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.