Séance No 4

- Commande read
- regroupements de commandes
- expressions arithmétiques
- mise au point des scripts shells
- Projet No 1

commande READ

- syntaxe: read var1 var2 ...
- lit son entrée standard et affecte le premier mot lu à var1, le deuxième à var2, ... le texte restant est affecté à la dernire variable
- Exemple:

```
echo hello world et salut les tepos
  read a b c
```

  - a contient hello
  - b contient world
  - c contient « et salut les tepos »

commande READ: utilisation

- 2 utilisations typiques:
  - pour mettre dans une variable la saisie d'un utilisateur
  - en liaison avec while :

```
while read a b
do
   traitement de a et b
done
```

  - la boucle s'arrête en fin de fichier (caractère Ctrl-D si sur une saisie utilisateur)

READ : exemple

- on suppose que le fichier nombres.txt contient deux entiers par ligne

```
$ cat test-read.sh
#! /bin/bash
while read a b
do
  echo $a + $b = $(( a+b ))
done
$ ./test-read.sh < nombres.txt
1 + 3 = 4
6 + 4 = 10
2 + -5 = -3
```

regroupement de commandes

- () et {}
  - pour rediriger les sorties des commandes vers une même fichier
  - pour exécuter des commandes dans un même contexte
- (): exécution dans un shell enfant
  - cd /;(cd /tmp;ls);pwd
  - (pwd;w)> /tmp/test1.txt
- {}: exécution par le shell courant
  - il faut un espace avant et après chaque accolade et un ; après la dernier commande
  - cd /; { cd /tmp;ls; } ;pwd
  - { pwd; w } > /tmp/test2.txt

expressions arithmétiques

- mécanisme historique: via la commande expr
  - => pas au programme de l'examen
- mécanisme moderne (SUS V3, POSIX): (( )) et $(( ))
  - => au programme de l'examen

expr:

- l'expression arithmétique doit être passée sous la forme d'arguments séparés par des espaces (représentés ici par _)
  - 1_+_2 : 3
  - 1+2: chaîne 1+2
- les caractères qui peuvent être interprétés par le shell doivent être protégés:
  - * (et pas * seul)
  - \> (et pas > seul)
  - ...

expr: exemples

```
$ expr 2 + 5
7
$ expr 2 * 3
syntax error
$ expr 2 \* 3
6
$ x=1
$ y=$( expr $x + 3 )
$ echo $y
4
```

expr: quelques opérateurs

opérateur	sens
n1 + n2	
n1 - n2	
n1 * n2	
n1 / n2	
n1 % n2	modulo
\>, \<, \>=, \<=, =, !=	opérateurs de comparaison
chaine : expression_rationnelle	comparaison
\(expression \)	regroupement
s1 \& s2	vrai si s1 et s2 sont non nulles (valeur 0 et chaîne nulle)
s1 | s2	vrai si s1 ou s2 est non nulle

commande (( ))

- syntaxe: ((expression)) ou let expression
- avantages par rapport à expr:
  - plus grand choix d'opérateurs (ceux du C)
  - les caractères spéciaux du shell peuvent ne pas être protégés
  - les noms de variables peuvent ne pas être préfixé par $
  - les affectations se font directement dans la commande

(( )): quelques opérateurs

opérateur	sens
n1 + n2	
n1 - n2	
n1 * n2	
n1 / n2	
n1 % n2	modulo
>, <, >=, <=, ==, !=	opérateurs de comparaison
(expression)	regroupement
s1 && s2	ET logique
! e1	négation de e1
s1 || s2	OU logique
~, >>, <<, &, |, ^	travaillant sur les bits
+=, -=, *=, ...	n+=e équivaut à n=n+e

(( )) : exemples

```
$ x=1
$ ((y=x+1))
$ echo $y
2
$ z=7
$ z*=y
$ echo $z
14
$ if (( (x==1) && (y>10) ))
  then
```

substitution d'expressions arithmétiques

- syntaxe: $((expression arithmétique))
- Exemple:

```
$ echo 1 + 2 vaut $(( 1+2))
1 + 2 vaut 3
$ x=1
$ echo x vaut maintenant $(( x=x+1))
x vaut maintenant 2
$  echo essai ((1+2))
bash: syntax error near unexpected
  token `(('
$ echo essai $( ((1+2)) )
essai
```

mise au point des scripts

- La mise au point est un gros point faible des shells
  - pas de débugger
  - portée dynamique des variables
  - pas de compilation: les erreurs de syntaxe sont détectées quand le flot d'exécution les atteint
- les outils utilisés pour la mise au point sont minimaux et archaïques
  - des options du shell permettant un affichage verbeux lors de l'exécution
  - afficher la valeur de variables à l'aide de commandes echo ou printf
  - mettre des points d'arrêt en lançant des sous-shell

option du shell: -e

- set -e: interrompt le shell à la première commande dont le code de retour est non nul
- Exemple: que se passe-t-il si le dossier n'existe pas ?

```
#! /bin/bash
cd /var/spool/aSupprimer
rm -rf *
```

- c'est une bonne pratique d'intégrer un 'set -e' à tous ses scripts.

option du shell: -u

- set -u: l'utilisation de la valeur d'une variable non définie est considérée comme une erreur.
- L'utilisation de ce paramètre supprime de nombreuses erreurs dues à des fautes de frappe
- Exemple:

```
$ echo $aaa
-bash: a: unbound variable
$ aaa=12
$ echo $aaa
12
```

option du shell: -x

- set -x: affiche les commandes telles qu'elles sont exécutées (après analyse de la ligne de commande et substitutions)
- Exemple:

```
$cat ex2.sh
#! /bin/bash
set -x
x=1
echo $x $(( 1+3)) t{a,b} *
$ ls
F3  ex2.sh  f1  f2
petit@ns:/tmp/Test$ ./ex2.sh
+ x=1
+ echo 1 4 ta tb F3 ex2.sh f1 f2
1 4 ta tb F3 ex2.sh f1 f2
```

option du shell: -v

- set -v : affichage des commandes avant substitution
- on reprend l'exemple précédente en remplaçant « set -x » par « set -v »:

```
$ ./ex2.sh
x=1
echo $x $(( 1+3)) t{a,b} *
1 4 ta tb F3 ex2.sh f1 f2
```

option du shell: -n

- set -n: lecture des commandes sans les exécuter (pour détecter des erreur de syntaxe)
- Exemple:

```
$ cat ./ex2.sh
#! /bin/bash
set -n
x=1
echo coucou
echo $x $(( 1+3) t{a,b} *
$ ./ex2.sh
./ex2.sh: line 6: unexpected EOF while looking
  for matching `)'
./ex2.sh: line 7: syntax error: unexpected end
  of file
```

Bilan

- A la fin de cette cinquième séance, vous devez savoir utiliser:
  - la commande read
  - les expressions arithmétiques entières avec (( )) et $(( ))
  - les options et techniques de mise au point (-e, -u et -x)