Introduction aux scripts shell

➢ Un script shell est un programme qui permet d'automatiser l'exécution d'une série d'opérations.

➢ Les scripts shell se présente sous la forme d'un fichier contenant une ou plusieurs instructions écrites en langage interprété (**bash**, **csh**, **ksh**, **php**, **perl**, …) qui sont exécutées Séquentiellement.

➢ Un langage interprété est un langage de programmation dont les codes sources sont interprétés, autrement dit, chaque ligne du code source est analysée et exécutée par l'interpréteur.

➢ Les langages interprétés sont portables, autrement dit, ils sont indépendants du système d'exploitation utilisé.

➢ **Autrement dit, le shell est :**

- ✓ l'interpréteur de commande ;
- ✓ l'interface entre l'utilisateur et les commandes ;
- ✓ un langage de programmation (interpréteur), il permet donc de réaliser de nouvelles commandes ;
- ✓ un gestionnaire de processus ;
- ✓ un environnement de travail configurable.

➢ **Le fichier qui contient un script, doit commencer par l'instruction :**

#! /(chemin absolu de l'interpréteur utilisé)

➢ Exemples :

- ✓ #!/bin/sh Toutes les instructions qui suivent, seront analysées par l'interpréteur sh
- ✓ #!/bin/bash Toutes les instructions qui suivent, seront analysées par l'interpréteur bash
- ✓ #!/bin/csh Toutes les instructions qui suivent, seront analysées par l'interpréteur csh
- ✓ #!/bin/tcsh Toutes les instructions qui suivent, seront analysées par l'interpréteur tcsh
- ✓ #!/usr/bin/perl Toutes les instructions qui suivent, seront analysées par l'interpréteur perl
- ✓ ……

➢ **Exécuter un script :**

✓ Un script doit être écrit dans un fichier texte, qui peut être créer soit en utilisant des éditeurs de textes graphiques (**emacs**, **gedit**, **textedit**, …) ou bien dans un terminal, (**vi**, …).

✓ Généralement, les fichiers sous **UNIX** sont crées avec un **mask=22** (par défaut), autrement dit ces fichiers ne sont pas exécutables, dans ce cas vous pouvez exécuter votre script, en tapant :

le_nom_de_l'interpréteur Nom_de_script

Si non, il faudra utiliser la fonction **chmod** pour ajouter le droit d'exécution :

chmod +x Nom_de_script

Dans ce cas, vous pouvez exécuter votre script en tapant :

./Nom_de_script ou bien /chemin_absolu/*Nom_de_script*

Vous pouvez aussi modifier la valeur de la variable **PATH** (**export PATH=$PATH:$HOME/Scripts**) afin de faciliter l'exécution.

➢ **Les variables d'un script Shell**

$? : Renvoie **1** si le nom du fichier du scripts Shell est connu, sinon **0**;

$0 : Renvoie le nom du fichier du scripts Shell (erreur si inconnu);

$*** : Renvoie la liste **$argv[*] des arguments;

$n ou **${n}** : Renvoie le **n**ème argument **$argv[n]** du scripts Shell;

$?var : Renvoie **1** si la variable var a une valeur, sinon **0** ;

$$: Renvoie le numéro de processus du Shell père ;

$# : Renvoie le nombre d'argumentsLes paramètres de réglage

➢ Les paramètres de réglage traduisent la position des composantes d'une saisie à l'intérieur d'une ligne de commande. Ils se composent du caractère signalant une variable **$** et un numéro d'ordre dans la saisie. Il peut y avoir au maximum 10 paramètres de réglage en commençant par **$0**

Prog.sh	**fichier1**	**fchier2**	**fichier3**
$0	**$1**	**$2**	**$3**

✓ Pour pouvoir traiter des commandes qui comportent plus de 10 paramètres de réglage. Les positions supérieures à 10 peuvent être demandées si la ligne de commandes est déplacée vers la gauche à l'aide de shift.

Sortie :

AA BB CC DD EE FF GG HH II KK LL MN NN
$0 $1 $2 $3 $4 $5 $6 $7 $8 $9

shift

Sortie :

BB CC DD EE FF GG HH II KK LL MM NN
$0 $1 $2 $3 $4 $5 $6 $7 $8 $9

Le langage interprété de Bourne Shell
SH (BASH, …)

Ce langage a été programmé pour la première fois en 1977 par Stephan Bourne (laboratoire d'AT&T Bell), il est devenue le shell par défaut pour les systèmes UNIX.

Les objets du langage:

I. Les variables

a. Déclaration et type de variables :
   i. Il n'est pas nécessaire de déclarer une variable avant de l'utiliser.
   ii. Les objets possédés par les variables sont d'un seul « **type** » : **chaîne de caractères**.

II. Affectation d'une valeur à une variable

a. Syntaxe : ***nom-de-variable = chaîne-de-caractères*** , (évitez les espaces).
   i. Création et modification en BASH :
      1.
         a. var1=5
         b. **var2=7**
      2. On peut déclarer des variables vides :
         a. var3=
         b. **var3=""**
      3. Affectation et interprétation des blancs : si la *chaîne-de-caractères* à affecter à la variable comporte des blancs, il faut en faire un seul mot à l'aide des quotes.

         var='En Normandie, il fait toujours beau !'

      4. Créer une variable en lecture seule :

         a. readonly variable=valeur

         b. Exemple : readonly MaVariable=7

      5. créer une variable locale (dans une fonction) :

         a. local variable=valeur

III. Variables dynamiques

a. Il est possible d'utiliser le contenu d'une variable pour en appeler une autre en remplaçant le caractère $ par un !
   1. Var="Test" ; Toto=Var
   echo $Toto ==> Var # Affiche le contenu de la variable **Toto**

      1. **echo ${Toto}** ==> bash: **echo${$Toto}:** bad substitution # Erreur

      2. **echo ${!Toto}** ==> Test # !Toto retourne le contenu de la variable **Var ==> echo $Var**
      (ou bien **${Var})**

IV. Afficher la longueur d'une variable ou le nombre d'éléments d'un tableau avec

i. Soit la variable : Var=abc.gm3@insa.GM

➡ La longueur de cette variable est : **echo ${#Var}** ==> 15 # Il y a 15 caractères

V. Retourner une valeur par défaut :

a. Si la variable n'existe pas "-" ou si la variable est vide ":-"
   i. ${Var-VarParDefaut}
      1. Exemple1 : VarParDefaut="Dimanche"
         a. echo ${Var-$VarParDefaut} ==> Dimanche
         b. echo ${Var:-$VarParDefaut} ==> Dimanche
      2. Exemple2: VarParDefaut="Dimanche" ; Var=""
         a. echo ${Var-$VarParDefaut} ==>
         b. echo ${Var:-$VarParDefaut} ==> Dimanche

VI. Opérations sur les contenus des variables # Ces opérations ne sont pas valides sur certaines version de bash

a. **Conversion minuscules <==> MAJUSCULES**
   i. **minuscules ==> MAJUSCULES**
      1. **Première lettre en MAJUSCULE ${Var ^}**
         a. **Chaine="un test"; echo ${Chaine^} ==> Un test**
      2. **La chaîne en MAJUSCULE ${Var^^}**
         a. **Chaine="un test"; echo ${Chaine^^} ==> UN TEST**
   ii. **MAJUSCULES ==> minuscules**
      1. **Première lettre en minuscule ${Var,}**
         a. **Chaine="UN TEST"; echo ${Chaine,} ==> uN TEST**
      2. **La chaîne en minuscule ${Var,,}**
         a. **Chaine="UN TEST"; echo ${Chaine,,} ==> un test**

---

b. **Substitutions :**
   i. **Remplacer la première occurence avec / ==> ${Var/Cherche_Motif/Remplace_Motif}**
      1. **Var="Mon premier script"**
         a. **echo ${Var/M/S} ==> Son premier script**
         b. **echo ${Var/Mon/Notre} ==> Notre premier script**
         c. **echo ${Var/premier/\?} ==> Mon ? script**
         d. **echo ${Var/p*m/?} ==> Mon ?ier script**
         e. **...**

---

   ii. **Remplacer toutes les occurences avec // ==> ${Var/Cherche_Motif/Remplace_Motif}**
      1. **Var="Mon premier script"**
         a. **echo ${Var//r/?} ==> Mon p?emie? sc?ipt**
         b. **echo ${Var//p/#} ==> Mon #remier scri#t**
      2. **Var="toto:x:309:50080:Monsieur Toto:/bin/bash**
         a. **echo ${Var//:/ } ==> toto x 309 50080 Monsieur Toto /bin/bash**

---

   iii. **Remplacer un motif du début ou fin de chaine "/#" ou "/%"**

      1. **Var="Mon premier script"**
         a. **echo ${Var/#Mon/Notre} ==> Notre premier script**
         b. **echo ${Var/#on/Notre} ==> # il n'y a pas de substitution, "on" n'est pas au début**
      2. **Var="Mon premier scriptSHELL"**
         a. **echo ${Var/%SHELL/ SHELL} ==> Notre premier script SHELL**
         b. **echo ${Var/%t/t SHELL} <==> echo ${Var/%SHELL/ SHELL}**

c. ...

c. **Conversion " binaire, octal, hexadécimale" vers la base décimale : B#Valeur**

i. **Conversion de binaire vers décimal**

1. **echo $((2#1)) ==> 1**
2. **echo $((2#10)) ==> 2**
3. **...**
4. **echo $((2#1010)) ==> 10**
5. **...**

ii. **Conversion de octal vers décimal**

1. **echo $((8#1)) ==> 1**
2. **...**
3. **echo $((8#7)) ==> 7**
4. **echo $((8#10)) ==> 8**
5. **...**
6. **echo $((8#12)) ==> 10**
7. **...**

iii. ....

Variables systèmes (extrait)	
$!	PID de la dernière commande lancée en arrière-plan
$$	PID du shell en cours (vérifiable avec la commande ps)
$_	Dernier paramètre de la dernière commande exécutée après substitution
!$	Dernière commande avant toute substitution
$BASH	Chemin d'accès de l'interpréteur bash
$BASH_VERSION	Version du bash (voir aussi le tableau $BASH_VERINFO)
$COLUMNS	Nombre de colonnes affichable par le terminal
$HOME	Dossier home de l'utilisateur (cd $HOME = cd ~)
$HOSTNAME	Nom de la machine
$HOSTTYPE	Informations sur la machine (ex x86_64) Voir aussi $MACHTYPE
$LINES	Nombre de lignes affichable par le terminal
$OLDPWD	Dossier précédent (cd $OLDPWD = cd ~-)
$PATH	Chemins de recherche des exécutables
$PIPESTATUS	Tableau contenant les codes de retour des commandes d'un pipe
$PPID	PID du processus père
$PWD	Dossier en cours (cd $PWD = cd ~+)
$RANDOM	Chiffre aléatoire entre 1 et 32237 (en bash seulement)
$SECONDS	Nombre de secondes écoulées depuis le lancement de l'interpréteur, donc du script
$SHELL	Nom du shell de login
$TMOUT	Nombre de secondes avant la déconnexion du script
$UID	UID de l'utilisateur en cours ($EUID donne l'identifiant réel)
$USER	Nom de l'utilisateur

VII. Les Tableaux

a) La création d'un tableau

- **En BASH, les tableaux commencent à l'index 0 !** (et à 1 en CSH)

✓ **Pour indiquer au bash la création ou la modification, il suffit d'utiliser des parenthèses !**

Exemple de création :

Tableau=(Un DeuxTrois Quatre)

✓ L'affichage se fait avec **${LeNomDuTableau[Index]}**

✓ Remplacer l'index par le caractère * ou @ pour afficher l'intégralité du tableau

☑ **Exemples d'utilisation :**

Montableau=(Janvier février Mars Avril Mai Juin Juillet Aout Septembre Octobre Novembre Décembre)

- Afficher le premier élément du tableau : **echo** $Montableau ==> Janvier
- Afficher le deuxième élément du tableau : **echo** ${Montableau[1]} ==> février
- Afficher l'intégralité du tableau : **echo** ${Montableau[@]} ==> Janvier février ... Décembre
- Ou bien **echo** ${Montableau[*]} ==> Janvier février ... Décembre
- Afficher la longueur du 1er élément du tableau : **echo** ${#Montableau} ==> 7
- Afficher la longueur du i ème élément du tableau : **echo** ${#Montableau[i]}
- Afficher le nombre d'éléments du tableau : **echo** ${#Montableau[*]} ==> 12
- Ajouter un nouveau élément au tableau : Montableau+=("2021")
- Afficher les indices utilisées du tableau : **echo** ${!Montableau[*]}

☑ **Suppression d'un élément du tableau** (pour supprimer tout le tableau :

- **unset Montableau** **# Effacer le contenu du tableau**
- **unset Montableau[0]** **# Effacer le premier élément du tableau**
- **unset Montableau[i]** **# Effacer le l'élément à l'indexe i+1 du tableau**

☑ **Ajout d'un élément en fin de tableau**

- **Tableau+=(Cinq); echo ${Tableau[4]} ==> Cinq # Affiche l'élément à l'index 4 du tableau (MonTableau contient bleu noir cyan), on utilise ici un calcul avec $((…))**
- **echo ${Tableau[*]:$((${#tableau[*]}-1))}** ==> # renvoie le dernier élément du tableau

☑ **Ajoute un élément a un indice spécifique (pour laisser des valeurs nulles par exemple)**

- **Tableau=(Un Deux Trois Quatre [9]=Tata)** Un, Deux, Trois, Quatre, puis à l'index 9 on trouve Tata
- Affiche les index utilisés du tableau remplis précédemment

echo ${!Tableau[@]} ==> 0 1 2 3 9

VIII. Affectation et substitutions

NomDuTableau=($(commande)) ou bien **NomDuTableau=`commande`**

- **Ou bien : Tab=($(cat NomDuFichier))** <==> **Tab=`cat NomDuFichier`**

a. On peut affecter à une variable le résultat d'exécution d'une commande :

i. **D=$(date)** ou bien **D=`date`**

1. **echo $D ==> Dim 31 jan 2021 17:45:41 CET**
2. **echo ${D[*]} ==> Dim 31 jan 2021 17:45:41 CET**

ii. **Soit le fichier SeR qui contient le texte "J'aime programmer des scripts SHELL"**

1. **Tab=($(cat SeR) ou bien Tab=`cat SeR`**
2. **echo ${Tab[*]} ==> J'aime programmer des scripts SHELL**

b. On peut aussi affecter à une variable le contenu d'une autre variable (substitution), sauf * :

i. U=$USER

ii. **F=*** On affecte à la variable **F** le caractère ***** et non pas la liste des noms des fichiers.

IX. Autres possibilités d'utilisation de l'opérateur $:

a. **Avec une variable :**

i. Var=azerty.qwerty@coucou.com

1. **echo ${#var} ==> 24** # Il y a 24 caractères

b. **Avec un tableau :**

i. **Tableau=(azerty . qwerty @ coucou . com)**

1. **echo ${#Tableau[*]} ==> 7** # Il y a 7 éléments

X. Découpage de chaines de caractères avant ou après un motif *m* avec #, ##, % et %%

a. **Elimine ce qui se trouve AVANT le premier motif**

i. Var=azerty.qwerty@coucou.com

1. **echo ${Var#*.}** ==> qwerty@coucou.com
2. **echo ${Var#*c}** ==> **oucou.com**

b. **Elimine ce qui se trouve APRES le dernier motif**

i. **Var=azerty.qwerty@coucou.com**

1. **echo ${Var%.*} ==> azerty.qwerty@coucou**
2. **echo ${Var#@*} ==> alerte.qwerty**

c. **Elimine ce qui se trouve AVANT le dernier motif**

i. **Var=azerty.qwerty@coucou**

1. **echo ${Var##*.} ==> qwerty@coucou**

2. **echo ${Var##*@} ==> coucou**

XI. Elimine ce qui se trouve APRES le premier motif m

i. **Var=azerty.qwerty@coucou**

1. **echo ${Var%.*} ==> azerty**

2. **echo ${Var%%@*} ==> azerty.qwerty**

XII. Récupérer une partie d'une chaine

i. **Var=azerty.qwerty@coucou**

1. **echo ${Var:0:5} ===> azert**

2. **echo ${Var:6:10} ==> qwerty@co**

3. **echo ${Var:(-4):3} ==> uco**

XIII. Découpage d'un tableau

i. **Tab**=(1 2 3 4 5 6)
   1. **echo ${Tab[*]:0:3}** ==> 1 2 3

XIV. Lire la saisie d'un utilisateur au clavier

- La fonction **read** permet de permet d'arrêter une procédure pour lire et y faire entrer une ou plusieurs saisies de l'entrée standard (normalement le clavier).

Syntaxe : read <option> <var1> <var2> ...

Options :

-n **NbCar** ==> Stoppe la saisie après NbCar caractères

Exemple : **read -n** 5 var

-p **Texte** = Affiche le texte avant la saisie

Exemple : **read -p** '*entrer 4 valeurs séparées par un espace* ' **-a** Tab

-t **NbSec** = Attends NbSec secondes avant d'annuler la saisie

Exemple : **read -t** 20 var

.....