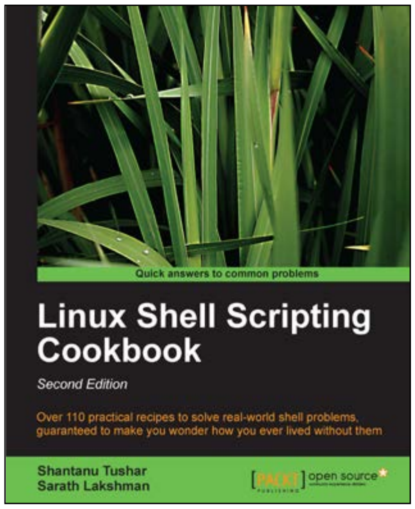
Linux Shell Scripting Cookbook
Second Edition

Shantanu Tushar
Sarath Lakshman



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

In this package, you will find:

A Biography of the authors 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 Authors

Shantanu Tushar is an advanced GNU/Linux user since his college days. He works as an application developer and contributes to the software in the KDE projects.

Shantanu has been fascinated by computers since he was a child, and spent most of his high school time writing C code to perform daily activities. Since he started using GNU/Linux, he has been using shell scripts to make the computer do all the hard work for him. He also takes time to visit students at various colleges to introduce them to the power of Free Software, including its various tools. Shantanu is a well-known contributor in the KDE community and works on Calligra, Gluon and the Plasma subprojects. He looks after maintaining Calligra Active – KDE's office document viewer for tablets, Plasma Media Center, and the Gluon Player. One day, he believes, programming will be so easy that everybody will love to write programs for their computers.

Shantanu can be reached by e-mail on `shantanu@kde.org`, `shantanutushar` on `identi.ca/twitter`, or his website `http://www.shantanutushar.com`.

I would like to thank my friends and family for the support and encouragement they've given me, especially to my sweet sister for her patience when I couldn't get time to talk to her. I am particularly thankful to Sinny Kumari for patiently testing the scripts to make sure they function properly and Sudhendu Kumar for helping me with the recipe on GNU Screen.

I must also thank Krishna, Madhusudan, and Santosh who introduced me to the wonderful world of GNU/Linux and Free Software. Also, a big thanks to all the reviewers of the book for taking the time to painfully go through every minute detail in the book and help me in improving it. I am also thankful to the whole team at Packt Publishing, without whose efforts and experience, this second edition wouldn't have happened.

Sarath Lakshman is a 23 year old who was bitten by the Linux bug during his teenage years. He is a software engineer working in ZCloud engineering group at Zynga, India. He 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. Currently, he is exploring his passion about scalable distributed systems in his spare time. Sarath can be reached via his website `http://www.sarathlakshman.com`.

Linux Shell Scripting Cookbook *Second Edition*

GNU/Linux is one of the most powerful and flexible operating systems in the world. In modern computing, there is absolutely no space where it is not used—from servers, portable computers, mobile phones, tablets to supercomputers, everything runs Linux. While there are beautiful and modern graphical interfaces available for it, the shell still remains the most flexible way of interacting with the system.

In addition to executing individual commands, a shell can follow commands from a script, which makes it very easy to automate tasks. Examples of such tasks are preparing reports, sending e-mails, performing maintenance, and so on. This book is a collection of chapters which contain recipes to demonstrate real-life usages of commands and shell scripts. You can use these as a reference, or an inspiration for writing your own scripts. The tasks will range from text manipulation to performing network operations to administrative tasks.

As with everything, the shell is only as awesome as you make it. When you become an expert at shell scripting, you can use the shell to the fullest and harness its true power. *Linux Shell Scripting Cookbook* shows you how to do exactly that!

What This Book Covers

Chapter 1, Shell Something Out, is an introductory chapter for understanding the basic concepts and features in Bash. We discuss printing text in the terminal, doing mathematical calculations, and other simple functionalities provided by Bash.

Chapter 2, Have a Good Command, shows commonly used commands that are available with GNU/Linux. This chapter travels through different practical usage examples that users may come across and that they could make use of. In addition to essential commands, this second edition talks about cryptographic hashing commands and a recipe to run commands in parallel, wherever possible.

Chapter 3, File In, File Out, contains a collection of recipes related to files and filesystems. This chapter explains how to generate large-size files, installing a filesystem on files, mounting files, and creating ISO images. We also deal with operations such as finding and removing duplicate files, counting lines in a file collecting details about files, and so on.

Chapter 4, Texting and Driving, has a collection of recipes that explains most of the commandline 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. It is an essential read for any serious task.

Chapter 5, Tangled Web? Not At All!, has a collection of shell-scripting recipes that talk to services on the Internet. 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. This is discussed using POST and GET to web pages, writing clients to web services. The second edition uses new authorization mechanisms such as OAuth for services such as Twitter.

Chapter 6, The Backup Plan, shows several commands used for performing data back up, archiving, compression, and so on. In addition to faster compression techniques, this second edition also talks about creating entire disk images.

Chapter 7, The Old-boy Network, has a collection of recipes that talks about networking on Linux and several commands useful for writing network-based scripts. The chapter starts with an introductory basic networking primer and goes on to cover usages of ssh – one of the most powerful commands on any modern GNU/Linux system. We discuss advanced port forwarding, setting up raw communication channels, configuring the firewall, and much more.

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, and CPU usage. In this second edition, we also learn how to optimize power consumption, monitor disks, and check their filesystems for errors.

Chapter 9, Administration Calls, has a collection of recipes for system administration. This chapter explains different commands to collect details about the system and user management using scripting. We also discuss bulk image resizing and accessing MySQL databases from the shell. New in this edition is that we learn how to use the GNU Screen to manage multiple terminals without needing a window manager.

8
Put on the Monitor's Cap

In this chapter, we will cover:

- Monitoring disk usage
- Calculating the execution time for a command
- Collecting information about logged in users, boot logs, and boot failures
- Listing the top 10 CPU consuming processes in an 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 monitor
- Finding out active user hours on a system
- Measuring and optimizing power usage
- Monitoring disk activity
- Checking disks and filesystems for errors

Introduction

An operating system consists of a collection of system software that is designed for different purposes. It is a good idea to monitor each of these programs in order to know whether they are working properly or not. We will also use a technique called logging by which we can get important information in a file while the program is running. The content of this file can be used to understand the timeline of operations that are taking place in a running program or daemon. For instance, if an application or a service crashes, this information helps to debug the issue and enables us to fix any issues.

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

Monitoring disk usage

Disk space is a limited resource. We frequently perform disk usage calculation on storage media (such as hard disks) to find out the free space available on them. When free space becomes scarce, we find out large files to be deleted or moved in order to create free space. In addition to this, disk usage manipulations are also used in shell scripting contexts. This recipe will illustrate various commands used for disk manipulations 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
```

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

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, this does 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 expressed in units such as 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 -h test/pcpu.sh
4.0K  test/pcpu.sh
# Multiple file arguments are accepted
```

Or

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

Displaying the grand total sum of disk usage

If 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.shpcpu.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`, in which case they will produce their usual output with 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 the flag `-h` can be used along with it to print in human-readable format. This combination has frequent use in practice:

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

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 the 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 the size in the 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 the disk usage calculation

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

- **Wildcards**: We can specify a wildcard as follows:

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

  For example:

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

- **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. With the `--max-depth` parameter, we can specify the maximum depth of the hierarchy du should traverse while calculating disk usage. Specifying a depth of `1` calculates the size of files in the current directory, a depth of `2`, specifies to calculate files in the current directory and the next subdirectory, and so on. For example:

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

du can be restricted to traverse only one filesystem 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. `-x` is used to prevent `du` from doing this. For example, `du -x /` will exclude all mount points in `/mnt/` for the 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 files is a task we come across regularly so that we can delete or move them. We can easily find out such files using `du` and `sort` commands like this:

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

Here, `-a` makes du traverse the `SOURCE_DIR` and calculates the size of all files and directories. 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.

`sort` is used to perform a 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 preceding 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 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. Use -h with df to print the disk space in human-readable format. For example:

```
$ df -h
Filesystem            Size  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
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 the execution time for a command

While testing an application's efficiency or comparing different algorithms to solve a given problem, the execution time taken is very critical. A good algorithm should execute in a minimum amount of time. Let's see how to calculate the execution time.

How to do it...

1. To measure the execution time, just prefix time to the command you want to run.

   For example:

   ```
   $ time COMMAND
   ```

   The command will execute and its output will be shown. Along with the 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.

> 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.

2. We can write these 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
   ```

3. We can also format the time outputs using format strings with `-f` option. A format string consists of parameters corresponding to specific options prefixed with `%`. 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 a 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 a 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
   ```

4. To show the page size, use 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 string parameters are available. Try `man time` for more details.

How it works...

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.
- **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 the library code, which is still running in the user space. Like user time, this is only the CPU time used by the process. Refer to the following table for a brief description of kernel mode (also known as supervisor mode) and the system call mechanism.

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.

Parameter	Description
`%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).

Collecting information about logged in users, boot logs, and boot failures

Collecting information about the operating environment, logged in users, the time for which the computer has been powered on, and 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 commands `who`, `w`, `users`, `uptime`, `last`, and `lastb`.

How to do it...

1. To obtain information about users currently logged into the machine use:

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

   This output lists the login name, the TTY used by the users, login time, and remote hostname (or X display information) about logged in users.

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

2. To obtain more detailed information about the logged in users, use:

```
$ 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
```

   This first line lists the current time, system uptime, number of users currently logged on, and the system load averages for the past 1, 5, and 15 minutes. Following this, the details about each login are displayed with each line containing the login name, the TTY name, the remote host, login time, idle time, total CPU time used by the user since login, CPU time of the currently running process, and the command line of their current process.

   Load average in the `uptime` command's output is a parameter that indicates system load. This is explained in more detail in *Chapter 9*, *Administration Calls*.

3. In order to list only the usernames of the users currently logged into the machine, use:

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

   If a user has opened multiple terminals, it will show that many entries for the same user. In the preceding 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 a combination of `sort` and `uniq` will produce unique entries for each user.

4. 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
```

   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:

```
$ uptime | grep -Po '\d{2}\:\d{2}\:\d{2}'
```

   This uses `grep` with a perl-style regex to extract only three two-digit numbers separated by colons.

5. 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-generic 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 the 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
```

6. In order to obtain information about login sessions for a single user, use:

```
$ last USER
```

7. 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)
```

8. 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.

Listing the top 10 CPU consuming processes in an hour

CPU is a major resource and it is good to keep a track of the processes that consume most of the CPU in a period of time. By monitoring the CPU usage for a certain period, we can identify the processes that keep the CPU busy all the time and troubleshoot them to efficiently use the CPU. In this recipe, we will discuss process monitoring and logging.

Getting ready

`ps` command is 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 processes, 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, refer to *Chapter 9*, *Administration Calls*.

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 -eocomm,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\n",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
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 preceding 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 -eocomm,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 sets of entries, each set containing 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 array index. Finally, it sorts the result with a numeric reverse sort according to the total CPU usage and pass through `head` to obtain the top 10 usage entries.

See also

- The *Using awk for advanced text processing* recipe of *Chapter 4*, *Texting and Driving*, explains the `awk` command
- The *Using head and tail for printing the last or first 10 lines* recipe of *Chapter 3*, *File In, File Out*, explains the `tail` command

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, while copying a large file, we might need to watch the growth of the file size. In order to do that, we can use the `watch` command to execute the `du` command and 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 the 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'
```

Logging access to files and directories

Logging of file and directory access is very helpful to keep a track of changes that are happening to files and folders. This recipe will describe how to log such 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, and `-r` enables a recursive watch of the directories (symbolic links are ignored). Finally, `-e` specifies the list of events to be watched and `-q` is to reduce the verbose messages and print only the 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:

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

Logfile management with logrotate

Logfiles are essential components of a Linux system to keep track of events happening on different services on the system. This helps to debug issues as well as provide statistics 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 techniques called **rotation** such that we limit the size of the logfile and if the logfile reaches a size beyond the limit, it will strip the logfile with that size and store the older entries in the logfile archived in log directories. Hence, older logs can be stored and kept for future references. 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 the 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 = 100 k) 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 of 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 its contents, many other logfile configurations can be found.

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.

How it works...

Let's see what each of the parameters in the configuration mean:

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 1 M for 1 MB.
`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 up to `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`.

Logging with syslog

Usually, logfiles related to different daemons and applications are located in the `/var/log` directory, as 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**, 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 very good 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 an 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:

Logfile	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...

Let's see how to use `logger` to create and manage log messages:

1. 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
   ```

   The logfile `/var/log/messages` is a general purpose logfile. When the `logger` command is used, it logs to `/var/log/messages` by default.

2. 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.

3. In order to log in to the system log with the last line from another logfile, use:

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

See also

- The *Using head and tail for printing the last or first 10 lines* recipe of *Chapter 3, File In, File Out*, 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, and 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 the IP address
- Time for which login attempts were performed

Getting ready

We can write a shell script that scans through the logfiles and gather the required information from them. For dealing with SSH login failures, it is useful to know that the user authentication session log is written to the logfile `/var/log/auth.log`. The script should scan the logfile 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.

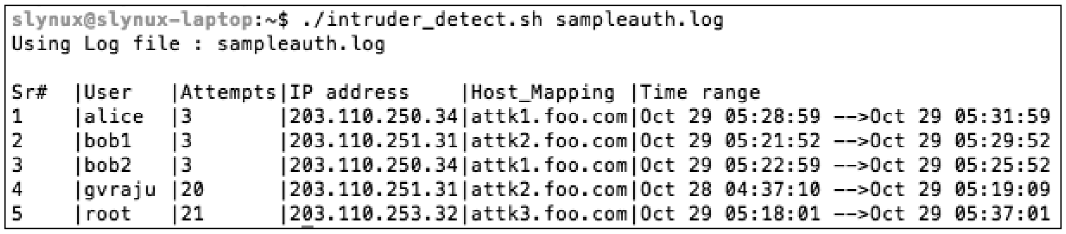
How to do it...

Let's write the intruder detection script that can generate a report to intruders by using a 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 ))

        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 logfile 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 usernames 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 logfile 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. Hence, we should find all the lines with words `Failed password`.

Next, 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 addresses is extracted by using a regular expression for the IP address and the `egrep` command. A `for` loop is used to iterate through the 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 username (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 user. 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 the last log line corresponds to the last attempt. We have used `head -1` to extract the first line and `tail -1` to extract the last line. Now, we have a timestamp 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 *Getting, setting dates, and delays* recipe of *Chapter 1, Shell Something Out*, explains Epoch time).

The variable's start and end has the 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 the 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 hostname mapping can be extracted from the output of the host command by running with the IP address as the argument. The time range can be printed using the timestamp we extracted. Finally, the temporary files used in the script are removed.

The previous 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 a better logic.

Remote disk usage health monitor

A network consists of several machines with different users and 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 like the date, IP address of the machine, device, capacity of 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 as `ALERT`, else it should be set as `SAFE`. This recipe will illustrate how to write a monitoring script that can collect details from remote machines in a network.

Getting ready

We need to collect the disk usage statistics from each machine on the network, individually, and write a logfile in the central machine. A script that collects the details and writes the log can be scheduled to run every day at a particular time. 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 *Password less auto-login with SSH* recipe of *Chapter 7*, *The Old-boy Network*, explains configuration of auto-login). We assume that there is a user 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 $IP_LIST;
do
  #slynux is the username, change as necessary
  ssh slynux@$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 $line | egrep -o "[0-9]+%")
  pusg=${pusg/\%/};
  if [ $pusg -lt 80 ];
  then
    echo SAFE
  else
    echo ALERT
  fi

  done< /tmp/$$.df
done

) >> $logfile
```

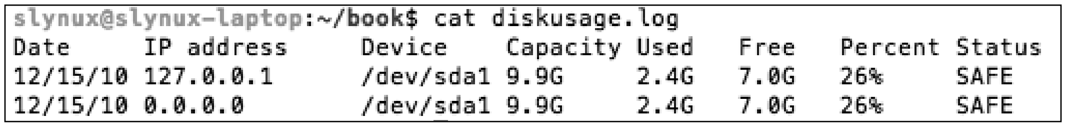
We can use the `cron` utility to run the script at regular intervals. For example, to run the script every day at 10 a.m., write the following entry in the `crontab`:

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

Run the command `crontab -e` and add the preceding line. You can run the script manually as follows:

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

A sample output log for the previous 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 exist, 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 nothing 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.

See also

- The *Scheduling with cron* recipe in *Chapter 9*, *Administration Calls*, explains the `crontab` command

Finding out active user hours on a system

Consider a web server with shared hosting. Many users log in and log out to the server every day and 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 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.$$ > /tmp/user.$$
  minutes=0

  while read t
  do
    s=$(echo $t | awk -F: '{ print ($1 * 60) + $2 }')
    let minutes=minutes+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=$(echo "$minutes / 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    349
2    demoproj   Dec 10    350    230
3    kjayaram   Dec 9     213    55
4    cinenews   Dec 11    85     139
5    thebenga   Dec 10    54     35
6    gateway2   Dec 11    52     34
7    soft132    Dec 12    49     25
8    sarathla   Nov 1     45     29
9    gtsminis   Dec 11    41     26
10   agentcde   Dec 13    39     32
```

How it works...

In the `active_users.sh` script, we can either provide the `wtmp` logfile as a command-line argument or it will use the default `wtmp` log file. The `last -f` command is used to print the logfile contents. The first column in the logfile is the username. 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 in to the session. Hence, in order to find out the total usage hours for a user, the session duration is to be added. The usage duration is in (`HOUR:SEC`) format and it is converted into minutes using a simple awk script.

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 the usage hour string is passed to the standard input for the `while` loop using the `<( COMMANDS )` operator, which acts as a file input. Each hour string is converted into seconds by using the `date` command 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, the `-nr` option is used along with `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.

Measuring and optimizing power usage

Power consumption is one of the factors that one must keep on monitoring, especially on mobile devices, such as notebook computers, tablets, and so on. There are few tools available for Linux systems to measure power consumption, one such command is `powertop` which we are going to use for this recipe.

Getting ready

`powertop` doesn't come preinstalled with most Linux distributions, you will have to install it using your package manager.

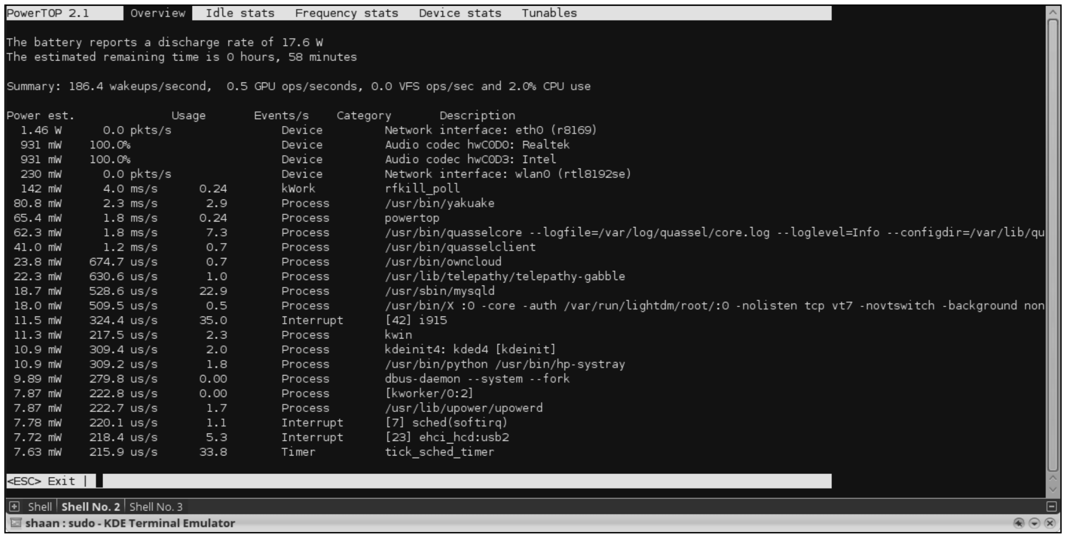
How to do it...

Let's see how to use `powertop` to measure and optimize power consumption:

1. Using `powertop` is pretty easy, just run:

   ```
   # powertop
   ```

   `powertop` will start taking some measurements and once it's done, it will show a screen which will have detailed information about power usage, the processes using the most power, and so on:



```
PowerTOP 2.1      Overview   Idle stats   Frequency stats   Device stats   Tunables

The battery reports a discharge rate of 17.6 W
The estimated remaining time is 0 hours, 58 minutes

Summary: 186.4 wakeups/second,  0.5 GPU ops/seconds, 0.0 VFS ops/sec and 2.0% CPU use

Power est.              Usage       Events/s    Category       Description
  1.46 W      0.0 pkts/s                Device         Network interface: eth0 (r8169)
  931 mW    100.0%                      Device         Audio codec hwC0D0: Realtek
  931 mW    100.0%                      Device         Audio codec hwC0D3: Intel
  230 mW      0.0 pkts/s                Device         Network interface: wlan0 (rtl8192se)
  142 mW      4.0 ms/s       0.24       kWork          rfkill_poll
 80.8 mW      2.3 ms/s        2.9       Process        /usr/bin/yakuake
 65.4 mW      1.8 ms/s       0.24       Process        powertop
 62.3 mW      1.8 ms/s        7.3       Process        /usr/bin/quasselcore --logfile=/var/log/quassel/core.log --loglevel=Info --configdir=/var/lib/qu
 41.0 mW      1.2 ms/s        0.7       Process        /usr/bin/quasselclient
 23.8 mW    674.7 us/s        0.7       Process        /usr/bin/owncloud
 22.3 mW    630.6 us/s        1.0       Process        /usr/lib/telepathy/telepathy-gabble
 18.7 mW    528.6 us/s       22.9       Process        /usr/sbin/mysqld
 18.0 mW    509.5 us/s        0.5       Process        /usr/bin/X :0 -core -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch -background non
 11.5 mW    324.4 us/s       35.0       Interrupt      [42] i915
 11.3 mW    217.5 us/s        2.3       Process        kwin
 10.9 mW    309.4 us/s        2.0       Process        kdeinit4: kded4 [kdeinit]
 10.9 mW    309.2 us/s        1.8       Process        /usr/bin/python /usr/bin/hp-systray
 9.89 mW    279.8 us/s       0.00       Process        dbus-daemon --system --fork
 7.87 mW    222.8 us/s       0.00       Process        [kworker/0:2]
 7.87 mW    222.7 us/s        1.7       Process        /usr/lib/upower/upowerd
 7.78 mW    220.1 us/s        1.1       Interrupt      [7] sched(softirq)
 7.72 mW    218.4 us/s        5.3       Interrupt      [23] ehci_hcd:usb2
 7.63 mW    215.9 us/s       33.8       Timer          tick_sched_timer

<ESC> Exit |
```

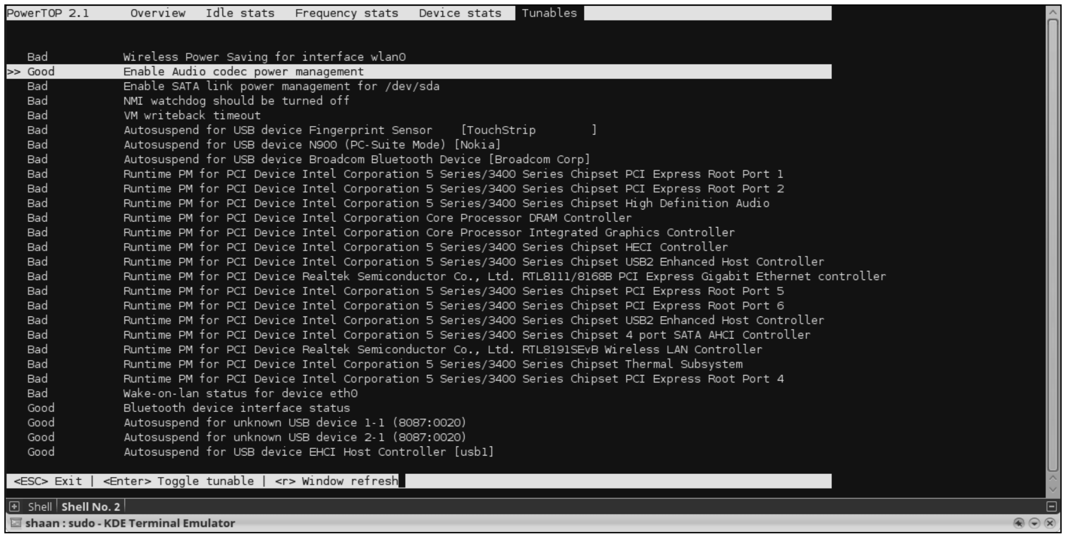
2. For generating HTML reports, use:

   ```
   # powertop --html
   ```

   `powertop` will take measurements over a period of time and generate an HTML report with the default filename `PowerTOP.html`, which you can open using any web browser.

3. For optimizing power usage, use:

   When `powertop` is running, use the arrow keys to switch to the **Tunables** tab; this will show you a list of things that `powertop` can tune so that they consume less power. Just choose whichever ones you want, press *Enter* to toggle from **Bad** to **Good**:



```
PowerTOP 2.1      Overview   Idle stats   Frequency stats   Device stats   Tunables


   Bad           Wireless Power Saving for interface wlan0
>> Good          Enable Audio codec power management
   Bad           Enable SATA link power management for /dev/sda
   Bad           NMI watchdog should be turned off
   Bad           VM writeback timeout
   Bad           Autosuspend for USB device Fingerprint Sensor    [TouchStrip        ]
   Bad           Autosuspend for USB device N900 (PC-Suite Mode) [Nokia]
   Bad           Autosuspend for USB device Broadcom Bluetooth Device [Broadcom Corp]
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset PCI Express Root Port 1
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset PCI Express Root Port 2
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset High Definition Audio
   Bad           Runtime PM for PCI Device Intel Corporation Core Processor DRAM Controller
   Bad           Runtime PM for PCI Device Intel Corporation Core Processor Integrated Graphics Controller
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset HECI Controller
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset USB2 Enhanced Host Controller
   Bad           Runtime PM for PCI Device Realtek Semiconductor Co., Ltd. RTL8111/8168B PCI Express Gigabit Ethernet controller
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset PCI Express Root Port 5
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset PCI Express Root Port 6
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset USB2 Enhanced Host Controller
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset 4 port SATA AHCI Controller
   Bad           Runtime PM for PCI Device Realtek Semiconductor Co., Ltd. RTL8191SEvB Wireless LAN Controller
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset Thermal Subsystem
   Bad           Runtime PM for PCI Device Intel Corporation 5 Series/3400 Series Chipset PCI Express Root Port 4
   Bad           Wake-on-lan status for device eth0
   Good          Bluetooth device interface status
   Good          Autosuspend for unknown USB device 1-1 (8087:0020)
   Good          Autosuspend for unknown USB device 2-1 (8087:0020)
   Good          Autosuspend for USB device EHCI Host Controller [usb1]

 <ESC> Exit | <Enter> Toggle tunable | <r> Window refresh
```

If you want to monitor the power consumption from a portable device's battery, it is required to remove the charger and use the battery for `powertop` to make measurements.

Monitoring disk activity

Going by the popular naming convention of monitoring tools ending in the word `'top'` (the command used to monitor processes), the tool to monitor disk I/O is called `iotop`.

Getting ready

`iotop` doesn't come preinstalled with most Linux distributions, you will have to install it using your package manager.

How to do it...

There are multiple ways of using `iotop` to perform I/O monitoring, some of which we will see in this recipe:

1. For interactive monitoring, use:

   ```
   # iotop -o
   ```

   The `-o` option to `iotop` tells it to show only those processes which are doing active I/O while it is running. It is a useful option to reduce the noise in the output.

2. For non-interactive use from shell scripts, use:

   ```
   # iotop -b -n 2
   ```

   This will tell iotop to print the statistics two times and then exit, which is useful if we want this output in a shell script and do some manipulation on it.

3. Monitor a specific process using the following:

   ```
   # iotop -p PID
   ```

   Put PID of the process that you wish to monitor, and `iotop` will restrict the output to it and show statistics.

> In most modern distros, instead of finding the PID and supplying it to `iotop`, you can use the `pidof` command and write the preceding command as:
>
> ```
> # iotop -p `pidof cp`
> ```

Checking disks and filesystems for errors

Data is the most important thing in any computer system. Naturally, it is important to monitor the consistency of data stored on physical media.

Getting ready

We will use the standard tool, `fsck` to check for errors in the filesystems. This command should be preinstalled on all modern distros. If not, use your package manager to install it.

How to do it...

Let us see how to use `fsck` with its various options to check filesystems for errors, and optionally fix them.

1. To check for errors on a partition or filesystem, just pass its path to `fsck`:

   ```
   # fsck /dev/sdb3
   fsck from util-linux 2.20.1
   e2fsck 1.42.5 (29-Jul-2012)
   HDD2 has been mounted 26 times without being checked, check
   forced.
   Pass 1: Checking inodes, blocks, and sizes
   Pass 2: Checking directory structure
   Pass 3: Checking directory connectivity
   Pass 4: Checking reference counts
   Pass 5: Checking group summary information
   HDD2: 75540/16138240 files (0.7% non-contiguous),
   48756390/64529088 blocks
   ```

2. To check all the filesystems configured in `/etc/fstab`, we can use the following syntax:

   ```
   # fsck -A
   ```

   This will go through the `/etc/fstab` file sequentially, checking each of the filesystems one-by-one. The `fstab` file basically configures a mapping between disks and mount points which makes it easy to mount filesystems. This also makes it possible to mount certain filesystems during boot.

3. Instruct `fsck` to automatically attempt fixing errors, instead of interactively asking us whether or not to repair, we can use this form of `fsck`:

   ```
   # fsck -a /dev/sda2
   ```

4. To simulate the actions, `fsck` is going to perform:

   ```
   # fsck -AN
   fsck from util-linux 2.20.1
   [/sbin/fsck.ext4 (1) -- /] fsck.ext4 /dev/sda8
   [/sbin/fsck.ext4 (1) -- /home] fsck.ext4 /dev/sda7
   [/sbin/fsck.ext3 (1) -- /media/Data] fsck.ext3 /dev/sda6
   ```

   This will print information on what actions will be performed, which is checking all the filesystems.

How it works...

`fsck` is just a frontend for filesystem specific `fsck` programs written for those filesystems. When we run `fsck`, it automatically detects the type of the filesystem and runs the appropriate `fsck.fstype` command where `fstype` is the type of the filesystem. For example, if we run `fsck` on an `ext4` filesystem, it will end up calling the `fsck.ext4` command.

Because of this, you will find that `fsck` itself supports only the common options across all such filesystem-specific tools. To find more detailed options, look at the man pages of specific commands such as `fsck.ext4`.

Further, simulating the actions `fsck` performs is useful when there is a suspicion of a filesystem being corrupt and we run `fsck` to fix it, it is sometimes important to make sure that `fsck` doesn't perform an operation which we don't want. An example maybe that `fsck` might want to mark some sectors as bad, but we might want to try recovering data from it. In this case, we ask `fsck` to just do a dry run and print the actions instead of actually performing the actions.