```
      dest_file="$dest_dir/$dest_file"
    fi

    if [[ -n $scale ]];
    then
      PARAM="-resize $scale"
    elif [[ -n $percent ]];   then
      PARAM="-resize $percent%"
    fi

    echo Processing file : $source_file
    convert $source_file $PARAM $dest_file

  done
```

The following is a sample output, to scale the images in the directory `sample_dir` to `20%` size:

```
$ ./image_help.sh -source sample_dir -percent 20%
Processing file :sample/IMG_4455.JPG
Processing file :sample/IMG_4456.JPG
Processing file :sample/IMG_4457.JPG
Processing file :sample/IMG_4458.JPG
```

In order to scale the images to the width `1024`, use:

```
$ ./image_help.sh -source sample_dir -scale 1024x
```

Change the files to a PNG format by adding `-ext png` along with the preceding commands.

Scale or convert files with a specified destination directory as follows:

```
$ ./image_help.sh -source sample -scale 50% -ext png -dest newdir
# newdir is the new destination directory
```

How it works...

The preceding `image_help.sh` script can accept several command-line arguments, such as `-source`, `-percent`, `-scale`, `-ext`, and `-dest`. A brief explanation of each is as follows:

- The `-source` parameter is used to specify the source directory for the images.
- The `-percent` parameter is used to specify the scale percent and `-scale` is used to specify the scale width and height.
- Either `-percent` or `-scale` is used. Both of them do not appear simultaneously.

- The `-ext` parameter is used to specify the target file format. `-ext` is optional; if it is not specified, format conversion is not performed.
- The `-dest` parameter is used to specify the destination directory for scale or conversion of image files. `-dest` is optional. If `-dest` is not specified, the destination directory will be the same as the source directory. As the first step in the script, it checks whether the number of command arguments given to the script are correct. Either four, six, or eight parameters can appear.

Now, by using a `while` loop and case statement, we parse the command-line arguments corresponding to variables. `$#` is a special variable that returns the number of arguments. The `shift` command shifts the command arguments one position to the left, so that on each execution of shift, we can access command arguments one by one, by using the same `$1` variable rather than using `$1`, `$2`, `$3`, and so on. The case statement matches the value of `$1`; it is like a switch statement in the C programming language. When a case is matched, the corresponding statements are executed. Each match case statement is terminated with `;;`. Once all the parameters are parsed in variables `percent`, `scale`, `source_dir`, `ext`, and `dest_dir`, a `for` loop is used to iterate through the path of each file in the source directory and the corresponding action to convert the file is performed.

If the variable `ext` is defined (if `-ext` is given in the command argument), the extension of the destination file is changed from `source_file.extension` to `source_file.$ext`. In the next statement, it checks whether the `-dest` parameter is provided. If the destination directory is specified, the destination file path is crafted by replacing the directory in the source path with the destination directory by using filename slicing. In the next statement, it crafts the parameter to the convert command for performing resize (`-resize widthx` or `-resize perc%`). After the parameters are crafted, the `convert` command is executed with proper arguments.

See also

- The *Slicing filenames based on extension* recipe of *Chapter 2*, *Have a Good Command*, explains how to extract a portion of the filename

Taking screenshots from the terminal

Taking screenshots is another common day-to-day activity for any computer user. It becomes even more important for administrators that maintain GUI applications running on computers and automate them. It is important to take screenshots when a particular event happens, to figure out what is going on in a GUI application.