**Loops**

**A bash for loop is a bash programming language statement which allows code to be repeatedly executed. A for loop is classified as an iteration statement i.e. it is the repetition of a process within a bash script. For example, you can run UNIX command or task 5 times or read and process list of files using a for loop.**

**There are different types of loops in shell scripting:**

### **for Loop**

**The for loop iterates over a list of items (like words, strings, or numbers) and executes a block of commands for each item.**

**Syntax:**

**for variable in list**

**do**

**commands**

**done**

**variable: A placeholder that takes the value of each item in the list during each iteration.**

**list(value to variables): A list of items (strings, numbers, etc.) separated by spaces.**

#### **Examples:**

**Example 1: Iterating Over Strings**

**#!/bin/bash**

**for fruit in apple banana cherry**

**do**

**echo "Fruit: $fruit"**

**done**

**Output:**

**Fruit: apple**

**Fruit: banana**

**Fruit: cherry**

**Example 2: Iterating Over a Range of Numbers**

**#!/bin/bash**

**for i in 1 2 3 4 5**

**do**

**echo "Number: $i"**

**done**

**or**

**#!/bin/bash**

**for i in {1..5}**

**do**

**echo "Number: $i"**

**done**

**Output:**

**Number: 1**

**Number: 2**

**Number: 3**

**Number: 4**

**Number: 5**

**Example 3: Using C-like Syntax**

**#!/bin/bash**

**for ((i=1; i<=5; i++))**

**do**

**echo "Number: $i"**

**done**

**Output:**

**Number: 1**

**Number: 2**

**Number: 3**

**Number: 4**

**Number: 5**

**for loop with files in a shell script**

### **1. File Content in Columns**

**Suppose your file f1.txt has the following content, with each item on a new line:**

**apple**

**banana**

**cherry**

**dog**

### **2. Using a for Loop Directly on the File**

**If you use a for loop directly on the file content like this:**

**#!/bin/bash**

**for item in $(cat f1.txt)**

**do**

**echo "Item: $item"**

**done**

**Output:**

**Item: apple**

**Item: banana**

**Item: cherry**

**Item: dog**

### **Explanation:**

* **$(cat f1.txt): This command reads the entire file and splits the content into words. Since each item is on a separate line, it will treat each line as a separate word, making it work perfectly for this type of content.**
* **$item: Represents each line (or item) in the file.**

### **3. Potential Issues with Spaces or Special Characters**

**If the file contains lines with spaces or special characters, using the for loop with $(cat f1.txt) might cause issues because it splits content based on spaces, not lines.**

**Example:**

**apple**

**banana fruit**

**cherry**

**dog**

**Using the previous for loop, the output would be:**

**Output:**

**Item: apple**

**Item: banana**

**Item: fruit**

**Item: cherry**

**Item: dog**

### **2. while Loop**

**The while loop executes a block of commands as long as a specified condition is true.**

#### **Syntax:**

**while [ condition ]**

**do**

**commands**

**done**

* **condition: A test condition that is checked before each iteration. If the condition is true, the loop continues.**

#### **Example:**

**#!/bin/bash**

**counter=1**

**while [ $counter -le 5 ]**

**do**

**echo "Counter: $counter"**

**let counter++**

**done**

**Output:**

**Counter: 1**

**Counter: 2**

**Counter: 3**

**Counter: 4**

**Counter: 5**

**To Read content from file using while**

**To read the contents of a file line by line using a while loop in a shell script, you can use the while read construct.**

**#!/bin/bash**

**while read var\_name**

**do**

**echo "Line: $var\_name"**

**done < file.txt**

### **Using IFS**

**This example reads each line and splits it into fields based on a specified delimiter (e.g., comma). It ensures that leading and trailing whitespace is preserved.**

**#!/bin/bash**

**while IFS=, read -r field1 field2 field3**

**do**

**echo "Field1: $field1, Field2: $field2, Field3: $field3"**

**done < file.csv**

* **IFS=,: Sets the Internal Field Separator to a comma, which allows splitting of each line into fields based on the comma delimiter.**

### **3. until Loop**

**The until loop is similar to the while loop, but it continues to execute as long as the condition is false. It stops once the condition becomes true.**

#### **Syntax:**

**until [ condition ]**

**do**

**commands**

**done**

* **condition: A test condition that is checked before each iteration. If the condition is false, the loop continues.**

#### **Example:**

**#!/bin/bash**

**counter=1**

**until [ $counter -gt 5 ]**

**do**

**echo "Counter: $counter"**

**let counter++**

**done**

**Output:**

**Counter: 1**

**Counter: 2**

**Counter: 3**

**Counter: 4**

**Counter: 5**

**Infinite loop**

**An infinite loop is a loop that continues to execute indefinitely because the loop's terminating condition is never met. Infinite loops can occur in any type of loop (for, while, until), and they can be intentional (for example, to keep a program running until manually stopped) or accidental.**

### **1. while Loop Infinite Loop**

**The most common way to create an infinite loop is with a while loop where the condition is always true.**

#### **Example:**

**#!/bin/bash**

**while true**

**do**

**echo "This is an infinite loop. Press [CTRL+C] to stop."**

**sleep 1 # Pause for 1 second between iterations**

**done**

* **while true: The loop will continue to execute because true is always true.**
* **sleep 1: This command pauses the loop for 1 second during each iteration to avoid overwhelming the system.**

### **2. for Loop Infinite Loop**

**A for loop can also create an infinite loop if designed to iterate over an endless sequence.**

#### **Example:**

**#!/bin/bash**

**for (( ; ; ))**

**do**

**echo "This is an infinite loop. Press [CTRL+C] to stop."**

**sleep 1 # Pause for 1 second between iterations**

**done**

* **for (( ; ; )): The for loop here has no initialization, condition, or increment, so it runs indefinitely.**

### **3. until Loop Infinite Loop**

**An until loop can become infinite if the condition is always false.**

#### **Example:**

**#!/bin/bash**

**until false**

**do**

**echo "This is an infinite loop. Press [CTRL+C] to stop."**

**sleep 1 # Pause for 1 second between iterations**

**done**

* **until false: The loop continues to execute because false is always false.**

### **How to Stop an Infinite Loop**

**To stop an infinite loop, you can typically use CTRL+C in the terminal. This sends an interrupt signal (SIGINT) to the running script, which stops its execution.**

### **Accidental Infinite Loops**

**Accidental infinite loops can occur if the loop's exit condition is never met due to a bug or logical error. For example:**

#### **Example:**

**#!/bin/bash**

**counter=1**

**while [ $counter -le 10 ]**

**do**

**echo "Counter: $counter"**

**# Forgetting to increment counter causes an infinite loop**

**# ((counter++)) # Uncommenting this will fix the loop**

**done**

**Output:**

**Counter: 1**

**Counter: 1**

**Counter: 1**

**... (continues indefinitely)**

**In this example, since counter is never incremented, the condition [ $counter -le 10 ] remains true, causing an infinite loop.**

**IFS**

**The IFS (Internal Field Separator) is an environment variable in shell scripting that defines the delimiter used for splitting input into fields. By default, IFS is set to whitespace (spaces, tabs, and newlines), but you can change it to other characters (like commas or colons) to handle different types of input formats.**

**In this example, we use IFS to split a string into multiple variables.**

**#!/bin/bash**

**# Define a string with a comma delimiter**

**input\_string="apple,banana,cherry"**

**# Set IFS to a comma**

**IFS=,**

**# Read the string into variables**

**read -r item1 item2 item3 <<< "$input\_string"**

**# Output the variables**

**echo "Item1: $item1"**

**echo "Item2: $item2"**

**echo "Item3: $item3"**

**Suppose you have a file named file.csv with the following content:**

**apple,banana,cherry**

**dog,cat,mouse**

#### **Script:**

**#!/bin/bash**

**# Set IFS to a comma**

**IFS=,**

**# Read each line of the file using cat**

**cat file.csv | while read -r item1 item2 item3**

**do**

**echo "Item1: $item1, Item2: $item2, Item3: $item3"**

**done**

**Output:**

**Item1: apple, Item2: banana, Item3: cherry**

**Item1: dog, Item2: cat, Item3: mouse**

* **IFS=,: Sets the Internal Field Separator to a comma.**
* **cat file.csv | while read -r item1 item2 item3: Reads each line from the file and splits it into variables based on the comma delimiter.**

### **Basic Usage**

**Here's a breakdown of how IFS works with a simple example:**

### **1. Default IFS (Whitespace)**

**By default, IFS uses whitespace as the delimiter. This means that read will split input based on spaces, tabs, or newlines.**

#### **Example:**

**Suppose you have a file named file.txt with the following content:**

**apple banana cherry**

**dog cat mouse**

#### **Script:**

**#!/bin/bash**

**while read -r item1 item2 item3**

**do**

**echo "Item1: $item1, Item2: $item2, Item3: $item3"**

**done < file.txt**

**Output:**

**Item1: apple, Item2: banana, Item3: cherry**

**Item1: dog, Item2: cat, Item3: mouse**

* **read -r item1 item2 item3: Reads each line and splits it into three variables based on whitespace.**

### **2. Custom IFS (Comma Delimiter)**

**You can change IFS to split input based on a different delimiter, such as a comma.**

#### **Example:**

**Suppose you have a file named file.csv with the following content:**

**apple,banana,cherry**

**dog,cat,mouse**

#### **Script:**

**#!/bin/bash**

**IFS=, # Set IFS to a comma**

**while read -r item1 item2 item3**

**do**

**echo "Item1: $item1, Item2: $item2, Item3: $item3"**

**done < file.csv**

**Output:**

**Item1: apple, Item2: banana, Item3: cherry**

**Item1: dog, Item2: cat, Item3: mouse**

* **IFS=,: Sets the Internal Field Separator to a comma, so read splits each line into variables based on the comma delimiter.**

### **3. Custom IFS (Colon Delimiter)**

**You can also use other characters like a colon as the delimiter.**

#### **Example:**

**Suppose you have a file named file.txt with the following content:**

**apple:banana:cherry**

**dog:cat:mouse**

#### **Script:**

**#!/bin/bash**

**IFS=: # Set IFS to a colon**

**while read -r item1 item2 item3**

**do**

**echo "Item1: $item1, Item2: $item2, Item3: $item3"**

**done < file.txt**

**Output:**

**Item1: apple, Item2: banana, Item3: cherry**

**Item1: dog, Item2: cat, Item3: mouse**

* **IFS=:: Sets the Internal Field Separator to a colon, so read splits each line into variables based on the colon delimiter.**

**More examples like /etc/passwd using IFS**

**#!/bin/bash**

**# Set IFS to a colon**

**IFS=:**

**# Read each line of /etc/passwd using cat**

**cat /etc/passwd | while read -r username password uid gid fullname home shell**

**do**

**echo "Username: $username"**

**echo "Password: $password"**

**echo "UID: $uid"**

**echo "GID: $gid"**

**echo "Fullname: $fullname"**

**echo "Home: $home"**

**echo "Shell: $shell"**

**echo "------------------------"**

**done**