Fonctionnalités avancées du Shell

- Objet : Fonctionnalités avancées du Shell
- Niveau requis :
  avisé
- Commentaires : *Faire des choses compliquées avec votre shell.*
- À savoir : shell
- Suivi :
  - Création par captnfab le 30/09/2013
  - Testé par smolski le 01/10/2013
- Commentaires sur le forum : ici [1)]

Boucles et instructions conditionnelles

Une instruction conditionnel (souvent appelé *bloc if-then-else* ou *bloc si-alors-sinon* se construit via une condition, et exécute une série de commandes ou une autre suivant le résultat de la condition.

Les conditions

En shell, les conditions s'expriment de la manière suivante :

Sur les fichiers et les dossiers

Condition	Description	Exemple
`[ -e fichier ]`	*vrai* si `fichier` existe	`[ -e /etc/network/interfaces ]`
`[ -d dossier ]`	*vrai* si `dossier` est un répertoire	`[ -d /etc/network ]`
`[ -f fichier ]`	*vrai* si `fichier` est un fichier régulier	`[ -f /etc/resolv.conf ]`
`[ -L fichier ]`	*vrai* si `fichier` est un lien symbolique	`[ -L /usr/bin/x-terminal-emulator ]`
`[ -r fichier ]`	*vrai* si fichier est lisible ®	`[ -r /bin/bash ]`
`[ -w fichier ]`	*vrai* si fichier est modifiable (w)	`[ -w /tmp ]`
`[ -x fichier ]`	*vrai* si fichier est exécutable (x)	`[ -x /sbin/ifconfig ]`
`[ f1 -nt f2 ]`	*vrai* si f1 plus récent que f2	`[ /tmp/bla -nt /tmp/bli ]`
`[ f1 -ot f2 ]`	*vrai* si f1 plus ancien que f2	`[ /tmp/bla -ot /tmp/bli ]`

Sur les chaînes de caractères

Condition	Description	Exemple
`[ -z chaîne ]`	*vrai* si `chaîne` est vide	`[ -z "$VAR" ]`
`[ -n chaîne ]`	*vrai* si `chaîne` est non-vide	`[ -n "$VAR" ]`
`[ c1 = c2 ]`	*vrai* si les chaînes `c1` et `c2` sont identiques	`[ "$ARG1" = "$VAR" ]`
`[ c1 != c2 ]`	*vrai* si les chaînes `c1` et `c2` sont différentes	`[ "plouf" != "$VAR" ]`

Sur les nombres entiers

Condition	Description	Exemple
`[ n1 -eq n2 ]`	*vrai* si les nombres n1 et n2 sont égaux	`[ "$num" -eq 42 ]`
`[ n1 -ne n2 ]`	*vrai* si les nombres n1 et n2 sont inégaux	`[ "$num" -ne 42 ]`
`[ n1 -lt n2 ]`	*vrai* si les n1 est strictement inférieur à n2	`[ "$num" -lt 42 ]`
`[ n1 -le n2 ]`	*vrai* si les n1 est inférieur ou égal à n2	`[ "$num" -le 42 ]`
`[ n1 -gt n2 ]`	*vrai* si les n1 est strictement supérieur à n2	`[ "$num" -gt 42 ]`
`[ n1 -ge n2 ]`	*vrai* si les n1 est supérieur ou égal à n2	`[ "$num" -ge 42 ]`

Instruction conditionnelle ''if''

Pour exécuter `commande1` si `condition` est vraie, et `commande2` sinon, la syntaxe est la suivante :

```
if condition;
then
  commande1;
else
  commande2;
fi
```

`condition` est en fait un programme. On considère que `condition` est *vrai* s'il se termine bien et retourne un code de retour `0`, et *faux* sinon.

Par exemple, `/bin/true` est un programme fournissant une condition toujours vrai :

```
if /bin/true;
then
  echo 'Je suis dans le vrai'
else
  echo 'Je suis dans le faux'
fi
```

Le copié/collé puis la validation de cette instruction vous donnera :

```
Je suis dans le vrai.
```

Ce qui n'est donc pas un mensonge ! 😉

D'autres exemples utilisant les conditions prédéfinies :

Cette commande vérifie s'il existe un fichier `/bin/echo` qui soit exécutable. Si c'est le cas, il l'utilise pour afficher « Bonjour le monde. », sinon il utilise la commande `echo` intégrée au shell.

```
if [ -e /bin/echo ] && [ -x /bin/echo ];
then
  /bin/echo 'Bonjour le monde.'
else
  echo 'Bonjour le monde.'
```

```
fi
```

```
if [ "$USER" = "smolski" ];
then
  echo 'Salut chef !'
else
  echo "Bonjour $USER."
fi
```

Boucle tant que ''while''

```
while condition;
do
  commande;
done
```

Les conditions sont les mêmes que pour la commande `if`.

Boucle pour tout ''for''

```
for i in "les fleurs" "le lapin" "le chou" "la salade" "le staff DF";
do
  echo "J'aime $i."
done
```

Sélecteur ''case''

```
read x ; case "$x" in
  bonjour)
      echo "Bonjour $USER"
      ;;
  aurevoir)
      echo "Au revoir $USER"
      ;;
   *)
      echo "Que voulez-vous dire par '$x' $USER ?"
esac
```

Les fonctions

La syntaxe à utiliser dans le corps d'une fonction pour en récupérer les arguments est similaire à celle d'un fichier de script.
Pour une explication sur les différentes syntaxes voir : les-fonctions

Rappel :

- Pour le reste, le corps d'une fonction se définit de manière assez simple. Notons que les arguments ne sont pas précisés entre parenthèses.
- L'appel de la fonction se fait ensuite comme l'appel d'un programme (cf. dernière ligne.)

```
function ma_fonction()
{
echo "Commande initiale : $0 $@"

N=0
while [ -n "$1" ];
do
  N=$(($N+1))
  echo "Argument $N : $1";
  shift;
done
}

ma_fonction test bla coucou plouf
```

Quelques liens

Des scripts en cascade commentés en anglais.

La page documentation :

- http://mywiki.wooledge.org/BashGuide

La page scripts :

- http://mywiki.wooledge.org/BashFAQ/

Merci **cthuluh** !

Références

Guide avancé d'écriture des scripts Bash

[1)]
N'hésitez pas à y faire part de vos remarques, succès, améliorations ou échecs !