For example, the following command copies the content of the `test` directory:

The following command copies the `test` directory to the destination:

If / is at the end of `destination_path`, `rsync` will copy the source to the destination directory.

If / is not used at the end of the destination path, `rsync` will create a folder, named similar to the source directory, at the end of the destination path and copy the source into that directory.

For example:

```
$ rsync -av /home/test /home/backups/
$ rsync -av /home/test /home/backups
```

How it works...

`rsync` works with source and destination paths which can be either local or remote. Most importantly, even both the paths can be remote paths. Usually the remote connections are made using SSH so that `rsync` can calculate what files to copy and what not to. Local and remote paths look like this:

- /home/slynux/data (local path)
- slynux@192.168.0.6:/home/backups/data (remote path)

`/home/slynux/data` specifies the absolute path in the machine in which the `rsync` command is executed. `slynux@192.168.0.6:/home/backups/data` specifies that the path is`/home/backups/data` in the machine with IP address `192.168.0.6` and is logged in as user `slynux`.

There's more...

The `rsync` command has several additional functionalities that can be specified using its command-line options. Let's go through them.

Excluding files while archiving with rsync

Some files need not be updated while archiving to a remote location. It is possible to tell `rsync` to exclude certain files from the current operation. Files can be excluded by two options:

```
--exclude PATTERN
```

We can specify a wildcard pattern of files to be excluded. For example:

```
$ rsync -avz /home/code/some_code /mnt/disk/backup/code --exclude "*.txt"
```

This command excludes `.txt` files from backing up.

Or, we can specify a list of files to be excluded by providing a list file.

Use `--exclude-from FILEPATH`.

Deleting non-existent files while updating rsync backup

By default, `rsync` does not remove files from the destination if they no longer exist at the source. In order to remove the files from the destination that do not exist at the source, use the `rsync --delete` option:

```
$ rsync -avz SOURCE DESTINATION --delete
```

Scheduling backups at intervals

You can create a cron job to schedule backups at regular intervals.

A sample is as follows:

```
$ crontab -ev
```

Add the following line:

```
0 */10 * * * rsync -avz /home/code user@IP_ADDRESS:/home/backups
```

The above `crontab` entry schedules the `rsync` to be executed every 10 hours.

`*/10` is the hour position of the `crontab` syntax. `/10` specifies to execute the backup every 10 hours. If `*/10` is written in the minutes position, it will execute every 10 minutes.

Have a look at the *Scheduling with cron* recipe in *Chapter 9*, *Administration Calls* to understand how to configure `crontab`.

Version control-based backup with Git

People use different strategies for backing up data. Out of these, differential backups are more efficient than making copies of the entire source directory to a target of the backup directory with the version number using date or time of a day as it causes wastage of space. We only need to copy the changes that occurred to files from the second time that the backups occur - this is also called incremental backup. We can manually create incremental backups using tools like `rsync` but restoring this sort of backup can be difficult. The best way to maintain and restore changes is to use version control systems. They are very much used in software development and maintenance of code, since coding frequently undergoes changes. Git is the most famous and the most efficient version control system available. Let us use Git for the backup of regular files in a non-programming context.