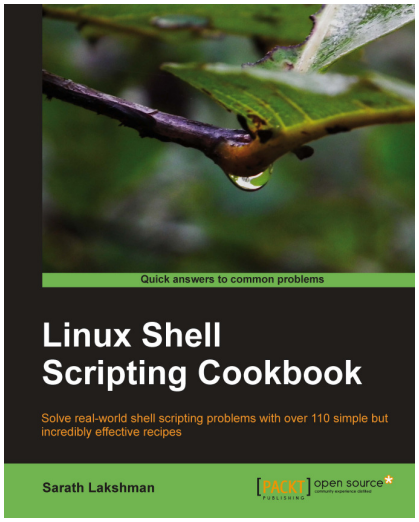
[PACKT] open source
community experience distilled
PUBLISHING

# Linux Shell Scripting Cookbook

**Sarath Lakshman**




## Chapter No. 8
## "Put on the Monitor's Cap"

## In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.8 "Put on the Monitor's Cap"

A synopsis of the book's content

Information on where to buy this book

# About the Author

**Sarath Lakshman** is a 21 year old who was bitten by the Linux bug during his teenage years. He is an undergraduate student of Computer Science and Engineering at Model Engineering College, Cochin, Kerala.

Sarath is a life hacker who loves to explore innovations. He is a GNU/Linux enthusiast and hactivist of free and open source software. He spends most of his time hacking with computers and having fun with his great friends. Sarath is well known as the developer of SLYNUX (2005)—a user friendly GNU/Linux distribution for Linux newbies. The free and open source software projects he has contributed to are PiTiVi Video editor, SLYNUX GNU/Linux distro, Swathantra Malayalam Computing, School-Admin, Istanbul, and the Pardus Project. He has authored many articles for the Linux For You magazine on various domains of FOSS technologies. He had made a contribution to several different open source projects during his multiple Google Summer of Code projects. He currently lives with his parents in Kerala. Sarath can be reached via his website `http://www.sarathlakshman.com`.

I would like to thank my friends and family for the great support and encouragement they have given me for all my endeavors. I would like to thank my friends Anu Mahadevan and Neenu Jacob for the tireless enthusiasm and patience to read through the chapter developments and providing comments during development. I would also like to thank Mr. Atanu Datta for helping me come up with the chapter titles. I extend my gratitude to the team at Packt Publishing who helped me in making this book happen.

**For More Information:**
**www.packtpub.com/linux-shell-scripting-cookbook/book**

# Linux Shell Scripting Cookbook

GNU/Linux is a remarkable operating system that comes with a complete development environment that is stable, reliable, and extremely powerful. The shell, being the native interface to communicate with the operating system, is capable of controlling the entire operating system. An understanding of shell scripting helps you to have better awareness of the operating system and helps you to automate most of the manual tasks with a few lines of script, saving you an enormous amount of time. Shell scripts can work with many external command-line utilities for tasks such as querying information, easy text manipulation, scheduling task running times, preparing reports, sending mails, and so on. There are numerous commands on the GNU/Linux shell, which are documented but hard to understand. This book is a collection of essential command-line script recipes along with detailed descriptions tuned with practical applications. It covers most of the important commands in Linux with a variety of use cases, accompanied by plenty of examples. This book helps you to perform complex data manipulations involving tasks such as text processing, file management, backups, and more with the combination of few commands.

Do you want to become the command-line wizard who performs any complex text manipulation task in a single line of code? Have you wanted to write shell scripts and reporting tools for fun or serious system administration? This cookbook is for you. Start reading!.

## What This Book Covers

*Chapter 1, Shell Something Out*, has a collection of recipes that covers the basic tasks such as printing in the terminal, performing mathematical operations, arrays, operators, functions, aliases, file redirection, and so on by using Bash scripting. This chapter is an introductory chapter for understanding the basic concepts and features in Bash.

*Chapter 2, Have a Good Command*, shows various commands that are available with GNU/Linux that come under practical usages in different circumstances. It introduces various essential commands such as cat, md5sum, find, tr, sort, uniq, split, rename, look, and so on. This chapter travels through different practical usage examples that users may come across and that they could make use of.

*Chapter 3, File In, File Out*, contains a collection of task recipes related to files and file systems. This chapter explains how to generate large size files, installing a file system on files and mounting files, finding and removing duplicate files, counting lines in a file, creating ISO images, collecting details about files, symbolic link manipulation, file permissions and file attributes, and so on.

**For More Information:**
www.packtpub.com/linux-shell-scripting-cookbook/book

*Chapter 4, Texting and Driving*, has a collection of recipes that explains most of the command-line text processing tools well under GNU/Linux with a number of task examples. It also has supplementary recipes for giving a detailed overview of regular expressions and commands such as sed and awk. This chapter goes through solutions to most of the frequently used text processing tasks in a variety of recipes.

*Chapter 5, Tangled Web? Not At All!*, has a collection of shell-scripting recipes that are adherent to the Internet and Web. This chapter is intended to help readers understand how to interact with the web using shell scripts to automate tasks such as collecting and parsing data from web pages, POST and GET to web pages, writing clients to web services, downloading web pages, and so on.

*Chapter 6, The Backup Plan*, shows several commands used for performing data backup, archiving, compression, and so on, and their usages with practical script examples. It introduces commands such as tar, gzip, bunzip, cpio, lzma, dd, rsync, git, squashfs, and much more. This chapter also walks through essential encryption techniques.

*Chapter 7, The Old-boy Network*, has a collection of recipes that talks about networking on Linux and several commands useful to write network-based scripts. The chapter starts with an introductory basic networking primer. Important tasks explained in the chapter include password-less login with SSH, transferring files through network, listing alive machines on a network, multi-cast messaging, and so on.

*Chapter 8, Put on the Monitor's Cap*, walks through several recipes related to monitoring activities on the Linux system and tasks used for logging and reporting. The chapter explains tasks such as calculating disk usage, monitoring user access, CPU usage, syslog, frequently used commands, and much more.

*Chapter 9, Administration Calls*, has a collection of recipes for system administration. This chapter explains different commands to collect details about the system, user management using scripting, sending messages to users, bulk image resizing, accessing MySQL databases from shell, and so on.

**For More Information:**
**www.packtpub.com/linux-shell-scripting-cookbook/book**

# 8
# Put on the Monitor's Cap

In this chapter, we will cover:

- Disk usage hacks
- Calculating the execution time for a command
- Information about logged users, boot logs, failure boots
- Printing the 10 most frequently-used commands
- Listing the top 10 CPU consuming process in 1 hour
- Monitoring command outputs with watch
- Logging access to files and directories
- Logfile management with logrotate
- Logging with syslog
- Monitoring user logins to find intruders
- Remote disk usage health monitoring
- Finding out active user hours on a system

**For More Information:**
www.packtpub.com/linux-shell-scripting-cookbook/book

# Introduction

An operating system consists of a collection of system software, designed for different purposes, serving different task sets. Each of these programs requires to be monitored by the operating system or the system administrator in order to know whether it is working properly or not. We will also use a technique called logging by which important information is written to a file while the application is running. By reading this file, we can understand the timeline of the operations that are taking place with a particular software or a daemon. If an application or a service crashes, this information helps to debug the issue and enables us to fix any issues. Logging and monitoring also helps to gather information from a pool of data. Logging and monitoring are important tasks for ensuring security in the operating system and for debugging purposes.

This chapter deals with different commands that can be used to monitor different activities. It also goes through logging techniques and their usages.

# Disk usage hacks

Disk space is a limited resource. We frequently perform disk usage calculation on hard disks or any storage media to find out the free space available on the disk. When free space becomes scarce, we will need to find out large-sized files that are to be deleted or moved in order to create free space. Disk usage manipulations are commonly used in shell scripting contexts. This recipe will illustrate various commands used for disk manipulations and problems where disk usages can be calculated with a variety of options.

## Getting ready

`df` and `du` are the two significant commands that are used for calculating disk usage in Linux. The command `df` stands for disk free and `du` stands for disk usage. Let's see how we can use them to perform various tasks that involve disk usage calculation.

## How to do it...

To find the disk space used by a file (or files), use:

```
$ du  FILENAME1 FILENAME2 ..
```

For example:

```
$ du file.txt
4
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

The result is, by default, shown as size in bytes.

In order to obtain the disk usage for all files inside a directory along with the individual disk usage for each file showed in each line, use:

```
$ du -a DIRECTORY
```

`-a` outputs results for all files in the specified directory or directories recursively.

Running `du DIRECTORY` will output a similar result, but it will show only the size consumed by subdirectories. However, they do not show the disk usage for each of the files. For printing the disk usage by files, `-a` is mandatory.

For example:

```
$  du -a test
4  test/output.txt
4  test/process_log.sh
4  test/pcpu.sh
16  test
```

An example of using `du DIRECTORY` is as follows:

```
$ du test
16  test
```

## There's more...

Let's go through additional usage practices for the `du` command.

### Displaying disk usage in KB, MB, or Blocks

By default, the disk usage command displays the total bytes used by a file. A more human-readable format is when disk usage is expressed in standard units KB, MB, or GB. In order to print the disk usage in a display-friendly format, use `-h` as follows:

```
du -h FILENAME
```

For example:

```
$ du -sh test/pcpu.sh
4.0K  test/pcpu.sh
# Multiple file arguments are accepted
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

Or:

```
# du -h DIRECTORY
$ du -h hack/
16K  hack/
```

## Displaying the grand total sum of disk usage

Suppose we need to calculate the total size taken by all the files or directories, displaying individual file sizes won't help. `du` has an option `-c` such that it will output the total disk usage of all files and directories given as an argument. It appends a line SIZE total with the result. The syntax is as follows:

```
$ du -c FILENAME1 FILENAME2..
```

For example:

```
du -c process_log.sh pcpu.sh
4  process_log.sh
4  pcpu.sh
8  total
```

Or:

```
$ du  -c DIRECTORY
```

For example:

```
$ du -c test/
16  test/
16  total
```

Or:

```
$ du -c *.txt
# Wildcards
```

`-c` can be used along with other options like `-a` and `-h`. It gives the same output as without using `-c`. The only difference is that it appends an extra line containing the total size.

There is another option `-s` (summarize), which will print only the grand total as the output. It will print the total sum, and flag `-h` can be used along with it to print in human readable format. This command has frequent use in practice. The syntax is as follows:

```
$ du -s FILES(s)
$ du -sh DIRECTORY
```



**For More Information:**
www.packtpub.com/linux-shell-scripting-cookbook/book

For example:

```
$ du -sh slynux
680K  slynux
```

## Printing files in specified units

We can force `du` to print the disk usage in specified units. For example:

- Print size in bytes (by default) by using:

  ```
  $ du -b FILE(s)
  ```

- Print the size in kilobytes by using:

  ```
  $ du -k FILE(s)
  ```

- Print the size in megabytes by using:

  ```
  $ du -m FILE(s)
  ```

- Print size in given BLOCK size specified by using:

  ```
  $ du -B BLOCK_SIZE FILE(s)
  ```

  Here, `BLOCK_SIZE` is specified in bytes.

An example consisting of all the commands is as follows:

```
$ du pcpu.sh
4  pcpu.sh
$ du -b pcpu.sh
439    pcpu.sh
$ du -k pcpu.sh
4  pcpu.sh
$ du -m pcpu.sh
1  pcpu.sh
$ du -B 4  pcpu.sh
1024  pcpu.sh
```

## Excluding files from disk usage calculation

There are circumstances when we need to exclude certain files from disk usage calculation. Such excluded files can be specified in two ways:

1. **Wildcards**

   We can specify a wildcard as follows:

   ```
   $ du --exclude "WILDCARD" DIRECTORY
   ```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

For example:

```
$ du --exclude "*.txt" FILES(s)
# Excludes all .txt files from calculation
```

2. **Exclude list**

   We can specify a list of files to be excluded from a file as follows:

```
$ du --exclude-from EXCLUDE.txt DIRECTORY
# EXCLUDE.txt is the file containing list
```

There are also some other handy options available with `du` to restrict the disk usage calculation. We can specify the maximum depth of the hierarchy that the `du` should traverse as a whole by calculating disk usage with the `--max-depth` parameter. Specifying a depth of `1` calculates the sizes of files in the current directory. Depth `2` will calculate files in the current directory and the next subdirectory and stop traversal at that second subdirectory.

For example:

```
$ du --max-depth 2 DIRECTORY
```

`du` can be restricted to traverse only a single file system by using the `-x` argument. Suppose `du DIRECTORY` is run, it will traverse through every possible subdirectory of `DIRECTORY` recursively. A subdirectory in the directory hierarchy may be a mount point (for example, `/mnt/sda1` is a subdirectory of `/mnt` and it is a mount point for the device `/dev/sda1`). `du` will traverse that mount point and calculate the sum of disk usage for that device filesystem also. In order to prevent `du` from traversing and to calculate from other mount points or filesystems, use the `-x` flag along with other `du` options. `du -x /` will exclude all mount points in `/mnt/` for disk usage calculation.

While using `du` make sure that the directories or files it traverses have the proper read permissions.

## Finding the 10 largest size files from a given directory

Finding large-size files is a regular task we come across. We regularly require to delete those huge size files or move them. We can easily find out large-size files using `du` and `sort` commands. The following one-line script can achieve this task:

```
$ du -ak SOURCE_DIR | sort -nrk 1 | head
```

Here `-a` specifies all directories and files. Hence `du` traverses the `SOURCE_DIR` and calculates the size of all files. The first column of the output contains the size in Kilobytes since `-k` is specified and the second column contains the file or folder name.



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

`sort` is used to perform numerical sort with column 1 and reverse it. `head` is used to parse the first 10 lines from the output.

For example:

```
$ du -ak /home/slynux | sort -nrk 1 | head -n 4
50220 /home/slynux
43296 /home/slynux/.mozilla
43284 /home/slynux/.mozilla/firefox
43276 /home/slynux/.mozilla/firefox/8c22khxc.default
```

One of the drawbacks of the above one-liner is that it includes directories in the result. However, when we need to find only the largest files and not directories we can improve the one-liner to output only the large-size files as follows:

```
$ find . -type f -exec du -k {} \; | sort -nrk 1 | head
```

We used `find` to filter only files to `du` rather than allow `du` to traverse recursively by itself.

## Disk free information

The `du` command provides information about the usage, whereas `df` provides information about free disk space. It can be used with and without `-h`. When `-h` is issued with `df` it prints the disk space in human readable format.

For example:

```
$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda1              9611492   2276840   6846412  25% /
none                    508828       240    508588   1% /dev
none                    513048       168    512880   1% /dev/shm
none                    513048        88    512960   1% /var/run
none                    513048         0    513048   0% /var/lock
none                    513048         0    513048   0% /lib/init/rw
none                   9611492   2276840   6846412  25% /var/lib/
ureadahead/debugfs

$ df -h
FilesystemSize  Used Avail Use% Mounted on
/dev/sda1              9.2G  2.2G  6.6G  25% /
none                   497M  240K  497M   1% /dev
none                   502M  168K  501M   1% /dev/shm
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

```
none                  502M   88K  501M   1% /var/run
none                  502M     0  502M   0% /var/lock
none                  502M     0  502M   0% /lib/init/rw
none                  9.2G  2.2G  6.6G  25% /var/lib/ureadahead/debugfs
```

# Calculating execution time for a command

While testing an application or comparing different algorithms for a given problem, execution time taken by a program is very critical. A good algorithm should execute in minimum amount of time. There are several situations in which we need to monitor the time taken for execution by a program. For example, while learning about sorting algorithms, how do you practically state which algorithm is faster? The answer to this is to calculate the execution time for the same data set. Let's see how to do it.

## How to do it...

`time` is a command that is available with any UNIX-like operating systems. You can prefix `time` with the command you want to calculate execution time, for example:

```
$ time COMMAND
```

The command will execute and its output will be shown. Along with output, the `time` command appends the time taken in `stderr`. An example is as follows:

```
$ time ls
test.txt
next.txt
real    0m0.008s
user    0m0.001s
sys     0m0.003s
```

It will show real, user, and system times for execution. The three different times can be defined as follows:

- **Real** is wall clock time—the time from start to finish of the call. This is all elapsed time including time slices used by other processes and the time that the process spends when blocked (for example, if it is waiting for I/O to complete).
- **User** is the amount of CPU time spent in user-mode code (outside the kernel) within the process. This is only the actual CPU time used in executing the process. Other processes and the time that the process spends when blocked do not count towards this figure.



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

- **Sys** is the amount of CPU time spent in the kernel within the process. This means executing the CPU time spent in system calls within the kernel, as opposed to library code, which is still running in the user space. Like 'user time', this is only the CPU time used by the process.

An executable binary of the `time` command is available at `/usr/bin/time` as well as a shell built-in named `time` exists. When we run `time`, it calls the shell built-in by default. The shell built-in time has limited options. Hence, we should use an absolute path for the executable (`/usr/bin/time`) for performing additional functionalities.

We can write this time statistics to a file using the `-o filename` option as follows:

```
$ /usr/bin/time -o output.txt COMMAND
```

The filename should always appear after the `–o` flag.

In order to append the time statistics to a file without overwriting, use the `-a` flag along with the `-o` option as follows:

```
$ /usr/bin/time –a -o output.txt COMMAND
```

We can also format the time outputs using format strings with the `-f` option. A format string consists of parameters corresponding to specific options prefixed with `%`. The format strings for real time, user time, and sys time are as follows:

- Real time - `%e`
- User - `%U`
- sys - `%S`

By combining parameter strings, we can create formatted output as follows:

```
$ /usr/bin/time -f "FORMAT STRING" COMMAND
```

For example:

```
$ /usr/bin/time -f "Time: %U" -a -o timing.log uname
Linux
```

Here `%U` is the parameter for user time.

When formatted output is produced, the formatted output of the command is written to the standard output and the output of the `COMMAND`, which is timed, is written to standard error. We can redirect the formatted output using a redirection operator (`>`) and redirect the time information output using the (`2>`) error redirection operator. For example:

```
$ /usr/bin/time -f "Time: %U" uname> command_output.txt 2>time.log
$ cat time.log
Time: 0.00
$ cat command_output.txt
Linux
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

Many details regarding a process can be collected using the `time` command. The important details include, exit status, number of signals received, number of context switches made, and so on. Each parameter can be displayed by using a suitable format string.

The following table shows some of the interesting parameters that can be used:

| Parameter | Description |
|---|---|
| `%C` | Name and command-line arguments of the command being timed. |
| `%D` | Average size of the process's unshared data area, in kilobytes. |
| `%E` | Elapsed real (wall clock) time used by the process in [hours:]minutes:seconds. |
| `%x` | Exit status of the command. |
| `%k` | Number of signals delivered to the process. |
| `%W` | Number of times the process was swapped out of the main memory. |
| `%Z` | System's page size in bytes. This is a per-system constant, but varies between systems. |
| `%P` | Percentage of the CPU that this job got. This is just user + system times divided by the total running time. It also prints a percentage sign. |
| `%K` | Average total (data + stack + text) memory usage of the process, in kilobytes. |
| `%w` | Number of times that the program was context-switched voluntarily, for instance while waiting for an I/O operation to complete. |
| `%c` | Number of times the process was context-switched involuntarily (because the time slice expired). |

For example, the page size can be displayed using the `%Z` parameters as follows:

```
$ /usr/bin/time -f "Page size: %Z bytes" ls> /dev/null
Page size: 4096 bytes
```

Here the output of the timed command is not required and hence the standard output is directed to the `/dev/null` device in order to prevent it from writing to the terminal.

More format strings parameters are available. Read `man time` for more details.

# Information about logged users, boot logs, and failure boot

Collecting information about the operating environment, logged in users, the time for which the computer has been powered on, and any boot failures are very helpful. This recipe will go through a few commands used to gather information about a live machine.

## Getting ready

This recipe will introduce the commands `who`, `w`, `users`, `uptime`, `last`, and `lastb`.



**For More Information:**
www.packtpub.com/linux-shell-scripting-cookbook/book

## How to do it...

To obtain information about users currently logged in to the machine use:

```
$ who
slynux   pts/0   2010-09-29 05:24 (slynuxs-macbook-pro.local)
slynux   tty7    2010-09-29 07:08 (:0)
```

Or:

```
$ w
 07:09:05 up  1:45,  2 users,  load average: 0.12, 0.06, 0.02
USER     TTY      FROM    LOGIN@   IDLE  JCPU PCPU WHAT
slynux   pts/0    slynuxs 05:24  0.00s  0.65s 0.11s sshd: slynux
slynux   tty7     :0      07:08  1:45m  3.28s 0.26s gnome-session
```

It will provide information about logged in users, the pseudo TTY used by the users, the command that is currently executing from the pseudo terminal, and the IP address from which the users have logged in. If it is localhost, it will show the hostname. `who` and `w` format outputs with slight difference. The `w` command provides more detail than `who`.

TTY is the device file associated with a text terminal. When a terminal is newly spawned by the user, a corresponding device is created in `/dev/` (for example, `/dev/pts/3`). The device path for the current terminal can be found out by typing and executing the command `tty`.

In order to list the users currently logged in to the machine, use:

```
$ users
Slynux slynux slynux hacker
```

If a user has opened multiple pseudo terminals, it will show that many entries for the same user. In the above output, the user `slynux` has opened three pseudo terminals. The easiest way to print unique users is to use `sort` and `uniq` to filter as follows:

```
$ users | tr ' ' '\n' | sort | uniq
slynux
hacker
```

We have used `tr` to replace `' '` with `'\n'`. Then combination of `sort` and `uniq` will produce unique entries for each user.

In order to see how long the system has been powered on, use:

```
$ uptime
 21:44:33 up  3:17,  8 users,  load average: 0.09, 0.14, 0.09
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

The time that follows the word `up` indicates the time for which the system has been powered on. We can write a simple one-liner to extract the uptime only.

Load average in uptime's output is a parameter that indicates system load. This is explained in more detail in the chapter, *Administration Calls!*. In order to get information about previous boot and user logged sessions, use:

```
$ last
slynux   tty7         :0               Tue Sep 28 18:27   still logged in
reboot   system boot  2.6.32-21-generi Tue Sep 28 18:10 - 21:46  (03:35)
slynux   pts/0        :0.0             Tue Sep 28 05:31 - crash  (12:39)
```

The `last` command will provide information about logged in sessions. It is actually a log of system logins that consists of information such as `tty` from which it has logged in, login time, status, and so on.

The `last` command uses the log file `/var/log/wtmp` for input log data. It is also possible to explicitly specify the log file for the last command using the `-f` option. For example:

```
$ last -f /var/log/wtmp
```

In order to obtain info about login sessions for a single user, use:

```
$ last USER
```

Get information about reboot sessions as follows:

```
$ last reboot
reboot   system boot  2.6.32-21-generi Tue Sep 28 18:10 - 21:48  (03:37)
reboot   system boot  2.6.32-21-generi Tue Sep 28 05:14 - 21:48  (16:33)
```

In order to get information about failed user login sessions use:

```
# lastb
test     tty8         :0                Wed Dec 15 03:56 - 03:56  (00:00)
slynux   tty8         :0                Wed Dec 15 03:55 - 03:55  (00:00)
```

You should run `lastb` as the root user.

# Printing the 10 most frequently-used commands

Terminal is the tool used to access the shell prompt where we type and execute commands. Users run many commands in the shell. Many of them are frequently used. A user's nature can be identified easily by looking at the commands he frequently uses. This recipe is a small exercise to find out 10 most frequently-used commands.



**For More Information:**
www.packtpub.com/linux-shell-scripting-cookbook/book

## Getting ready

Bash keeps track of previously typed commands by the user and stores in the file `~/.bash_history`. But it only keeps a specific number (say 500) of the recently executed commands. The history of commands can be viewed by using the command history or `cat ~/.bash_history`. We will use this for finding out frequently-used commands.

## How to do it...

We can get the list of commands from `~/.bash_history`, take only the command excluding the arguments, count the occurrence of each command, and find out the 10 commands with the highest count.

The following script can be used to find out frequently-used commands:

```
#!/bin/bash
#Filename: top10_commands.sh
#Description: Script to list top 10 used commands

printf "COMMAND\tCOUNT\n" ;

cat ~/.bash_history | awk '{ list[$1]++; } \
END{
for(i in list)
{
printf("%s\t%d\n",i,list[i]); }
}'| sort -nrk 2 | head
```

A sample output is as follows:

```
$ ./top10_commands.sh
COMMAND  COUNT
ping     80
ls       56
cat      35
ps       34
sudo     26
du       26
cd       26
ssh      22
sftp     22
clear    21
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

## How it works...

In the above script, the history file `~/.bash_history` is the source file used. The source input is passed to `awk` through a pipe. Inside `awk`, we have an associative array list. This array can use command names as index and it stores the count of the commands in array locations. Hence for each arrival or occurrence of a command it will increment by one (`list[$1]++`). `$1` is used as the index. `$1` is the first word of text in a line input. If `$0` were used it would contain all the arguments for the command also. For example, if `ssh 192.168.0.4` is a line from `.bash_history`, `$0` equals to `ssh 192.168.0.4` and `$1` equals to `ssh`.

Once all the lines of the history files are traversed, we will have the array with command names as indexes and their count as the value. Hence command names with maximum count values will be the commands most frequently used. Hence in the `END{}` block of `awk`, we traverse through the indexes of commands and print all command names and their counts. `sort -nrk 2` will perform a numeric sort based on the second column (`COUNT`) and reverse it. Hence we use the `head` command to extract only the first 10 commands from the list. You can customize the top 10 to top 5 or any other number by using the argument `head -n NUMBER`.

# Listing the top 10 CPU consuming process in a hour

CPU time is a major resource and sometimes we require to keep track of the processes that consume the most CPU cycles in a period of time. In regular desktops or laptops, it might not be an issue that the CPU is heavily consumed. However, for a server that handles numerous requests, CPU is a critical resource. By monitoring the CPU usage for a certain period we can identify the processes that keep the CPU busy all the time and optimize them to efficiently use the CPU or to debug them due to any other issues. This recipe is a practice with process monitoring and logging.

## Getting ready

`ps` is a command used for collecting details about the processes running on the system. It can be used to gather details such as CPU usage, commands under execution, memory usage, status of process, and so on. Processes that consume the CPU for one hour can be logged, and the top 10 can be determined by proper usage of `ps` and text processing. For more details on the `ps` command, see the chapter: *Administration Calls!*.



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

## How to do it...

Let's go through the following shell script for monitoring and calculating CPU usages in one hour:

```
#!/bin/bash
#Name: pcpu_usage.sh
#Description: Script to calculate cpu usage by processes for 1 hour

SECS=3600
UNIT_TIME=60

#Change the SECS to total seconds for which monitoring is to be
performed.
#UNIT_TIME is the interval in seconds between each sampling

STEPS=$(( $SECS / $UNIT_TIME ))

echo Watching CPU usage... ;

for((i=0;i<STEPS;i++))
do
  ps -eo comm,pcpu | tail -n +2 >> /tmp/cpu_usage.$$
  sleep $UNIT_TIME
done

echo
echo CPU eaters :

cat /tmp/cpu_usage.$$ | \
awk '
{ process[$1]+=$2; }
END{
  for(i in process)
  {
    printf("%-20s %s",i, process[i] ;
  }

   }' | sort -nrk 2 | head
rm /tmp/cpu_usage.$$
#Remove the temporary log file
```

A sample output is as follows:

```
$ ./pcpu_usage.sh
Watching CPU usage...
CPU eaters :
Xorg        20
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

```
firefox-bin   15
bash          3
evince        2
pulseaudio    1.0
pcpu.sh         0.3
wpa_supplicant  0
wnck-applet     0
watchdog/0      0
usb-storage     0
```

## How it works...

In the above script, the major input source is `ps -eocomm, pcpu`. `comm` stands for command name and `pcpu` stands for the CPU usage in percent. It will output all the process names and the CPU usage in percent. For each process there exists a line in the output. Since we need to monitor the CPU usage for one hour, we repeatedly take usage statistics using `ps -eo comm,pcpu | tail -n +2` and append to a file `/tmp/cpu_usage.$$` running inside a `for` loop with 60 seconds wait in each iteration. This wait is provided by `sleep 60`. It will execute `ps` once in each minute.

`tail -n +2` is used to strip off the header and `COMMAND  %CPU` in the `ps` output.

`$$` in `cpu_usage.$$` signifies that it is the process ID of the current script. Suppose PID is 1345, during execution it will be replaced as `/tmp/cpu_usage.1345`. We place this file in `/tmp` since it is a temporary file.

The statistics file will be ready after one hour and will contain 60 entries corresponding to the process status for each minute. Then `awk` is used to sum the total CPU usage for each process. An associative array process is used for the summation of CPU usages. It uses the process name as an array index. Finally, it sorts the result with a numeric reverse sort according to the total CPU usage and pass through head to obtain top 10 usage entries.

## See also

- *Basic awk primer* of *Chapter 4*, explains the awk command
- *head and tail - printing the last or first ten lines* of *Chapter 3*, explains the tail command



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

# Monitoring command outputs with watch

We might need to continuously watch the output of a command for a period of time in equal intervals. For example, for a large file copy, we need to watch the growing file size. In order to do that, newbies repeatedly type commands and press return a number of times. Instead we can use the watch command to view output repeatedly. This recipe explains how to do that.

## How to do it...

The `watch` command can be used to monitor the output of a command on the terminal at regular intervals. The syntax of the `watch` command is as follows:

```
$ watch COMMAND
```

For example:

```
$ watch ls
```

Or:

```
$ watch 'COMMANDS'
```

For example:

```
$ watch 'ls -l | grep "^d"'
# list only directories
```

This command will update the output at a default interval of two seconds.

We can also specify the time interval at which the output needs to be updated, by using `-n SECONDS`. For example:

```
$ watch -n 5 'ls -l'
#Monitor the output of ls -l at regular intervals of 5 seconds
```

## There's more

Let's explore an additional feature of the `watch` command.

### Highlighting the differences in watch output

In `watch`, there is an option for updating the differences that occur during the execution of the command at an update interval to be highlighted using colors. Difference highlighting can be enabled by using the `-d` option as follows:

```
$ watch -d 'COMMANDS'
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

# Logging access to files and directories

Logging of file and directory access is very helpful to keep track of changes that are happening to files and folders. This recipe will describe how to log user accesses.

## Getting ready

The `inotifywait` command can be used to gather information about file accesses. It doesn't come by default with every Linux distro. You have to install the `inotify-tools` package by using a package manager. It also requires the Linux kernel to be compiled with inotify support. Most of the new GNU/Linux distributions come with inotify enabled in the kernel.

## How to do it...

Let's walk through the shell script to monitor the directory access:

```
#/bin/bash
#Filename: watchdir.sh
#Description: Watch directory access
path=$1
#Provide path of directory or file as argument to script

inotifywait -m -r -e create,move,delete $path  -q
```

A sample output is as follows:

```
$ ./watchdir.sh .
./ CREATE new
./ MOVED_FROM new
./ MOVED_TO news
./ DELETE news
```

## How it works...

The previous script will log events create, move, and delete files and folders from the given path. The `-m` option is given for monitoring the changes continuously rather than going to exit after an event happens. `-r` is given for enabling a recursive watch the directories. `-e` specifies the list of events to be watched. `-q` is to reduce the verbose messages and print only required ones. This output can be redirected to a log file.

We can add or remove the event list. Important events available are as follows:



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

| Event | Description |
|---|---|
| `access` | When some read happens to a file. |
| `modify` | When file contents are modified. |
| `attrib` | When metadata is changed. |
| `move` | When a file undergoes move operation. |
| `create` | When a new file is created. |
| `open` | When a file undergoes open operation. |
| `close` | When a file undergoes close operation. |
| `delete` | When a file is removed. |

# Logfile management with logrotate

Logfiles are essential components of a Linux system's maintenance. Logfiles help to keep track of events happening on different services on the system. This helps the sysadmin to debug issues and also provides statistics on events happening on the live machine. Management of logfiles is required because as time passes the size of a logfile gets bigger and bigger. Therefore, we use a technique called rotation to limit the size of the logfile and if the logfile reaches a size beyond the limit, it will strip the logfile and store the older entries from the logfile in an archive. Hence older logs can be stored and kept for future reference. Let's see how to rotate logs and store them.

## Getting ready

`logrotate` is a command every Linux system admin should know. It helps to restrict the size of logfile to the given SIZE. In a logfile, the logger appends information to the log file. Hence the recent information appears at the bottom of the log file. `logrotate` will scan specific logfiles according to the configuration file. It will keep the last 100 kilobytes (for example, specified SIZE = 100k) from the logfile and move rest of the data (older log data) to a new file `logfile_name.1` with older entries. When more entries occur in the logfile (`logfile_name.1`) and it exceeds the SIZE, it updates the logfile with recent entries and creates `logfile_name.2` with older logs. This process can easily be configured with `logrotate`. `logrotate` can also compress the older logs as `logfile_name.1.gz`, `logfile_name2.gz`, and so on. The option for whether older log files are to be compressed or not is available with the `logrotate` configuration.

## How to do it...

`logrotate` has the configuration directory at `/etc/logrotate.d`. If you look at this directory by listing contents, many other logfile configurations can be found.



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

We can write our custom configuration for our logfile (say `/var/log/program.log`) as follows:

```
$ cat /etc/logrotate.d/program
/var/log/program.log {
missingok
notifempty
size 30k
  compress
weekly
  rotate 5
create 0600 root root
}
```

Now the configuration is complete. `/var/log/program.log` in the configuration specifies the logfile path. It will archive old logs in the same directory path. Let's see what each of these parameters are:

| Parameter | Description |
|---|---|
| `missingok` | Ignore if the logfile is missing and return without rotating the log. |
| `notifempty` | Only rotate the log if the source logfile is not empty. |
| `size 30k` | Limit the size of the logfile for which the rotation is to be made. It can be 1M for 1MB. |
| `compress` | Enable compression with gzip for older logs. |
| `weekly` | Specify the interval at which the rotation is to be performed. It can be weekly, yearly, or daily. |
| `rotate 5` | It is the number of older copies of logfile archives to be kept. Since 5 is specified, there will be `program.log.1.gz`, `program.log.2.gz`, and so on till `program.log.5.gz`. |
| `create 0600 root root` | Specify the mode, user, and the group of the logfile archive to be created. |

The options specified in the table are optional; we can specify the required options only in the `logrotate` configuration file. There are numerous options available with `logrotate`. Please refer to the man pages (`http://linux.die.net/man/8/logrotate`) for more information on `logrotate`.



**For More Information:**
www.packtpub.com/linux-shell-scripting-cookbook/book

# Logging with syslog

Logfiles are an important component of applications that provide services to the users. An applications writes status information to its logfile while it is running. If any crash occurs or we need to enquire some information about the service, we look into the logfile. You can find lots of logfiles related to different daemons and applications in the `/var/log` directory. It is the common directory for storing log files. If you read through a few lines of the logfiles, you can see that lines in the log are in a common format. In Linux, creating and writing log information to logfiles at `/var/log` are handled by a protocol called syslog. It is handled by the `syslogd` daemon. Every standard application makes use of syslog for logging information. In this recipe, we will discuss how to make use of `syslogd` for logging information from a shell script.

## Getting ready

Logfiles are useful for helping you deduce what is going wrong with a system. Hence while writing critical applications, it is always a good practice to log the progress of application with messages into a logfile. We will learn the command logger to log into log files with `syslogd`. Before getting to know how to write into logfiles, let's go through a list of important logfiles used in Linux:

| Log file | Description |
|---|---|
| `/var/log/boot.log` | Boot log information. |
| `/var/log/httpd` | Apache web server log. |
| `/var/log/messages` | Post boot kernel information. |
| `/var/log/auth.log` | User authentication log. |
| `/var/log/dmesg` | System boot up messages. |
| `/var/log/mail.log` | Mail server log. |
| `/var/log/Xorg.0.log` | X Server log. |

## How to do it...

In order to log to the syslog file `/var/log/messages` use:

```
$ logger LOG_MESSAGE
```

For example:

```
$ logger This is a test log line
$ tail -n 1 /var/log/messages
Sep 29 07:47:44 slynux-laptop slynux: This is a test log line
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

The logfile `/var/log/messages` is a general purpose logfile. When the `logger` command is used, it logs to `/var/log/messages` by default. In order to log to the syslog with a specified tag, use:

```
$ logger -t TAG This is a message

$ tail -n 1 /var/log/messages
Sep 29 07:48:42 slynux-laptop TAG: This is a message
```

syslog handles a number of logfiles in `/var/log`. However, while logger sends a message, it uses the tag string to determine in which logfile it needs to be logged. `syslogd` decides to which file the log should be made by using the `TAG` associated with the log. You can see the tag strings and associated logfiles from the configuration files located in the `/etc/rsyslog.d/` directory.

In order to log to the system log with the last line from another logfile use:

```
$ logger -f /var/log/source.log
```

## See also

- *head and tail - printing the last or first 10 lines* of *Chapter 3*, explains the head and tail commands

# Monitoring user logins to find intruders

Logfiles can be used to gather details about the state of the system. Here is an interesting scripting problem statement:

We have a system connected to the Internet with SSH enabled. Many attackers are trying to log in to the system. We need to design an intrusion detection system by writing a shell script. Intruders are defined as users who are trying to log in with multiple attempts for more than two minutes and whose attempts are all failing. Such users are to be detected and a report should be generated with the following details:

- User account to which a login is attempted
- Number of attempts
- IP address of the attacker
- Host mapping for IP address
- Time range for which login attempts are performed.



**For More Information:**
www.packtpub.com/linux-shell-scripting-cookbook/book

## Getting started

We can write a shell script that can scan through the logfiles and gather the required information from them. Here, we are dealing with SSH login failures. The user authentication session log is written to the log file `/var/log/auth.log`. The script should scan the log file to detect the failure login attempts and perform different checks on the log to infer the data. We can use the `host` command to find out the host mapping from the IP address.

## How to do it...

Let's write an intruder detection script that can generate a report of intruders by using the authentication logfile as follows:

```
#!/bin/bash
#Filename: intruder_detect.sh
#Description: Intruder reporting tool with auth.log input
AUTHLOG=/var/log.auth.log

if [[ -n $1 ]];
then
  AUTHLOG=$1
  echo Using Log file : $AUTHLOG
fi

LOG=/tmp/valid.$$.log
grep -v "invalid" $AUTHLOG > $LOG
users=$(grep "Failed password" $LOG | awk '{ print $(NF-5) }' | sort |
uniq)

printf "%-5s|%-10s|%-10s|%-13s|%-33s|%s\n" "Sr#" "User" "Attempts" "IP
address" "Host_Mapping" "Time range"

ucount=0;

ip_list="$(egrep -o "[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+" $LOG | sort |
uniq)"

for ip in $ip_list;
do
  grep $ip $LOG > /tmp/temp.$$.log

for user in $users;
do
  grep $user /tmp/temp.$$.log> /tmp/$$.log
  cut -c-16 /tmp/$$.log > $$.time
  tstart=$(head -1 $$.time);
  start=$(date -d "$tstart" "+%s");

  tend=$(tail -1 $$.time);
  end=$(date -d "$tend" "+%s")

  limit=$(( $end - $start ))
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

```
    if [ $limit -gt 120 ];
    then
      let ucount++;
      IP=$(egrep -o "[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+" /tmp/$$.log | head
-1 );
      TIME_RANGE="$tstart-->$tend"
      ATTEMPTS=$(cat /tmp/$$.log|wc -l);
      HOST=$(host $IP | awk '{ print $NF }' )
    printf "%-5s|%-10s|%-10s|%-10s|%-33s|%-s\n" "$ucount" "$user"
"$ATTEMPTS" "$IP" "$HOST" "$TIME_RANGE";
    fi
done
done
rm /tmp/valid.$$.log /tmp/$$.log $$.time /tmp/temp.$$.log 2> /dev/null
```

A sample output is as follows:

```
slynux@slynux-laptop:~$ ./intruder_detect.sh sampleauth.log
Using Log file : sampleauth.log

Sr#  |User   |Attempts|IP address    |Host_Mapping |Time range
1    |alice  |3       |203.110.250.34|attk1.foo.com|Oct 29 05:28:59 -->Oct 29 05:31:59
2    |bob1   |3       |203.110.251.31|attk2.foo.com|Oct 29 05:21:52 -->Oct 29 05:29:52
3    |bob2   |3       |203.110.250.34|attk1.foo.com|Oct 29 05:22:59 -->Oct 29 05:25:52
4    |gvraju |20      |203.110.251.31|attk2.foo.com|Oct 28 04:37:10 -->Oct 29 05:19:09
5    |root   |21      |203.110.253.32|attk3.foo.com|Oct 29 05:18:01 -->Oct 29 05:37:01
```

## How it works...

In the `intruder_detect.sh` script, we use the `auth.log` file as input. We can either provide a log file as input to the script by using a command-line argument to the script or, by default, it reads the `/var/log/auth.log` file. We need to log details about login attempts for valid user names only. When a login attempt for an invalid user occurs, a log similar to `Failed password for invalid user bob from 203.83.248.32 port 7016 ssh2` is logged to `auth.log`. Hence, we need to exclude all lines in the log file having the word "invalid". The `grep` command with the invert option (`-v`) is used to remove all logs corresponding to invalid users. The next step is to find out the list of users for which login attempts occurred and failed. The SSH will log lines similar to `sshd[21197]: Failed password for bob1 from 203.83.248.32 port 50035 ssh2` for a failed password.



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

Hence we should find all the lines with words "failed password". Now all the unique IP addresses are to be found out for extracting all the log lines corresponding to each IP address. The list of IP address is extracted by using a regular expression for IP address and the `egrep` command. A `for` loop is used to iterate through IP address and the corresponding log lines are found using `grep` and are written to a temporary file. The sixth word from the last word in the log line is the user name (for example, bob1 ). The `awk` command is used to extract the sixth word from the last word. `NF` returns the column number of the last word. Therefore, `NF-5` gives the column number of the sixth word from the last word. We use `sort` and `uniq` commands to produce a list of users without duplication.

Now we should collect the failed login log lines containing the name of each users. A `for` loop is used for reading the lines corresponding to each user and the lines are written to a temporary file. The first 16 characters in each of the log lines is the timestamp. The `cut` command is used to extract the timestamp. Once we have all the timestamps for failed login attempts for a user, we should check the difference in time between the first attempt and the last attempt. The first log line corresponds to the first attempt and last log line corresponds to last attempt. We have used `head -1` to extract the first line and `tail -1` to extract the last line. Now we have a time stamp for first (`tstart`) and last attempt (`tends`) in string format. Using the `date` command, we can convert the date in string representation to total seconds in UNIX Epoch time (the recipe, *Getting, setting dates, and delays* of *Chapter 1*, explains Epoch time).

The variables start and end have a time in seconds corresponding to the start and end timestamps in the date string. Now, take the difference between them and check whether it exceeds two minutes (120 seconds). Thus, the particular user is termed as an intruder and the corresponding entry with details are to be produced as a log. IP addresses can be extracted from the log by using a regular expression for IP address and the `egrep` command. The number of attempts is the number of log lines for the user. The number of lines can be found out by using the `wc` command. The host name mapping can be extracted from the output of the host command by running with IP address as argument. The time range can be printed using the timestamp we extracted. Finally, the temporary files used in the script are removed.

The above script is aimed only at illustrating a model for scanning the log and producing a report from it. It has tried to make the script smaller and simpler to leave out the complexity. Hence it has few bugs. You can improve the script by using better logic.

# Remote disk usage health monitor

A network consists of several machines with different users. The network requires centralized monitoring of disk usage of remote machines. The system administrator of the network needs to log the disk usage of all the machines in the network every day. Each log line should contain details such as the date, IP address of the machine, device, capacity of the device, used space, free space, percentage usage, and health status. If the disk usage of any of the partitions in any remote machine exceeds 80 percent, the health status should be set to ALERT, else it should be set to SAFE. This recipe will illustrate how to write a monitoring script that can collect details from remote machines in a network.



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

## Getting ready

We need to collect the disk usage statistics from each machine on the network, individually, and write a log file in the central machine. A script that collects the details and writes the log can be scheduled to run everyday at a particular time. The SSH can be used to log in to remote systems to collect disk usage data.

## How to do it...

First we have to set up a common user account on all the remote machines in the network. It is for the disklog program to log in to the system. We should configure auto-login with SSH for that particular user (the recipe, *Password-less auto-login with SSH* in *Chapter 7*, explains configuration of auto-login). We assume that there is a user called test in all remote machines configured with auto-login. Let's go through the shell script:

```
#!/bin/bash
#Filename: disklog.sh
#Description: Monitor disk usage health for remote systems

logfile="diskusage.log"

if [[ -n $1 ]]
then
  logfile=$1
fi

if [ ! -e $logfile ]
then
  printf "%-8s %-14s %-9s %-8s %-6s %-6s %-6s %s\n" "Date" "IP
address" "Device" "Capacity" "Used" "Free" "Percent" "Status" >
$logfile
fi

IP_LIST="127.0.0.1 0.0.0.0"
#provide the list of remote machine IP addresses

(
for ip in $IP_LIST;
do
  ssh slynux@$ip 'df -H' | grep ^/dev/ > /tmp/$$.df

  while read line;
  do
    cur_date=$(date +%D)
    printf "%-8s %-14s " $cur_date $ip
    echo $line | awk '{ printf("%-9s %-8s %-6s %-6s
%-8s",$1,$2,$3,$4,$5); }'

  pusg=$(echo $line | egrep -o "[0-9]+%")
```



**For More Information:**
www.packtpub.com/linux-shell-scripting-cookbook/book

```
    pusg=${pusg/\%/};
    if [ $pusg -lt 80 ];
    then
      echo SAFE
    else
      echo ALERT
    fi
    done< /tmp/$$.df
  done
  ) >> $logfile
```

We can schedule using the cron utility to run the script at regular intervals. For example, to run the script everyday at 10 am, write the following entry in the `crontab`:

```
00 10 * * * /home/path/disklog.sh /home/user/diskusg.log
```

Run the command `crontab -e`. Add the above line and save the text editor.

You can run the script manually as follows:

```
$ ./disklog.sh
```

A sample output log for the above script is as follows:

```
slynux@slynux-laptop:~/book$ cat diskusage.log
Date     IP address     Device    Capacity Used   Free   Percent Status
12/15/10 127.0.0.1      /dev/sda1 9.9G     2.4G   7.0G   26%     SAFE
12/15/10 0.0.0.0        /dev/sda1 9.9G     2.4G   7.0G   26%     SAFE
```

## How it works...

In the `disklog.sh` script, we can provide the logfile path as a command-line argument or else it will use the default logfile. If the logfile does not exists, it will write the logfile header text into the new file. `-e $logfile` is used to check whether the file exists or not. The list of IP addresses of remote machines are stored in the variable `IP_LIST` delimited with spaces. It should be made sure that all the remote systems listed in the `IP_LIST` have a common user `test` with auto-login with SSH configured. A `for` loop is used to iterate through each of the IP addresses. A remote command `df -H` is executed to get the disk free usage data using the `ssh` command. It is stored in a temporary file. A `while` loop is used to read the file line by line. Data is extracted using `awk` and is printed. The date is also printed. The percentage usage is extracted using the `egrep` command and `%` is replaced with `none` to get the numeric value of percent. It is checked whether the percentage value exceeds 80. If it is less than 80, the status is set as SAFE and if greater than or equal to 80, the status is set as ALERT. The entire printed data should be redirected to the logfile. Hence the portion of code is enclosed in a subshell `()` and the standard output is redirected to the logfile.



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

## See also

- *Scheduling with cron* of *Chapter 9*, explains the crontab command

# Finding out active user hours on a system

Consider a web server with shared hosting. Many users log in to and log out of the server every day. The user activity gets logged in the server's system log. This recipe is a practice task to make use of the system logs and to find out how many hours each of the users have spent on the server and rank them according to the total usage hours. A report should be generated with the details, such as the rank, user, first logged in date, last logged in date, number of times logged in, and total usage hours. Let's see how we can approach this problem.

## Getting ready

The `last` command is used to list the details about the login sessions of the users in a system. The log data is stored in the `/var/log/wtmp` file. By individually adding the session hours for each user we can find out the total usage hours.

## How to do it...

Let's go through the script to find out active users and generate the report:

```
#!/bin/bash
#Filename: active_users.sh
#Description: Reporting tool to find out active users

log=/var/log/wtmp

if [[ -n $1 ]];
then
  log=$1
fi

printf "%-4s %-10s %-10s %-6s %-8s\n" "Rank" "User" "Start" "Logins"
"Usage hours"

last -f $log | head -n -2   > /tmp/ulog.$$

cat /tmp/ulog.$$ |  cut -d' ' -f1 | sort | uniq> /tmp/users.$$

(
while read user;
do
  grep ^$user /tmp/ulog.$$ > /tmp/user.$$
  seconds=0

while read t
  do
```



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

```
    s=$(date -d $t +%s 2> /dev/null)
    let seconds=seconds+s
  done< <(cat /tmp/user.$$ | awk '{ print $NF }' | tr -d ')(')
  firstlog=$(tail -n 1 /tmp/user.$$ | awk '{ print $5,$6 }')
  nlogins=$(cat /tmp/user.$$ | wc -l)
  hours=$(echo "$seconds / 60.0" | bc)
  printf "%-10s %-10s %-6s %-8s\n"  $user "$firstlog" $nlogins $hours
done< /tmp/users.$$
) | sort -nrk 4 | awk '{ printf("%-4s %s\n", NR, $0) }'
rm /tmp/users.$$ /tmp/user.$$ /tmp/ulog.$$
```

A sample output is as follows:

```
$ ./active_users.sh
Rank User       Start      Logins Usage hours
1    easyibaa   Dec 11     531    11437311943
2    demoproj   Dec 10     350    7538718253
3    kjayaram   Dec 9      213    4587849555
4    cinenews   Dec 11     85     1830831769
5    thebenga   Dec 10     54     1163118745
6    gateway2   Dec 11     52     1120038550
7    soft132    Dec 12     49     1055420578
8    sarathla   Nov 1      45     969268728
9    gtsminis   Dec 11     41     883107030
10   agentcde   Dec 13     39     840029414
```

## How it works...

In the `active_users.sh` script, we can either provide the `wtmp` log file as a command-line argument or it will use the `defaulwtmp` log file. The `last -f` command is used to print the logfile contents. The first column in the logfile is the user name. By using `cut` we extract the first column from the logfile. Then the unique users are found out by using the `sort` and `uniq` commands. Now for each user, the log lines corresponding to their login sessions are found out using `grep` and are written to a temporary file. The last column in the last log is the duration for which the user logged a session. Hence in order to find out the total usage hours for a user, the session durations are to be added. The usage duration is in `(HOUR:SEC)` format and it is to be converted into seconds using the date command.



For More Information:
www.packtpub.com/linux-shell-scripting-cookbook/book

In order to extract the session hours for the users, we have used the `awk` command. For removing the parenthesis, `tr -d` is used. The list of usage hour string is passed to the standard input for the `while` loop using the `< ( COMMANDS )` operator. It acts as a file input. Each hour string, by using the `date` command, is converted into seconds and added to the variable `seconds`. The first login time for a user is in the last line and it is extracted. The number of login attempts is the number of log lines. In order to calculate the rank of each user according to the total usage hours, the data record is to be sorted in the descending order with usage hours as the key. For specifying the number reverse sort `-nr` option is used along with the `sort` command. `-k4` is used to specify the key column (usage hour). Finally, the output of the sort is passed to `awk`. The `awk` command prefixes a line number to each of the lines, which becomes the rank for each user.



**For More Information:**
www.packtpub.com/linux-shell-scripting-cookbook/book

## Where to buy this book

You can buy Linux Shell Scripting Cookbook from the Packt Publishing website:
`https://www.packtpub.com/linux-shell-scripting-cookbook/book`

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.


**www.PacktPub.com**

**For More Information:**
**www.packtpub.com/linux-shell-scripting-cookbook/book**