# Les structures de contrôle en shell

Comme tout principe de programmation, des structures de contrôle sont représentées et permettent d'exécuter des scripts plus complexes.

## Quelques concepts syntaxiques

Le script shell **n'est pas** un langage C-like. Il existe quelques règles syntaxiques qui s'appliquent de façon générale :

### La syntaxe de blocs

- Le shell comme tout autre langage connaît un principe de blocs.
- La notion de bloc n'est pas liée au langage, elle est liée
- La notion de bloc en shell est marquée par des mots-clefs et non pas par des marqueurs explicites '{' '}'.
- Vous avez déjà remarqué que les instructions ne se terminent pas par un ";". La fin de la ligne marque la fin de la commande, comme si vous la tapiez dans une console.

### Les expressions évaluables

Il existe plusieurs façons d'évaluer des expressions en bash. Par défaut, la syntaxe implicite est "text orientée". Il faut donc des marqueurs spéciaux pour indiquer que l'on veut une évaluation booléene ou encore numérique.

#### L'évaluation booléene : [ ... expression ... ]

Evalue avec des opérateurs booléens de test l'expression.

Exemples :

```
f=monfichier.txt
if [ -x $f ]
then
    ...
fi

...

while (( $x > 0 ))
do
     ...
done
```

#### L'évaluation arithmétique : (( ... expression arithmétique ... ))

Evalue arithmétiquement l'expression, en interprétant correctement les opérateurs.

Exemples :

```
x=1
if (( x == 1 ))
```

```
then
    ...
fi

...

while (( $x > 0 ))
do
     ...
done
```

## La structure conditionnelle

La structure conditionnelle ressemble en bash à toutes les autres syntaxes conditionnelles :

<mot-clef> <condition> <bloc conditionnel>

### Structure conditionnelle simple

La fin de ligne étant un marqueur de fin d'instruction pour bash le début de bloc est marqué par le mot-clef "then".

```
if [ condition ]
then
   commande1
   commande2
   ...
fi
```

Le marquage de fin e bloc reste cependant nécessaire. Le shell choisit un mot clef inverse du mot-clef de début de structure : "fi" dans ce cas.

### Alternative conditionnelle

La deuxième forme est l'alternative comme dans les langages de programmation classiques.

```
if [ condition ]
then
   commande1
   commande2
   ...
else
  commandej
  commandej+1
  ...
fi
```

### Choix multiples

Par extension, le choix entre de multiples possibilités peut être réalisé par une nouvelle extension de la syntaxe :

```
if [ condition ]
then
```

```
    commande1
    commande2
    ...
elif [ condition2 ]
then
   commandej
   commandej+1
   ...
else
   commandek
   commandek+1
   ...
fi
```

On note donc une très grande similitude de construction avec le C ou d'autres langages procéduraux.

On note également une susceptibilité de l'analyseur syntaxique qui demande un respect assez pointu de l'écriture et de l'agencement syntaxique.