


- Sometimes we need to check the time taken by a set of commands. We can display it using the following code:

```
#!/bin/bash
#Filename: time_take.sh
start=$(date +%s)
commands;
statements;

end=$(date +%s)
difference=$(( end - start))
echo Time taken to execute commands is $difference seconds.
```

 An alternate method would be to use `time <scriptpath>` to get the time that it took to execute the script.

How it works...

While considering dates and time, epoch is defined as the number of seconds that have elapsed since midnight proleptic **Coordinated Universal Time (UTC)** of January 1, 1970, not counting leap seconds. Epoch time is very useful when you need to calculate the difference between two dates or time. You may find out the epoch times for two given timestamps and take the difference between the epoch values. Therefore, you can find out the total number of seconds between two dates.

To write a date format to get the output as required, use the following table:

Date component	Format
Weekday	%a (for example, Sat) %A (for example, Saturday)
Month	%b (for example, Nov) %B (for example, November)
Day	%d (for example, 31)
Date in format (mm/dd/yy)	%D (for example, 10/18/10)
Year	%Y (for example, 10) %Y (for example, 2010)
Hour	%I or %H (For example, 08)
Minute	%M (for example, 33)
Second	%S (for example, 10)

Date component	Format
Nano second	%N (for example, 695208515)
Epoch Unix time in seconds	%s (for example, 1290049486)

There's more...

Producing time intervals is very essential when writing monitoring scripts that execute in a loop. Let us see how to generate time delays.

Producing delays in a script

To delay execution in a script for a particular period of time, use `sleep:$ sleepno_of_seconds`. For example, the following script counts from 0 to 40 by using `tput` and `sleep`:

```
#!/bin/bash
#Filename: sleep.sh
echo -n Count:
tput sc

count=0;
while true;
do
    if [ $count -lt 40 ];
    then
        let count++;
        sleep 1;
        tput rc
        tput ed
        echo -n $count;
    else exit 0;
    fi
done
```

In the preceding example, a variable `count` is initialized to 0 and is incremented on every loop execution. The `echo` statement prints the text. We use `tput sc` to store the cursor position. On every loop execution we write the new count in the terminal by restoring the cursor position for the number. The cursor position is restored using `tput rc`. This clears text from the current cursor position to the end of the line, so that the older number can be cleared and the count can be written. A delay of 1 second is provided in the loop by using the `sleep` command.