Les scripts shell

Structures de contrôle — Sélection **if-then-else-fi**

- Pour exécuter des instructions sous certaines conditions on peut utiliser la structure `if-then-else-fi`
- Syntaxe :
  - ```
    if condition
    then
        commande_1
    else
        commande_2
    fi
    ```
- La `condition` peut prendre trois formes. Elle peut être
  - une commande qui est exécutée :
    - `if grep 'or'`
  - la commande `test` ou les crochets simples `[ ]` avec une expression :
    - `if [ "$a" = "yes" ]`
  - les crochets doubles `[[ ]]` avec une expression :
    - `if [[ "$a" = "yes" ]]`

Les scripts shell

Structures de contrôle — Sélection **if-then-else-fi**

- 1ère forme : La condition est une commande
  - Si le code de retour de la commande est 0, la condition est considérée vraie, sinon elle est considérée fausse.
  - Exemple :
    - ```
      if grep 'or' to_be.txt
      then
          echo "Found the string \"or\" in the file to_be.txt."
      fi
      ```

Cette branche est executée si `grep` retourne 0.

  - L'opérateur `!` permet d'inverser la condition.
    - ```
      if ! grep 'or' to_be.txt
      then
          echo "Could not find the string \"or\" in the file to_be.txt."
      fi
      ```
      ```

Les scripts shell

Structures de contrôle — Sélection **if-then-else-fi**

- 2ème forme : La condition utilise la commande `test`
  - Cette commande spéciale permet de de faire des comparaisons de chaînes de caractère et valeur entières, tester la présence et certaines caractéristiques des fichiers, ...
  - Elle prend une expression dans ses arguments et retourne la valeur 0 si l'expression est vraie, 1 si elle est fausse.
  - Exemple :

```
if test -f to_be.txt
then
    echo "File to_be.txt is an ordinary file."
fi
```

Cette branche est executée si le fichier `to_be` existe et est un fichier ordinaire.

- Souvent on remplace la commande `test` par son synonyme `[ ]` qui est plus lisible

```
if [ -f to_be.txt ]
then
    echo "File to_be.txt is an ordinary file."
fi
```

  - Il faut mettre un espace blanc des deux cotés de chaque crochet

```
if [ -f to_be.txt ]
```


Les scripts shell

Structures de contrôle — La commande **test** — Tests sur des fichiers

- Les options ci-après permettent de tester l'existence et certaines propriétés d'un fichier
  - Rappel : un *fichier* peut être un fichier ordinaire ou un répertoire ou une socket ou un tube nommé etc.
  - Sont mentionnés ici les tests les plus courants, la liste complète se trouve dans le Bash Reference Manual http://www.gnu.org/software/bash/manual/bashref.html#Bash-Conditional-Expressions

Option	**Vraie si**
-d *fichier*	Le fichier existe et est un répertoire
-e *fichier*	Le fichier existe
-f *fichier*	Le fichier existe et est un fichier ordinaire
-r *fichier*	Le fichier existe et est lisible
-s *fichier*	Le fichier existe et a une taille plus grande que zéro
-w *fichier*	Le fichier existe et on peut écrire dedans
-x *fichier*	Le fichier existe et est exécutable

Les scripts shell

Structures de contrôle — La commande **test** — Tests sur des chaînes de caractères

- Les options ci-après permettent de tester et comparer des chaînes de caractères

Option	Vraie si
chaîne_1 = *chaîne_2*	Les deux chaînes sont identiques
chaîne_1 != *chaîne_2*	Les deux chaînes sont différentes
chaîne_1 < *chaîne_2*	Dans un tri lexicographique la première chaîne apparaît avant la seconde
chaîne_1 > *chaîne_2*	Dans un tri lexicographique la première chaîne apparaît après la seconde
-n *chaîne*	La chaîne est non-nulle
-z *chaîne*	La chaîne est nulle

Les scripts shell

Structures de contrôle — La commande **test** — Tests sur des valeurs entières

- Les options ci-après permettent de comparer des variables ou expressions arithmétiques entières.

Option	Vraie si
valeur_1 -eq *valeur_2*	(**eq**ual) valeur_1 == valeur_2
valeur_1 -ne *valeur_2*	(**n**ot **e**qual) valeur_1 != valeur_2
valeur_1 -ge *valeur_2*	(**g**reater or **e**qual) valeur_1 >= valeur_2
valeur_1 -gt *valeur_2*	(**g**reater **t**han) valeur_1 > valeur_2
valeur_1 -le *valeur_2*	(**l**ess or **e**qual) valeur_1 <= valeur_2
valeur_1 -lt *valeur_2*	(**l**ess **t**han) valeur_1 < valeur_2

Les scripts shell

Structures de contrôle — La commande **test** — Opérateurs booléens

- Les options ci-après permettent de combiner des conditions avec des opérateurs booléens.
- Pour regrouper des expressions on peut utiliser des parenthèses.

Option	**Signification**
! *condition*	Négation
condition_1 -a *condition_2*	(**a**nd) ET logique
condition_1 -o *condition_2*	(**o**r) OU logique
(...)	Regroupement

Les scripts shell

Structures de contrôle — La commande **test** — Opérateurs booléens

- La commande test demande beaucoup de précautions :
  - Il faut protéger les variables avec des guillemets
  - Il faut protéger les opérateurs > et <
  - Il faut protéger les parenthèses ( )

- Exemple : Surveiller la capacité restante d'une partition de disque `$part`. Afficher un message s'il s'agit de la partition `/home` ou `/var` et si la capacité est insuffisante.

```
# Déterminer la capacité restante de la partition $part avec la commande df
capacity=$(df "$part" | tail -n 1 | tr -s ' ' | cut -d ' ' -f 6)
# Maintenant on a un pourcentage entre 0% 100%. Enlever le signe pourcent %
capacity=${capacity%\%}
if [ \( "$part" = "/home" -o "$part" = "/var" \) -a "$capacity" -le 20 ]
then
    echo "Warning: $part has less than 20% capacity"
fi
```

Les scripts shell

Structures de contrôle — Sélection **if-then-else-fi**

- L'expérience a montrée que les développeurs font beaucoup d'erreurs en utilisant la commande `test` / les crochets simples `[ ]`. C'est pourquoi le shell Bash a introduit pour la condition une ...
- 3ème forme : La condition utilise les doubles crochets `[[ ]]`
  - Écriture plus simple et sûre des conditions
    - Pas nécessaire de protéger les opérateurs > et <
    - Pas nécessaire de protéger les parenthèses ( )
    - Les opérateurs booléens s'écrivent plus naturellement && et || au lieu de `-a` et `-o`
  - Malheureusement ils sont seulement disponibles avec le shell Bash.
  - Exemple :
    - ```
      if [[ ( "$part" = "/home" || "$part" = "/var" ) && "$capacity" -le 20 ]]
      then
          echo "Warning: $part has less than 20% capacity"
      fi
      ```
- Recommandation : Utiliser les double crochets [[ ]] pour minimiser les erreurs.

Les scripts shell

Structures de contrôle — La commande **test**

- Pour utiliser une structure `if-then-else-fi` sur la ligne de commande il faut la mettre sur une seule ligne et pour cela on utilise des points-virgule
  - avant les mot-clés `then`, `elif`, `else` et `fi`
  - entre les commandes des branches

```
$ if [ -f to_be.txt ] ; then echo "File to_be.txt is an ordinary file."; fi
File to_be.txt is an ordinary file.
$
```
```

Les scripts shell

Structures de contrôle — Sélection **if-then-else-fi**

- Pour les structures `if` imbriquées il existe une syntaxe spéciale avec le mot-clé `elif`
- Structure imbriquée

```
if condition_1
then
    commande_1
else
    if condition_2
    then
        commande_2
    else
        if condition_3
        then
            commande_3
        else
            commande_n
        fi
    fi
fi
```

- Structure équivalente utilisant `elif` :

```
if condition_1
then
    commande_1
elif condition_2
then
    commande_2
elif condition_3
then
    commande_3
else
    commande_n
fi
```


Exercice 03.06

- Écrivez un script qui utilise les options -d -e -f -r -s -w -x pour décrire les fichiers qui lui sont passés en argument.

Les scripts shell

Structures de contrôle — Boucle **while-do-done**

- La boucle `while-do-done` sert à répéter un traitement tant qu'une condition est vraie.
  - Syntaxe :
    -
```
while condition
do
    commandes
done
```
- La condition a la même forme et la même signification que dans la structure `if-then-else-fi`
- Exemple d'une boucle qui compte de 0 à 9 :
  -
```
typeset -i i=0
while [[ $i -lt 10 ]]
do
    echo -n $i
    i=$i+1
done
echo
```

```
0123456789
```

- Quelques shells (pas tcsh) connaissent aussi la boucle `until-do-done`
  - La signification de la condition est inversée : la boucle continue tant que la condition est *fausse*

Les scripts shell

Structures de contrôle — Rupture de séquence avec **break** et **continue**

- On peut modifier le déroulement des boucles avec les mots-clé `break` et `continue`
  - `break` : sortir de la boucle en cours
  - `continue` : renvoyer le contrôle au début de la boucle
- Exemple :
  -
```
for index in 1 2 3 4 5 6 7 8 9 10
do
    if [[ "$index" -le 3 ]]
    then
        echo "continue"
        continue
    fi

    echo "$index"

    if [[ "$index" -ge 8 ]]
    then
        echo "break"
        break
    fi
done
```

```
continue
continue
continue
4
5
6
7
8
break
```

Les scripts shell
Paramètres positionnels

- Lorsqu'on appelle un script et on lui passe des paramètres sur la ligne de commande, le script peut accéder à ces paramètres, dits *paramètres positionnels*.
  - Le script utilise la syntaxe `$1`, `$2`, `$3`, etc.
  - Le premier argument est accessible en lisant `$1`, le deuxième dans `$2`, et ainsi de suite.
    - Attention, le dixième argument s'écrit `${10}`
  - Par convention l'argument numéro zéro `$0` contient toujours le nom du script tel qu'il a été invoqué.
- Exemple : script `show_arguments`

```
#!/bin/bash
echo "\$0: $0"
if [[ -n "$1" ]] ; then echo "\$1: $1" ; fi
if [[ -n "$2" ]] ; then echo "\$2: $2" ; fi
if [[ -n "$3" ]] ; then echo "\$3: $3" ; fi
```

```
$ ./show_arguments aaa bbb
$0: ./show_arguments
$1: aaa
$2: bbb
$
```

- Le paramètre spécial `$#` contient le nombre d'arguments, sans compter le paramètre $0.
  - Dans l'exemple précédent `$#` vaut 2

Les scripts shell
Paramètres positionnels

- La commande `shift` est utile pour examiner les paramètres positionnels un à un. Un appel de `shift` décale l'ensemble des paramètres à gauche :
  - `$1` est remplacé par `$2`, celui-ci par `$3`, et ainsi de suite.
  - Le paramètre `$0` n'est pas concerné
  - Le paramètre spécial $# est décrémenté de 1.
- Exemple : script `show_arguments2`

```
#!/bin/bash
while [ $# -ne 0 ]
do
    echo $1
    shift
done
```

```
$ ./show_arguments2 aaa bbb "" ddd
aaa
bbb

ddd
$
```

Exercice 03.07

- Créez un script qui prenne en argument une chaîne de caractères représentant une question. Votre script devra afficher cette chaîne et attendre une réponse de l'utilisateur
  - Si l'utilisateur saisit Y la fonction renvoie Vrai ; s'il saisit N, elle renvoie Faux ; dans tous les autres cas elle reboucle sur la question