Petit résumé de Script Shell Linux : Bash

Important : les explications peuvent être inexactes et mêmes fausses, je ne suis responsable de rien, si vous trouvez des fautes, ou si vous avez d'autres choses à ajouter veuillez me contacter s'il vous plait.

Définition :

Une interface système (shell en anglais) est une couche logicielle qui fournit l'interface utilisateur d'un système d'exploitation. Il correspond à la couche la plus externe de ce dernier. L'interface système est utilisée comme diminutif de l'interface utilisateur du système d'exploitation.

En-tête du script :
Un script doit débuter par un sha-bang : #!/bin/ « nom_du _shell » , ça permet d'indiquer quelle shell doit être utilisé pour exécuter le script , le shell choisi doit être déjà installé sur votre système.

Commandes générales :

- Afficher un message avec **echo** :
  - Ajouter l'option –e pour pouvoir faire des retours à la ligne avec \n
  - –n pour ne pas faire le retour à la ligne à la fin.
  - Ecrire un message dans un fichier :
    - En écrasant son contenu echo "message" > file_name
    - Sans écraser son contenu echo "message" >> file_name
- printf permet aussi d'afficher des messages.
- Définition et initialisation : nom=valeur(msg='texte')
- Notez que vous n'êtes pas obligés de définir une variable ni l'initialisé avant de l'utilisé
- Les types de quotes pour délimiter un string :
  - Simple quote ' ' : le contenu n'est pas analysé.
  - Double quote "" : le contenu est analyser , s'il contient des variables , ils sont remplacés par leurs valeurs
  - Back quote ` ` : le contenu est exécuté (msg=`pwd` , la varaible msg va contenir le résultat de la commande pwd)
- Lire le clavier avec **read** :
  - Read variable1 variable2… cette commande va lire ce que l'utilisateur entre au clavier et affectera chaque entrée de l'utilisateur (séparé par des espaces) à une variable, si les entrées sont plus que les variables, la dernière variable prendra toutes les valeurs restantes
  - Affichage d'un message de prompt : read –p 'message à afficher' variable (on ne peut pas faire un retour à la ligne avec \n)
  - Pour lire un nombre limité de caractères : read –n nombre_max_caracteres variable
- let "x = a op b" op = +,-,/,*,%
  - Les espaces ne sont pas obligatoires dans let
- on peut affecter une valeur à une variable en utilisant seulement l'operateur « = » sauf qu'il ne faut pas ajouter des espace entre l'operateur et les operands , et il faut que l'operand à droite de '=' soit une valeur et non pas une variable

La comparaison :

- La comparaison des deux variables numériques a et b : (les espaces sont obligatoires)
  - a -lt b (a lower then b : a<b)
  - a -gt b (a gower then b a>b)

- a -ge b (a higher or equal to b a>=b)
- a –le b (a<=b)
- a -ne b (not equal : a !=b)
- a -eq b (a=b)

- La comparaison de deux chaines de caractères a et b :
  - [a == b] (resp !=)
  - [a \< b] (resp >) , cette condition ne peut s'utiliser qu'entre [ ] et le caractère d'échappement est obligatoire
  - -z $chaine vérifie si une chaine est vide
  - –n $chaine vérifie si une chaine n'est pas vide
  - `[[ $string == a* ]] permet de tester si un string commence par a par exemple, attention les doubles crochets sont obligatoires.`
  - **Remarques importantes :**
    - Si on compare une chaine de caractère avec une autre qui est vide, le shell plante.
    - Il faut laisser un espace entre les opérandes (les variables/les valeurs) et les signes de comparaison
- On peut aussi utiliser la syntaxe suivante :
  - des comparaison : if ((i>0)) (resp <,==, !=)
  - Incrémentation/décrémentation : utiliser les opération unaire : ++ , -- : ((var++)) ou ((var--))

Testes sur fichiers :

Condition	Significationr
`-e $nomfichier`	Vérifie si le fichier existe.
`-d $nomfichier`	Vérifie si le fichier est un répertoire. N'oubliez pas que sous Linux, tout est considéré comme un fichier, même un répertoire !
`-f $nomfichier`	Vérifie si le fichier est un… fichier. Un vrai fichier cette fois, pas un dossier.
`-L $nomfichier`	Vérifie si le fichier est un lien symbolique (raccourci).
`-r $nomfichier`	Vérifie si le fichier est lisible (r).
`-w $nomfichier`	Vérifie si le fichier est modifiable (w).
`-x $nomfichier`	Vérifie si le fichier est exécutable (x).
`$fichier1-nt $fichier2`	Vérifie si fichier1 est plus récent que fichier2 (newerthan).
`$fichier1 -ot $fichier2`	Vérifie si fichier1 est plus vieux que fichier2 (olderthan)

Conditions :

- Instruction de choix (pareil que switch en java et en C) :
  - **case** VALEUR_A_TESTER(une valeur cte) ou **$**VARIABLE in
    - valeur_1) i0nstructions_1
    - ;;
    - valeur_2) instruction_2
    - ;;
    - …
    - *) (=défault) instructions par défaut.
    - ;;
  - esac (inverse de case pareil qu'une accolade fermante)
  - pour la commande case « Ou » est représenter par un seul '|' non ||
- **If** [teste] #attention aux espaces # ou bien if ((test))
  - then
  - Instruction
  - elif
  - then
  - instruction

    - else
    - **(pas de then ici)**
    - instruction
    - fi (inverse de if pareil qu'une accolade fermante)
- on peut combiner plusieurs condition entre (()) ex : ((cond1 && (cond2 || cond3)))

Boucles :

- **while** [condition] ; do // ou bien ((condition))
    - instructions
    - done

ou bien :

- **while** [condition] ou bien ((condition))
    - do
    - instructions
    - done
- **for** x in values # exemple : for x in 1 5 7 8 a b (pas de $ devant la variable car for attend une variable)
    - do
    - instructions
    - done
- **for** x in `commande`
    - do
    - instruction
    - done
- OU BIEN
- for ((i=valeur_initial ; i <= valeur_maximal ; le pas d'incrementation/decrementation))
    - do
    - instructions
    - done
- pour générer une séquence de nombre entre i et j on fait :
    - $(seq $i $j)

Combinaison de conditions :

- [cond1] && [cond2]
- [cond1 –a cond2] (-a = and et –o = or)
- ((cond && cond2)) # ne marche pas avec les strings (comparaison entre deux strings, -z string…)

Commandes pour les paramètres envoyés au script :

- $# = renvoie le nombre d'arguments
- $*= liste des paramètres (paramètre 0 est ignoré, c'est le nom du fichier) :
    - Exemple : bash valeur0(nom de script) valeur1 valeur2…
    - Retourne tous les paramètres à la fois séparés par des espaces
- $@=retourne les paramètres à partir du paramètre paramètre par paramètre
- La différence entre $@ et $* est remarqué quand on ajoute les doubles quotes $* renverra une seule chaine de caractère contenant tous les paramètres et $@ retournera plusieurs chaines chacune contenant un paramètre.
- shift nbre : décalage à gauche de «nbre» arguments (les arguments décalés seront perdus)
    - exemple : script.sh 1 2 3 4 5 6
    - dans le script.sh : shift 5
    - résultat : $1 = 6 (les autres arguments sont « perdus »)
- modifier les valeurs des paramètres du script :

- set val_1 val_2 val_3 … : donnera les valeurs val_1 val_2 val_3 à respectivement les arguments 1 2 3
  - si set est utiliser dans une fonction elle va affecter les valeurs données aux arguments de la fonction
  - si set est utiliser hors des fonctions elle va affecter les valeurs données aux arguments de script

Les fonctions :

- On déclare une fonction avec son nom précédé du mot clé « function »
- Les parenthèses après le nom de la fonction ne sont pas obligatoires
- le corps de la fonction est délimité par des accolades
- Pour envoyer des variables à une fonction on ajoute leurs noms après le nom de la fonction séparés par des espaces
- Les variables envoyées à la fonction sont reçues sur les paramètres $1 , $2 … (qui sont propres à la fonction , et non pas ceux envoyés au script.)
- Exemple :

```
function calcul
{
        let "m = $1 + $2 "
        echo $m
}
let "a = 5 "
let "b = 6"
# appelant la fonction :
calcul a b
```

- Quand on déclare une variable dans une fonction ou dans le main la variable est globale.
- Quand on déclare une variable avec « declare –i var_name=valeur » dans une fonction, var_name est locale à la fonction
- declare –F : affiche les nom de toutes les fonctions déclarer
- declare –f : affiche le code de toute les fonction déclarer
- declare –f functi1on_name : affiche le code de la fonction function_name
- unset –f function_name : rend une fonction indéfini (la supprimer)
- Exporter une fonction, afin qu'elle soit utilisable par tous les scripts appelés à partir du script actuel

- exemple :

```
#########Script 1#########
# !/bin/bash
function ma_fonction
{
#corps
}
export –f ma_fonction
bash scirpt2
```

```
#############Script 2###########
# !/bin/bash
ma_fonction
```

Attention si on execute bash scipt2 directement depuis la console, ça ne marchera pas !

- pour importer le contenu d'un script « s1 » depuis un autre script « s2 » on ajoute dans s2 :
  - source nom_fichier (ou bien chemin)
  - TOUT LE CONTENU SERA IMPORTÉ
- pour retourner une variable NUMERIQUE INFERIEUR A 256 on écrit à la fin de la fonction return nom_variable ou return valeur
- la valeur retournée peut être lu sur le parametre « $? »

Les tableaux :

- On déclare un tableau comme suit : declare -a nom_tableau
- Un tableau n'a pas de taille définit.
- Pour initialiser une case d'un tableau :
  - nom_tableau [index]=valeur(le premier index est 0)
- Pour utiliser «$» sur une case du tableau il faut ajouter les accolades comme suit: ${T[index]}
  - T[index] ou T[$index]
- ${#T[*]} renvoie le nombre d'élément d'un tableau
- ${T[*]} renvoie tous les éléments d'un tableau
- Pour stocker el résultat d'une commande dans un tableau on peut soit écrire :
  - tab=`commande` : tout ce que la commande renvoie sera stocker dans la case 0
  - tab=(`commande`) : chaque mot sera mis dans une case (les mots sont séparés par des espaces ou retour à la ligne)

Autre :

- pour exécuter un script depuis un programme C il faut utilisé la commande :
  - system ("commande arguments") ;
  - ou bien l'une des commandes execl ,execlp…
- pour exécuter un programme C depuis un script il suffit d'indiquer son chemin (absolu ou relatif) et son nom, exemple :
  - ./nom_exécutable (pour un fichier qui est dans le même dossier que votre script)
  - /home/username/nom exécutable : donner un chemin absolu

FAQ :

- **Quand ajouté le « $ » devant les variables et quand le pas faire ?**

- Le « $ » est ajouté devant les variables dans les chaines de caractères (dans echo par exemple) et dans les test (dans les if ,les cases et les boucles (while,for))
- **Dans les commandes qu'on a vu au cours** : **on n'ajoute pas** le « $ » que dans :
  - for variable in…. # for attend une variable , on peut pas lui donnée une valeur donc pas besoin d'ajouter $ devant variable
  - read variable # même chose !
  - let "variable1 = variable2 operand variable 3" # idem