Unix / Linux - Shell Loop Control

Dans ce chapitre, nous discuterons du contrôle de boucle shell dans Unix. Jusqu'à présent, vous avez envisagé de créer des boucles et de travailler avec des boucles pour accomplir différentes tâches. Parfois, vous devez arrêter une boucle ou ignorer les itérations de la boucle.

Dans ce chapitre, nous apprendrons les deux instructions suivantes qui sont utilisées pour contrôler les boucles de coque −

- Le **pause** déclaration
- Le **continuer** déclaration

La boucle infinie

Toutes les boucles ont une durée de vie limitée et elles sortent une fois que la condition est fausse ou vraie selon la boucle.

Une boucle peut se poursuivre indéfiniment si la condition requise n'est pas remplie. Une boucle qui s'exécute pour toujours sans terminer les exécutions pendant un nombre infini de fois. Pour cette raison, ces boucles sont appelées boucles infinies.

Exemple

Voici un exemple simple qui utilise le **pendant** boucle pour afficher les nombres zéro à neuf −

```
#!/bin/sh

a=10

until [ $a -lt 10 ]
do
   echo $a
   a=`expr $a + 1`
done
```

Cette boucle continue pour toujours car **un** est toujours **supérieur à** ou **égal à 10** et il n'est jamais inférieur à 10.

La déclaration de rupture

Le **pause** instruction est utilisé pour mettre fin à l'exécution de la boucle entière, après avoir terminé l'exécution de toutes les lignes de code jusqu'à l'instruction break. Il descend ensuite vers le code après la fin de la boucle.

Syntaxe

The following **break** statement is used to come out of a loop −

```
break
```

The break command can also be used to exit from a nested loop using this format −

```
break n
```

Here **n** specifies the **n**th enclosing loop to the exit from.

Example

Here is a simple example which shows that loop terminates as soon as **a** becomes 5 −

```
#!/bin/sh

a=0

while [ $a -lt 10 ]
do
   echo $a
   if [ $a -eq 5 ]
   then
      break
   fi
   a=`expr $a + 1`
done
```

Upon execution, you will receive the following result −

```
0
1
2
3
4
5
```

Here is a simple example of nested for loop. This script breaks out of both loops if **var1 equals 2** and **var2 equals 0** −

Live Demo

```
#!/bin/sh

for var1 in 1 2 3
do
   for var2 in 0 5
   do
      if [ $var1 -eq 2 -a $var2 -eq 0 ]
      then
         break 2
      else
         echo "$var1 $var2"
      fi
   done
done
```

Lors de l'exécution, vous recevrez le résultat suivant. Dans la boucle intérieure, vous avez une commande break avec l'argument 2. Cela indique que si une condition est remplie, vous devez également sortir de la boucle externe et finalement de la boucle intérieure.

```
1 0
1 5
```

La déclaration continue

Le **continuer** la déclaration est similaire à la **pause** commande, sauf qu'elle fait sortir l'itération actuelle de la boucle, plutôt que la boucle entière.

Cette instruction est utile lorsqu'une erreur s'est produite, mais vous souhaitez essayer d'exécuter la prochaine itération de la boucle.

Syntaxe

```
continue
```

Comme avec l'instruction break, un argument entier peut être donné à la commande continue pour ignorer les commandes des boucles imbriquées.

```
continue n
```

Ici **n** précise le **n**^{e} boucle de verrouillage pour continuer.

Exemple

La boucle suivante utilise le **continuer** instruction qui revient de l'instruction continue et commence le traitement de l'instruction suivante −

Démo en direct

```
#!/bin/sh

NUMS="1 2 3 4 5 6 7"

for NUM in $NUMS
do
   Q=`expr $NUM % 2`
   if [ $Q -eq 0 ]
   then
      echo "Number is an even number!!"
      continue
   fi
   echo "Found odd number"
done
```

Lors de l'exécution, vous recevrez le résultat suivant −

```
Found odd number
Number is an even number!!
Found odd number
Number is an even number!!
Found odd number
Number is an even number!!
Found odd number
```