pour spécifier cela, la première ligne doit être comme suit:
  + #! /bin/sh -> on veut que sh interprète le script
  + #! /bin/ksh -> on veut que ksh interprète le script
  + #! /bin/csh -> on veut que csh interprète le script
  + #! /bin/bash -> on veut que bash interprète le script

#### Exécution du script

* Elle s'effectue par la commande **bash nom-script** ou **sh nom-script**
* La commande **$ ./bonjour**, lance l'exécution de façon générale, si le répertoire courant contenant bonjour n'est pas listé dans le PATH
* Pour que le script soit exécutable, il convient de lui accorder la permission x pour tous ses utilisateurs avec chmod :  
  **chmod a+x bonjour**
* Si les scripts sont systématiquement stockés dans un répertoire précis, par exemple */home/toto/bin*, on peut ajouter ce chemin dans le PATH personnel de toto.
* Pour cela, il suffit d'ajouter la ligne suivante dans */home/toto/.bash\_profile*, dont le rôle est d'affiner le profil personnel du shell de chaque utilisateur.   
  *# bash\_profile*   
  .....................   
  *#user specific environment and statup programs*   
  **PATH=$PATH:$HOME/bin**

## Entrées-sorties

Ce sont les voies de communication entre le programme bash et la console :

* **echo,** affiche son argument texte entre guillemets sur la sortie standard, c-à-d l'écran.  
  La validation d'une commande echo provoque un saut de ligne.  
  echo "Bonjour à tous !"
* **read**, permet l'affectation directe par lecture de la valeur, saisie sur l'entrée standard au clavier   
  read *var1 var2 ...* attend la saisie au clavier d'une liste de valeurs pour les affecter, après la validation globale, respectivement aux variables var1, var2 ..

**Exemple 1 :**Ecrire un script appelé script2 qui demande votre nom et prénom et affiche ‘’bonjour Mr votre\_nom votre\_prenom.

**Solution :  
  
#! /bin/bash  
  
echo "Tapez votre nom : "**

**read nom**

**echo "Tapez votre prénom : "  
read prenom   
  
echo "Bonjour Mr. $nom $prenom "**

#### Variables prédéfinies spéciales pour le passage de paramètres :

Elles sont gérées par le système et s'avèrent très utiles dans les scripts. Bien entendu, elles ne sont accessibles qu'en lecture.

Ces variables sont automatiquement affectées lors d'un appel de script suivi d'une liste de paramètres. Leurs valeurs sont récupérables dans $1, $2 ...$9

|  |  |
| --- | --- |
| **$?** | C'est la valeur de sortie de la dernière commande. Elle vaut 0 si la commande s'est déroulée sans problème. |
| **$0** | Cette variable contient le nom du script |
| **$1 à $9** | Les (éventuels) premiers arguments passés à l'appel du script |
| **$#** | Le nombre d'arguments passés au script |
| **$\*** | La liste des arguments à partir de $1 |
| **$$** | le n° PID du processus courant |
| **$!** | le n° PID du processus fils |

Exemple :  
$ cmd a 1 fich -2 t3 (cmd un script Shell quelconque)

$# => 5

$0 => cmd

$1 => a, $2 = 1, $3 = fich, $4 = -2, $5 = t3

$\* => a 1 fich -2 t3

**Exemple2 :**essayez ce script :  **#! /bin/bash  
  
echo "mon script est $0"**

**echo "il ya eu $# paramètres en entrée"  
echo "le premier paramètre est $1"**

**echo "le deuxième paramètre est $2"**

lancez le script avec des paramètres : **./exemple2 –s -a**

**Manipulation des variables**

Accès à la valeur : $NomVar

•remarque : Si ambiguïté utiliser {}

Exemples3 :

**var=’esprit’**

**echo $var**

**echo $varShell # variable varShell non définie**

**echo $var\_Shell # variable var\_Shell non définie**

**echo ${var}Shell # espritShell**

**echo $var-Shell # esprit-Shell**

**echo ${#var} # longueur de la valeur de var**

**Récupérer le résultat d’une commande**

Pour récupérer le résultat d’une commande et l’enregistrer dans une variable pour une ultérieure utilisation on utilise la syntaxe suivante :

**Nom\_variable=$(commande)**

Exemple1 :  **mois=$(date +%D)** #remarque : lancez **man date** pour voire tous les genres de formatage de la date (%D , %d, %F etc….)

**echo $moi**

**Exercice :**

1. Ecrire un script qui affiche la date et le répertoire courant de travail

2. Ecrire un script qui affiche la liste et le nombre de ces arguments

**Expressions arithmétiques**

**Exemple4 :**

**pour affecter une valeur à une variable (deux méthodes):**

**x=5**

**((y = (2 \* $x) +3))**

**echo $y**

**– Affecte à y la valeur de l'expression 2x+3**

**ou :**

**y=$(($x+1))**

**echo $y**

**les Opérateurs :**

(1) *a***++** *a*-- post-incrémentation, post-décrémentation

(2) **++**a **--**a pré-incrémentation, pré-décrémentation

(3) **-**a **+**a moins unaire, plus unaire

(4) **!** a négation logique

(5) a**\*\***b exponentiation : ab

(6) a**\***b a**/**b a**%**b multiplication, division entière, reste

(7) a**+**b a**-**b addition, soustraction

(8) a**<**b a**<=**b a**>**b a**>=**b comparaisons

(9) a**==**b a**!=**b égalité, différence

(10) a**&&**b ET logique

(11) a**||**b OU logique

(12) a**?**b**:**c opérateur conditionnel : Le fonctionnement de cet opérateur ternaire est le suivant : *expr\_arith1* est évaluée, si sa valeur est *vraie* la valeur de l'expression arithmétique est celle de *expr\_arith2*, sinon c'est celle de *expr\_arith3*

exemple : (( b = a < 3 ? 5 : 54 )).

(13) a**=**b a**\*=**b a%=b a**+=**b a-=b opérateurs d'affectation

(14) a**,**b opérateur virgule

## Chaines de caractères

***>Protection ou désactivation des caractèrse spéciaux***

Protection d'un caractère à l'aide du caractère **\**

Ce caractère protège le caractère qui suit immédiatement le caractère **\**.

Exemple :

**echo N\’noublie pas** => N’oublie pas

**echo N'oublie pas =>** erreur

Exemple 2 :

**x=5**

**echo $x** => 5

**echo \$x** => $x

Exemple 3 :

**echo ‘’abc’’** => abc

**echo \’’abc\’’**  => ‘’abc’’

Protection de caractères à l'aide d'une paire de guillemets **"***chaîne***"**

Tous les caractères de *chaîne* sauf **$ \ ` "** sont protégés de l'interprétation du shell. Cela signifie, par exemple, qu’à l’intérieur d’une paire de guillemets le caractère **$** sera quand même interprété comme une substitution.

Ex :

**echo "< \* $PWD \* >" =>** < \* /home/arafet \* >

**echo "< \* \"$PWD\" \* > "**  => < \* "/home/arafet" \* >

Protection totale **'***chaîne***'**

Entre une paire d’**apostrophes** (**′**), aucun caractère de *chaîne* (sauf le caractère **'**) n'est interprété.

Ex :

**echo '< \* $PWD \* >'**  => < \* $PWD \* >

***>Extraction***

Longueur d'une chaîne de caractères

Syntaxe : **${#*paramètre*}**

Cette syntaxe est remplacée par la longueur de la chaîne de caractères contenue dans ***paramètre***. Ce dernier peut être une variable, un paramètre spécial ou un paramètre de position.

Ex :

**echo $PWD**

/home/arafet

**echo ${#PWD}**

12 => longueur de la chaîne */home/arafet*

Extraction de sous-chaînes

**${paramètre:ind}** : extrait de la valeur de ***paramètre*** la sous - chaîne débutant à l'indice ***ind***. La valeur de ***paramètre*** n’est pas modifiée.

Attention : l'indice du premier caractère d'une chaîne est **0**

Ex :

ch="abcdefghijk"

**echo ${ch:3}** => *defghijk*

**${paramètre:ind:nb}** : extrait ***nb*** caractères à partir de l'indice **ind**

Ex :

**echo ${ch:8:2} =>** ij

***>Remplacement***

Remplacement de la première occurrence

Ex :

**v=totito**

**echo ${v/to/ma} =>** matito

La valeur de la variable *v* (***totito***) contient deux occurrences du modèle *to*. Seule la première occurrence est remplacée par la chaîne ***ma***.

Remplacement de la plus longue sous-chaîne

Ex :

**v=abcfefg**

**v1=${v/b\*f/toto}** => utilisation du caractère générique **\***

**echo $v1**

atotog

Deux sous-chaînes de *v* satisfont le modèle *b\*f* : *bcf* et *bcfef*

C’est la plus longue qui est remplacée par *toto*.

***>Suppression***

Suppression de la plus courte sous-chaîne à gauche

Syntaxe : **${*paramètre*#*modèle*}**

Ex :

x= **"12a34a"**

**echo ${x#\*a}**

34a => suppression de la sous-chaîne *12a*

Suppression de la plus longue sous-chaîne à gauche

Syntaxe : **${*paramètre*##*modèle*}**

Ex :

x= **12a34ab**

**echo ${x##\*a}**

b

Suppression de la plus courte sous-chaîne à droite

Syntaxe : **${*paramètre*%*modèle*}**

Ex:

**echo $PWD**

/home/arafet

**echo ${PWD%/\*}**

/home

**Exercice1** :

a) Ecrire un programme *calibre* prenant deux arguments, une chaîne de caractères *ch* et un nombre *nb*, qui affiche les *nb* premiers caractères de *ch*. Aucune erreur ne doit être traitée.

b) Modifier le programme précédent, soit *calibre1*, pour qu’il vérifie que :

- le nombre d’arguments est correct

- le deuxième argument est un nombre (suite non vide de chiffres).

**Exercice2 :**

En utilisant les modificateurs de chaînes,

a) écrire un programme shell *basenom* ayant un fonctionnement similaire à la commande unix **basename**. Cette commande affiche le dernier élément d’un chemin passé en argument. Il n'est pas nécessaire que ce chemin existe réellement.

Ex : $ **basename /toto/tata/tutu**

tutu

**Divers**

**la commande set**

set variables : initialisation d’une variable liste de tous les paramètres du système positionne les paramètres $i (set a b c : positionne $1 = a , $2 = b et $3 = c )

**La commande seq**

la commande **seq** permet de sortir une séquence de nombres, avec un intervalle éventuel

**seq [debut] [increment] fin**

Ex3

seq 0 2 10

0

2

4

6

8

10

Ex2

seq -2 3

-2

-1

0

1

2

3

Ex1

seq  4

1

2

3

4

**Le Paramètre spécial ?**

Le paramètre spécial **?** (à ne pas confondre avec le caractère générique **?**) contient le code de retour de la dernière commande exécutée de manière séquentielle (*exécution synchrone*).

Ex2

**ls -l vi**

ls: vi: Aucun fichier ou répertoire de ce type

**echo $?**

2   
--------------------------------

une erreur s’est produite !

Ex1

**pwd**

/home/sanchis

**echo $?**

0  
--------------------------------   
la commande *pwd* s'est exécutée correctement

code de retour = 0 si vrai

code de retour !0 si faux

Chaque commande positionne « à sa manière » les codes de retour différents de **0**. Ainsi, un code de retour égal à 1 positionné par la commande unix **ls** n'a pas la même signification qu'un code de retour à 1 positionné par la commande unix **grep**. Les valeurs et significations du code de retour d’une commande unix ou du shell sont documentées dans les pages correspondantes du manuel (ex : **man grep**).

Lorsque une commande est exécutée en arrière-plan (exécution asynchrone), son code de retour n'est pas mémorisé dans le paramètre spécial **?**.

**Structures de contrôle**

**Choix if**

La commande interne **if** implante le choix alternatif.

Syntaxe : **if** *suite\_de\_commandes1*

**then**

*suite\_de\_commandes2*

**elif** *suite\_de\_commandes*

**then** *suite\_de\_commandes*

**else** *suite\_de\_commandes*

**fi**

Le fonctionnement est le suivant : *suite\_de\_commandes1* est exécutée ; si son code de retour est égal à **0,** alors *suite\_de\_commandes2* est exécutée sinon c'est la branche **elif** ou la branche **else** qui est exécutée, si elle existe.

**Exemple :**

programme ***rm1***

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#!/bin/bash

if rm $1 2>/dev/null

then echo ‘’$1 a ete supprime’’

else echo ‘’$1 n\'a pas ete supprime ou n\’existe pas’’

fi

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**Remarque : Je n'arrive pas à mettre une condition dans le if**

Vérifiez bien qu'il y a des espaces autour des crochets.

Exemples :

* fichiers :
  + if [ -f fich ] # si le fichier fich existe et n'est pas un répertoire
  + if [ -d fich ] # si le fichier fich existe et c'est un répertoire
* chaînes :
  + if [ $word = "coucou" ]
  + if [ $var != "chaîne" ]
  + if [ $var ] # si la variable var n'est pas la chaîne vide
* nombres :
  + if [ $# -eq 3 ] # EQual. Si le nombre de paramètres est 3
  + if [ $1 -ne 1 ] # NotEqual. Si le premier paramètre est numériquement 1
  + if [ $chiffre -gt 4 ] # GreaterThan. Si $chiffre est numériquement supérieur à 4
  + if [ $chiffre -ge 5 ] # GreaterorEqual
  + if [ $chiffre -lt 5 ] # LessThan
  + if [ $chiffre -le 5 ] # LessorEqual

**Commande composée [[**

Syntaxe : **[[** *expr\_cond* **]]**

Le fonctionnement de cette commande interne est le suivant : l’expression conditionnelle *expr\_cond* est évaluée :

* + - * + si sa valeur est *Vrai*, alors le code de retour de la commande interne **[[** est égal à **0**.
        + Si sa valeur est *Faux*, le code de retour est égal à **1**.
        + Si *expr\_cond* est mal formée ou si les caractères **crochets** ne sont pas collés, une valeur différente est retournée.

Les deux caractères **crochets** doivent être collés et un caractère séparateur doit être présent de part et d’autre de *expr\_cond*. Les mots **[[** et **]]** sont des mots-clé.

La commande interne **[[** offre de nombreuses expressions conditionnelles ; c’est pourquoi, seules les principales formes de expr\_cond seront présentées, regroupées par catégories.

* Permissions :

**-r** fichier => vraie si entrée existe et est accessible en lecture par le processus courant.

**-w** fichier => vraie si entrée existe et est accessible en écriture par le processus courant.

**-x** fichier => vraie si le fichier entrée existe et est exécutable par le processus courant ou si le répertoire entrée existe et le processus courant possède la permission de passage.

**Exemple :**

*echo coucou > toto*

*chmod 200 toto*

*ls -l toto*

*--w------- 1 sanchis sanchis 7 déc 17 17:21 toto*

**le script :**

**[[ -r toto ]]**

**echo $?**

* Types d'une entrée :

**-f**  *fichier =>* vraie si *fichier* existe et est un fichier ordinaire

**-d** *fichier =>* vraie si *fichier* existe et est un répertoire

**-L** *fichier =>* vraie si *fichier* existe et est un lien symbolique

**Exemple :** programme ***affic***

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#!/bin/bash

if [[ -f $1 ]]

then echo $1 : fichier ordinaire

cat $1

elif [[ -d $1 ]]

then echo $1 : repertoire

ls $1

else echo $1 : type non traite

fi

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

* Renseignements divers sur une entrée :

**-a** *entrée* vraie si *entrée* existe

**-s** *entrée* vraie si *entrée* existe et sa taille est différente de zéro

*entrée1* **-nt** *entrée2* vraie si *entrée1* existe et sa date de modification est plus récente que celle de *entrée2*

*entrée1* **-ot** *entrée2* vraie si *entrée1* existe et est plus ancienne que celle de *entrée2*

*fich1* **-et** *fich2*  vrai si fich1 est un lien vers fich2

* Longueur d’une chaîne de caractères :

-z ch vraie si la longueur de la chaîne ch est égale à zéro

-n ch vraie si la longueur de la chaîne ch est différente de zéro

* Comparaisons de chaînes de caractères :

ch1 < ch2 vraie si ch1 précède ch2

ch1 > ch2 vraie si ch1 suit ch2

L’ordre des chaînes ch1 et ch2 est commandé par la valeur des paramètres régionaux.

ch == mod vraie si la chaîne ch correspond au modèle mod

ch != mod vraie si la chaîne ch ne correspond pas au modèle mod

**mod** est un modèle de chaînes pouvant contenir caractères et expressions génériques.

Si l'on souhaite désigner une classe de caractères tels que les minuscules ou les majuscules, il est préférable d'utiliser la syntaxe POSIX 1003.2 correspondante. Ce standard définit des classes de caractères sous la forme **[:***nom\_classe***:]**

Des exemples de classes de caractères définies dans ce standard sont :

**[:upper:]** (majuscules)

**[:lower:]** (minuscules)

**[:digit:]** (chiffres, 0 à 9)

**[:alnum:] (**caractères alphanumériques).

Exercice :

Écrire un script qui vérifie les droits d’un fichier donné en paramètre et

* + - * + affiche ‘’lecture et écriture’’ si le fichier si le fichier est accessible en mode rw
        + affiche ‘’lecture seul’’ si le fichier si le fichier est accessible en mode r-
        + affiche ‘’écriture’’ si le fichier si le fichier est accessible en mode –w

**Choix multiple case**

Syntaxe : case ... in ... esac

**case** valeur **in**

val11|val12|…|val1x**)**  liste\_commandes1**;;**

val21|val22|…|val1y**)**  liste\_commandes2**;;**

...

valn1|valn2|…|val1z**)** liste\_commandesn**;;**

**\*)** liste\_commandes **;;**

**esac**

• *liste\_commandesi* exécutée si *valeur* est de la forme de l'une des *valij*

* Le caractère \* couvre toute valeur

**Exemple :** programme shell ***oui*** affichant *OUI* si l’utilisateur a saisi le caractère *o* ou *O*

----------------------------------------------------------------

#!/bin/bash

read -p ″Entrez votre réponse : ″ **rep**

**case $rep in**

o**|**O **)** echo OUI **;;**

**\*)** echo Indefini ;;

**esac**

----------------------------------------------------------------

Exercice :

écrire un script shell « case1 »prenant un caractère en argument et qui affiche ce caractère si il est un chiffre, une lettre ou un caractère spécial.

**Itération for**

L'itération **for** possède plusieurs syntaxes dont les deux plus générales sont :

Deuxième forme :

**for** *var* **in** *liste\_mots*

**do**

suite\_de\_commandes

**done**

La variable *var* prend successivement la valeur de chaque mot de *liste\_mots*.

Première forme :

**for** *var*

**do**

suite\_de\_commandes

**done**

Lorsque cette syntaxe est utilisée, la variable *var* prend successivement la valeur de chaque paramètre de position initialisé.

***Exemple*** : programme ***for\_form1*** programme ***for\_form2***

- - - - - - - - - - - - - - - - -

#!/bin/bash

**for i**

**do**

echo **$i**

echo "Passage a l'argument suivant ..."

**done**

- - - - - - - - - - - - - - - - -

lancer : ./for\_arg 1 3 8 2

- - - - - - - - - - - - - - - - -

#!/bin/bash

i=0

**for** x **in** un deux trois quatre

**do**

**(( i++ ))**

echo **$i-$x**

**done**

- - - - - - - - - - - - - - - - -

**Exercice :** Ecrire un programme shell ***lsrep*** ne prenant aucun argument, qui demande à l'utilisateur de saisir un nom de répertoire et qui affiche des infos sur son contenu (nombre de fichiers, de répertoires et de liens).

**Itération while**

La commande interne **while** correspond à l’itération *tant que* présente dans de nombreux langages de programmation.

Syntaxe :

**while** *suite\_cmd1*

**do**

*suite\_cmd2*

**done**

- - - - - - - - - - - - - - - - -

#!/bin/bash

**cat ‘’**/etc/passwd’’ **| while read line**

**do**

**echo ‘’----------------------‘’**

echo **$line**

**done**

- - - - - - - - - - - - - - - - -

***Exemple*** : programme ***while1*** programme ***while1***

- - - - - - - - - - - - - - - - -

#!/bin/bash

**i=0**

**while (( i < 10 ))**

**do**

**(( i++ ))**

**echo "incrémentation $i"**

**done**

- - - - - - - - - - - - - - - - -