Système d'exploitation
Scripts Bash
(7)

Opérations Arithmétiques
Tableaux

expr : calculs

- **expr : évalue** une expression arithmétique entière et **affiche** le résultat

  expr *var1* ***operateur*** *var2*

  - ***operateur*** peut prendre les valeurs : **+ , - , * , / , %**
    - l'opérateur * doit être protégé
  - ***var1*** *et* ***var2*** : arguments de commande *(espaces)*
    - expr 5 / 2 => 2 *# calcul entier*
    - expr 5/2 => 5/2 *# chaîne*
  - attention : pas d'affectation mais des comparaisons
    - expr a = 2 + 4 *# comparaison => booléen : 0/1*
    - a=$(expr 2 + 4) *# affectation => a=6 (= en dehors de expr)*
    - expr a=2+4 *# chaine => a=2+4*

expr : comparaisons

expr *var1* ***operateur*** *var2*

- ***operateur*** peut prendre les valeurs : **\< , \> , \>= , \<= , = (ou ==)**
  (nécessité de protéger)
  - *ne pas oublier le $, sinon comparaison ASCII (caractère x)* :
    - exemple avec x=3
      - expr x \> 4 => 1
      - expr $x \> 4 => 0
  - Composition de comparaisons avec : **&** et **|** simples
    - expr $x \> 2 \& $x \> 10 => 0
    - expr $x \> 2 \| $x \> 10 => 1

expr : comparaison de chaînes

- Comparaison d'une chaîne à une expression régulière
  affiche le nombre de caractères de la chaîne en correspondances

- **opérateur ":"** expr *chaine* **:** *expression*

  expr **"bonjour" : "b.*"** *=> affiche 7*

  expr **"bonjour" : "b.* .*"** *=> affiche 0*

  expr **"bonjour" : "b.*o"** *=> affiche 5*

  - syntaxe équivalente **"match"**
  - expr **match** *chaine expression*
    - expr **match "bon jour" "b.* .*"** *=> affiche 8*

expr : autres opérateurs chaînes

- **expr substr chaine pos long**

  affiche la sous-chaîne de chaine , débutant à la position pos (compté à partir de 1), de longueur long

  expr **substr** "bonjour" 4 4 => affiche jour

- **expr index chaine car**

  indice de la 1ere occurrence de car trouvée dans chaine,

  expr **index** "bonjour" o => affiche 2

- **expr length chaine**

  longueur dela chaine chaine

  expr **length** "bonjour" => affiche 7

let : expressions arithmétiques

- **let évalue** une expression arithmétique **sans l'afficher**
  - **let $[***expression mathématique***]**
  - Opérateurs : **+ , - , * , / , % , || , == , != , += , =** *(affectation)*
  - Espaces non significatifs, et pas de **$** pour les variables
    - ***$** uniquement pour les positionnelles : let $[c=$1+1]*
      - a=6
      - b=7
      - **let $[c=a* b]**
      - echo $c *=> affiche 42*
  - le **let** peut être omis, les crochets suffisent :
    - echo **$[**a*b**]**
    - c=**$[**a*b**]**

Syntaxe condensée : ((...)) et $((...))

- ((*expression mathématique*)) : **évaluation** comprenant une **affectation**
  - ((x=c++)) *# x vaut c, puis $c est incrémenté*
  - ((x=++c)) *# c est incrémenté, puis affecté à x*
  - ((i=i+$1)) *# ajouter le $ pour les positionnelles*

- $((*expression mathématique*)) : **évaluation** retournant un **résultat**
  - *pour un affichage ou une affectation*
  - calcul en ligne de commande
    - echo $((10%3)) *# affiche 1, i.e. 10 modulo 3*
    - x=$((c++))

((...)) et $((...))

- Syntaxe utilisable pour toute commande *(valeur du compte-rendu)*
  - if **((**x>5**))** # condition
  - echo "Nb fichiers **$((**$(ls -l | wc -l) -1**))**"
  - echo $((RANDOM%49+1))
    - *affiche une valeur comprise entre 1 et 50*
- Attention : if **$((**x>5**))** => incorrect : résultat interprété comme une commande

Forcer le type

- Les **commandes declare** et **typeset** fixent le type d'une variable
  - portée de la variable réduite au bloc

- declare **–i** x *# $x est traité comme un entier*
  - *en cas d'affectation d'une chaîne : assigne 0 (pas d'erreur!)*
  - *variables utilisables sans le $*

```
prix=1000
marge=100
ttc=prix+marge
echo $ttc
```

➡ prix+marge

chaîne

```
declare -i prix marge ttc
prix=1000
marge=100
ttc=prix+marge
echo $ttc
```

➡ 1100

- declare –i x=2 y z=1 *# déclaration multiple, initialisation*
- declare –i**r** x=-4 *# constante entière, affectation unique*
- declare **-a** s *# déclaration d'une chaîne de caractères*

Boucle arithmétique

- Boucle contrôlée par l'évaluation d'expressions arithmétiques

```
for ((expr1;expr2;expr3))
do
   ...
done
```

- Exemple : *décompte (affiche 5 4 3 2 1)*

```
for ((i=5;i>0;i--))
do
   echo -n " $i "
done
```

Tableaux : déclarations

- Uniquement des tableaux à **une** dimension (*index depuis 0)*

- Déclaration implicite lors d'une affectation, ou de l'initialisation :
  - tab**[**0**]**="un"
  - tab=**(**"un" "deux" "trois"**)**

- Déclaration explicite :
  - tableau de 10 chaînes : **declare –a** tab[10]
  - tableau d'entiers *avec initialisation* : **declare –i** tab=(2 5 -2)
  - tableau en lecture seule : **declare** –ar tab[10]

- Destruction
  - **unset** $tab # d'un tableau
  - **unset** $tab[i] # d'un élément

Tableaux : Accès

- Accès à un élément du tableau : **${tableau[indice]}**
  - Attention :

```
t[0]="zero"
echo ${t[0]}    => affiche zero
echo $t         => affiche zero  (1er élément)
echo $t[0]      => affiche zero[0]  !!!
```

- Exemple : affichage d'un tableau

```
declare -i entiers=(-5 50 12)
for ((x=0;x<3;x++))
do
   echo ${entiers[x]}
done
```

Tableaux : initialisation

- Initialisation
  - tableau=([indice]="chaîne" [indice]="chaîne"...[indice]="chaîne")
  - l'indice est facultatif, l'espace est le séparateur
    - tableau=([4]="quatre" "cinq" "six") *=> tableau[5]="cinq"...*
  - **read –a** *tableau*
    - renseigne *tableau* avec les chaînes saisies sur l'entrée standard
      - **sur une seule ligne** avec l'espace comme séparateur
      - chacun des mots est affecté successivement à une cellule du tableau
  - **read –a** *tableau* ***<fichier***
    - renseigne *tableau* avec les chaînes contenues dans le fichier
      - chaînes **sur une seule ligne** avec l'espace comme séparateur

Tableaux : Accès au contenu

- **${tableau[*]}** : liste des valeurs du tableau
- **${!tableau[*]}** : liste des indices *(pour lesquels une valeur a été affectée)*
- **${#tableau[*]}** : nombre d'éléments contenus *(pas sa taille)*

- Affichage complet des valeurs : **echo ${tableau[*]}**
- Ajout en fin *(sous réserve)* : **tableau[${#tableau[*]}]=valeur**
- Ajout en tête : **tableau=(valeur ${tableau[*]})**

crée un nouveau tableau

Tableaux : Parcours complet

tableau=("un" "deux" [4]="quatre")

3 écritures :

```
cpt=0
for i in ${tableau[*]}          # les différentes valeurs
do echo "$[++cpt] => $i
done
```

1 => un
2 => deux
3 => quatre

```
for i in ${!tableau[*]}         # les différents indices
do echo "$i => ${tableau[i]}" # le $ n'est pas repris
done
```

0 => un
1 => deux
4 => quatre

```
for ((i=0;i<${#tableau[*]};i++)) # uniquement si éléments contigus
do echo "$i => ${tableau[i]}"
done
```

0 => un
1 => deux
2 =>