**Les conditions**

La prise de décision est un élément indispensable dans tout programme. Si on ne pouvait pas décider quoi faire, le programme ferait toujours la même chose… ce qui serait bien ennuyeux.

Les branchements conditionnels (que nous abrègerons « conditions ») constituent un moyen de dire dans notre script « SI cette variable vaut tant, ALORS fais ceci, SINON fais cela ».  
Si vous connaissez déjà un autre langage de programmation, cela doit vous être familier. Sinon, ne vous en faites pas, vous allez très vite comprendre le concept.

**if : la condition la plus simple**

Le type de condition le plus courant est le if, qui signifie « si ».

**Si**

Les conditions ont la forme suivante :

SI test\_de\_variable

ALORS

------> effectuer\_une\_action

FIN SI

Bien entendu, ce n'est pas du bash. Il s’agit juste d’un schéma pour vous montrer quelle est la forme d'une condition.

La syntaxe en bash est la suivante :

if [ test ]

then

        echo "C'est vrai"

fi

Le motfi(ifà l'envers !) à la fin indique que le if s'arrête là. Tout ce qui est entre lethenet lefisera exécuté uniquement si le test est vérifié.

Vous noterez — c'est très important — qu'il y a des espaces à l'intérieur des crochets. On ne doit pas écrire[test]mais[ test ] !

Il existe une autre façon d'écrire leif : en plaçant lethensur la même ligne. Dans ce cas, il ne faut pas oublier de rajouter un point-virgule après les crochets :

if [ test ]; then

        echo "C'est vrai"

fi

À la place du mot test, il faut indiquer votre test. C'est à cet endroit que vous testerez la valeur d'une variable, par exemple. Ici, nous allons voir un cas simple où nous testons la valeur d'une chaîne de caractères, puis nous apprendrons à faire des tests plus compliqués un peu plus loin dans le chapitre.

Faisons quelques tests sur un script que nous appellerons conditions.sh :

#!/bin/bash

nom="Bruno"

if [ $nom = "Bruno" ]

then

        echo "Salut Bruno !"

fi

Comme$nomest bien égal à « Bruno », ce script affichera :

Salut Bruno !

Essayez de changer le test : si vous n'écrivez pas précisément « Bruno », le if ne sera pas exécuté et votre script n'affichera donc rien.

Notez aussi que vous pouvez tester deux variables à la fois dans le if :

#!/bin/bash

nom1="Bruno"

nom2="Marcel"

if [ $nom1 = $nom2 ]

then

        echo "Salut les jumeaux !"

fi

Comme ici$nom1est différent de$nom2, le contenu du if ne sera pas exécuté. Le script n'affichera donc rien.

**Sinon**

Si vous souhaitez faire quelque chose de particulier quand la condition n'est **pas** remplie, vous pouvez rajouter un else qui signifie « sinon ».

En français, cela s'écrirait comme ceci :

SI test\_de\_variable

ALORS

------> effectuer\_une\_action

SINON

------> effectuer\_une\_action

FIN SI

if [ test ]

then

        echo "C'est vrai"

else

        echo "C'est faux"

fi

Reprenons notre script de tout à l'heure et ajoutons-lui un else :

#!/bin/bash

nom="Bruno"

if [ $nom = "Bruno" ]

then

        echo "Salut Bruno !"

else

        echo "J'te connais pas, ouste !"

fi

Bon : comme la variable vaut toujours la même chose, le else ne sera jamais exécuté, ce n'est pas rigolo. Je vous propose plutôt de vous baser sur le premier paramètre ($1) envoyé au script :

#!/bin/bash

if [ $1 = "Bruno" ]

then

        echo "Salut Bruno !"

else

        echo "J'te connais pas, ouste !"

fi

Testez maintenant votre script en lui donnant un paramètre :

$ ./conditions.sh Bruno

Salut Bruno !

Et si vous mettez autre chose :

$ ./conditions.sh Jean

J'te connais pas, ouste !

Notez que le script plante si vous oubliez de l'appeler avec un paramètre. Pour bien faire, il faudrait d'abord vérifier dans unifs'il y a au moins un paramètre. Nous apprendrons à faire cela plus loin.

**Sinon si**

Il existe aussi le mot cléelif, abréviation de « else if », qui signifie « sinon si ». Sa forme ressemble à ceci :

SI test\_de\_variable

ALORS

------> effectuer\_une\_action

SINON SI autre\_test

ALORS

------> effectuer\_une\_action

SINON SI encore\_un\_autre\_test

ALORS

------> effectuer\_une\_action

SINON

------> effectuer\_une\_action

FIN SI

C'est un peu plus compliqué, n’est-ce pas ?

Sachez que l'on peut mettre autant de « sinon si » que l'on veut ; là, j'en ai mis deux. En revanche, on ne peut mettre qu'un seul « sinon », qui sera exécuté à la fin si aucune des conditions précédentes n'est vérifiée.

Bash va d'abord analyser le premier test. S'il est vérifié, il effectuera la première action indiquée ; s'il ne l'est pas, il ira au premier « sinon si », au second, etc., jusqu'à trouver une condition qui soit vérifiée. Si aucune condition ne l’est, c'est le « sinon » qui sera lu.

Bien ! Voyons comment cela s'écrit en bash :

if [ test ]

then

        echo "Le premier test a été vérifié"

elif [ autre\_test ]

then

        echo "Le second test a été vérifié"

elif [ encore\_autre\_test ]

then

        echo "Le troisième test a été vérifié"

else

        echo "Aucun des tests précédents n'a été vérifié"

fi

On peut reprendre notre script précédent et l'adapter pour utiliser des elif :

#!/bin/bash

if [ $1 = "Bruno" ]

then

        echo "Salut Bruno !"

elif [ $1 = "Michel" ]

then

        echo "Bien le bonjour Michel"

elif [ $1 = "Jean" ]

then

        echo "Hé Jean, ça va ?"

else

        echo "J'te connais pas, ouste !"

fi

Vous pouvez tester ce script ; encore une fois, n'oubliez pas d'envoyer un paramètre sinon il plantera, ce qui est normal.

**Les tests**

Voyons maintenant un peu quels sont les tests que nous pouvons faire. Pour l'instant, on a juste vérifié si deux chaînes de caractères étaient identiques, mais on peut faire beaucoup plus de choses que cela !

**Les différents types de tests**

Il est possible d’effectuer trois types de tests différents en bash :

* des tests sur des chaînes de caractères ;
* des tests sur des nombres ;
* des tests sur des fichiers.

Nous allons maintenant découvrir tous ces types de tests et les essayer. :)

**Tests sur des chaînes de caractères**

Comme vous devez désormais le savoir, en bash toutes les variables sont considérées comme des chaînes de caractères. Il est donc très facile de tester ce que vaut une chaîne de caractères. Vous trouverez les différents types de tests disponibles sur le tableau suivant.

Vérifions par exemple si deux paramètres sont différents :

#!/bin/bash

if [ $1 != $2 ]

then

        echo "Les 2 paramètres sont différents !"

else

        echo "Les 2 paramètres sont identiques !"

fi

$ ./conditions.sh Bruno Bernard

Les 2 paramètres sont différents !

$ ./conditions.sh Bruno Bruno

Les 2 paramètres sont identiques !

| **Condition** | **Signification** |
| --- | --- |
| $chaine1 = $chaine2 | Vérifie si les deux chaînes sont identiques. Notez que bash est sensible à la casse : « b » est donc différent de « B ». Il est aussi possible d'écrire « == » pour les habitués du langage C. |
| $chaine1 != $chaine2 | Vérifie si les deux chaînes sont différentes. |
| -z $chaine | Vérifie si la chaîne est vide. |
| -n $chaine | Vérifie si la chaîne est non vide. |

On peut aussi vérifier si le paramètre existe avec-z (vérifie si la chaîne est vide). En effet, si une variable n'est pas définie, elle est considérée comme vide par bash. On peut donc par exemple s'assurer que$1existe en faisant comme suit :

#!/bin/bash

if [ -z $1 ]

then

        echo "Pas de paramètre"

else

        echo "Paramètre présent"

fi

$ ./conditions.sh

Pas de paramètre

$ ./conditions.sh param

Paramètre présent

**Tests sur des nombres**

Bien que bash gère les variables comme des chaînes de caractères pour son fonctionnement interne, rien ne nous empêche de faire des comparaisons de nombres si ces variables en contiennent. Vous trouverez les différents types de tests disponibles sur le tableau suivant.

| **Condition** | **Signification** |
| --- | --- |
| $num1 -eq $num2 | Vérifie si les nombres sont égaux (**eq***ual*). À ne pas confondre avec le « = » qui, lui, compare deux chaînes de caractères. |
| $num1 -ne $num2 | Vérifie si les nombres sont différents (**n***on***e***qual*). Encore une fois, ne confondez pas avec « != » qui est censé être utilisé sur des chaînes de caractères. |
| $num1 -lt $num2 | Vérifie sinum1est inférieur ( < ) ànum2(**l***ower***t***han*). |
| $num1 -le $num2 | Vérifie sinum1est inférieur ou égal ( <= ) ànum2(**l***oweror***e***qual*). |
| $num1 -gt $num2 | Vérifie sinum1est supérieur ( > ) ànum2(**g***reater***t***han*). |
| $num1 -ge $num2 | Vérifie sinum1est supérieur ou égal ( >= ) ànum2(**g***reateror***e***qual*). |

Vérifions par exemple si un nombre est supérieur ou égal à un autre nombre :

#!/bin/bash

if [ $1 -ge 20 ]

then

        echo "Vous avez envoyé 20 ou plus"

else

        echo "Vous avez envoyé moins de 20"

fi

$ ./conditions.sh 23

Vous avez envoyé 20 ou plus

$ ./conditions.sh 11

Vous avez envoyé moins de 20

**Tests sur des fichiers**

Un des avantages de bash sur d'autres langages est que l'on peut très facilement faire des tests sur des fichiers : savoir s'ils existent, si on peut écrire dedans, s'ils sont plus vieux, plus récents, etc. Le tableau suivant présente les différents types de tests disponibles.

| **Condition** | **Signification** |
| --- | --- |
| -e $nomfichier | Vérifie si le fichier existe. |
| -d $nomfichier | Vérifie si le fichier est un répertoire. N'oubliez pas que sous Linux, tout est considéré comme un fichier, même un répertoire ! |
| -f $nomfichier | Vérifie si le fichier est un… fichier. Un vrai fichier cette fois, pas un dossier. |
| -L $nomfichier | Vérifie si le fichier est un lien symbolique (raccourci). |
| -r $nomfichier | Vérifie si le fichier est lisible (r). |
| -w $nomfichier | Vérifie si le fichier est modifiable (w). |
| -x $nomfichier | Vérifie si le fichier est exécutable (x). |
| $fichier1 -nt $fichier2 | Vérifie sifichier1est plus récent quefichier2(**n***ewer***t***han*). |
| $fichier1 -ot $fichier2 | Vérifie sifichier1est plus vieux quefichier2(**o***lder***t***han*). |

Je vous propose de faire un script qui demande à l'utilisateur d'entrer le nom d'un répertoire et qui vérifie si c'en est bien un :

#!/bin/bash

read -p 'Entrez un répertoire : ' repertoire

if [ -d $repertoire ]

then

        echo "Bien, vous avez compris ce que j'ai dit !"

else

        echo "Vous n'avez rien compris..."

fi

Entrez un répertoire : /home

Bien, vous avez compris ce que j'ai dit !

Entrez un répertoire : rienavoir.txt

Vous n'avez rien compris...

Notez que bash vérifie au préalable que le répertoire existe bel et bien.

**Effectuer plusieurs tests à la fois**

Dans unif, il est possible de faire plusieurs tests à la fois. En général, on vérifie :

* si un test est vrai **ET** qu'un autre test est vrai ;
* si un test est vrai **OU** qu'un autre test est vrai.

Les deux symboles à connaître sont :

* **&&** : signifie « et » ;
* **||** : signifie « ou ».

Il faut encadrer chaque condition par des crochets. Prenons un exemple :

#!/bin/bash

if [ $# -ge 1 ] && [ $1 = 'koala' ]

then

        echo "Bravo !"

        echo "Vous connaissez le mot de passe"

else

        echo "Vous n'avez pas le bon mot de passe"

fi

Le test vérifie deux choses :

* qu'il y a au moins un paramètre (« si$#est supérieur ou égal à 1 ») ;
* que le premier paramètre est bienkoala(« si$1est égal àkoala »).

Si ces deux conditions sont remplies, alors le message indiquant que l'on a trouvé le bon mot de passe s'affichera.

$ ./conditions.sh koala

Bravo !

Vous connaissez le mot de passe

Notez que les tests sont effectués l'un après l'autre et seulement s'ils sont nécessaires. Bash vérifie d'abord s'il y a au moins un paramètre. Si ce n'est pas le cas, il ne fera pas le second test puisque la condition ne sera de toute façon pas vérifiée.

**Inverser un test**

Il est possible d'inverser un test en utilisant la négation. En bash, celle-ci est exprimée par le point d'exclamation « ! ».

if [ ! -e fichier ]

then

        echo "Le fichier n'existe pas"

fi

Vous en aurez besoin, donc n'oubliez pas ce petit point d'exclamation.

**case : tester plusieurs conditions à la fois**

On a vu tout à l'heure unifun peu complexe qui faisait appel à deselifet à unelse :

#!/bin/bash

if [ $1 = "Bruno" ]

then

        echo "Salut Bruno !"

elif [ $1 = "Michel" ]

then

        echo "Bien le bonjour Michel"

elif [ $1 = "Jean" ]

then

        echo "Hé Jean, ça va ?"

else

        echo "J'te connais pas, ouste !"

fi

Ce genre de « grosifqui teste toujours la même variable » ne pose pas de problème mais n'est pas forcément très facile à lire pour le programmeur. À la place, il est possible d’utiliser l'instructioncasesi nous voulons.

Le rôle decaseest de tester la valeur d'une même variable, mais de manière plus concise et lisible.

Voyons comment on écrirait la condition précédente avec uncase :

#!/bin/bash

case $1 in

        "Bruno")

                echo "Salut Bruno !"

                ;;

        "Michel")

                echo "Bien le bonjour Michel"

                ;;

        "Jean")

                echo "Hé Jean, ça va ?"

                ;;

        \*)

                echo "J'te connais pas, ouste !"

                ;;

esac

Cela fait beaucoup de nouveautés d'un coup.  
Analysons la structure ducase !

case $1 in

Tout d'abord, on indique que l'on veut tester la valeur de la variable$1. Bien entendu, vous pouvez remplacer$1par n'importe quelle variable que vous désirez tester.

"Bruno")

Là, on teste une valeur. Cela signifie « Si$1est égal à Bruno ». Notez que l'on peut aussi utiliser une étoile comme joker : « B\* » acceptera tous les mots qui commencent par un B majuscule.

Si la condition est vérifiée, tout ce qui suit est exécuté jusqu'au prochain double point-virgule :

;;

Important, il ne faut pas l'oublier : le double point-virgule dit à bash d'arrêter là la lecture ducase. Il saute donc à la ligne qui suit leesacsignalant la fin ducase.

\*)

C'est en fait le « else » ducase. Si aucun des tests précédents n'a été vérifié, c'est alors cette section qui sera lue.

esac

Marque la fin ducase(esac, c'est « case » à l'envers !).

Nous pouvons aussi faire des « ou » dans uncase. Dans ce cas, petit piège, il ne faut pas mettre deux||mais un seul ! Exemple :

#!/bin/bash

case $1 in

        "Chien" | "Chat" | "Souris")

                echo "C'est un mammifère"

                ;;

        "Moineau" | "Pigeon")

                echo "C'est un oiseau"

                ;;

        \*)

                echo "Je ne sais pas ce que c'est"

                ;;

esac

**En résumé**

* On effectue des tests dans ses programmes grâce aux  if,  then,  [[ elif, then, fi] else,] fi.
* On peut comparer deux chaînes de caractères entre elles, mais aussi des nombres. On peut également effectuer des tests sur des fichiers : est-ce que celui-ci exis‌te ? Est-il exécutable ? Etc.
* Au besoin, il est possible de combiner plusieurs tests à la fois avec les symboles&&(ET) et||(OU).
* Le symbole!(point d'exclamation) exprime la négation dans une condition.
* Lorsque l'on effectue beaucoup de tests sur une même variable, il est parfois plus pratique d'utiliser un bloc case in… esac plutôt qu'un bloc if… fi.

Haut du formulaire