Performing Essential System Administration Tasks

So you have your server up and running, and you've just learned how to get your work done from the command line. This is where the real work starts! Next, you need to learn how to tune your server so it does exactly what you want it to. First, you need to know how to manage software. Next, even if many in the Linux community will flame you for it, you probably want to work with a graphical interface on your server to accomplish common tasks. Even if Ubuntu Server is a command line–oriented server, in some situations the graphical interface just makes things much easier. So I’ll explain how to install that at your server. Once the server starts to take shape, you’ll want to make sure that it is properly backed up. And, finally, if something goes wrong, you’ll need logging to find out what happened. All these are considered essential system administration tasks, and you’ll learn about them in this chapter.

Software Management

As on any other computer, you’ll need to install software on Ubuntu Server on a regular basis. You can approach software installations in two ways. First and most important are the software packages containing programs that are ready to install and that integrate easily with Ubuntu Server. The server keeps a list of all software packages that are installed, which makes managing them much easier. The second approach to software installation is the tarball, which is basically just an archive that contains files. These files can be really anything (for example, a backup of your server’s data is stored in a tarball), but the tarball can also be used to deliver software to install.

You should be aware of two important differences between the two methods of software installation. One is that your server keeps track of everything that is installed only if that software is installed from a package. Software installed from tarballs is not tracked. The second difference between tarballs and packages is that some software needs other software to be present before it can be installed. (This is called a dependency.) Imagine a graphical application that would need a graphical user interface (GUI) to be present before you can use it. Both the tarball and the software package have installation programs that can check if all dependencies have been met, but only the software package interacts via the package manager software with a database of packages that are installed and packages that are available at your server. Because of this interaction, the package manager can install missing dependencies for
you automatically, and this is why, on modern Linux distributions, software packages are preferred over tarballs.

Currently, software packages can be created with two different formats. On Red Hat, SUSE, and many similar Linux distributions, the RPM Package Manager (RPM) format is the way to go. Ubuntu and Debian, however, currently use the Debian Package (DEB) format. Although the formats can be converted, you typically don't want to install RPM on Ubuntu Server, and the available management utilities will check for DEB by default.

Software Repositories and Package Databases

To understand a Linux package manager, you need to know about software repositories. A software repository can be considered a source of installation for software. On your server, a list of all these installation sources is kept in the file /etc/apt/sources.list. As an administrator, it is important to be aware of this list. Although the most important software repositories are added to this file automatically, you may occasionally want to add other software repositories to this list.

In all repositories, you’ll always find the following five package categories:

- **main**: The main category portion of the software repository contains software that is officially supported by Canonical, the company behind Ubuntu. The software that is normally installed to your server is in this category. By working with only this software, you can make sure that your system remains as stable as possible and—very important for an enterprise environment—that you can get support for it at all times.

- **restricted**: The restricted category is basically for supported software that uses a license that is not freely available, such as drivers for specific hardware components that use a specific license agreement, or software that you have to purchase. You'll typically find restricted software in a specific subdirectory on the installation media.

- **universe**: The universe category contains free software that is not officially supported. You can use it and it is likely to work without problems, but you won’t be able to get support from Canonical for software components in this category.

- **multiverse**: The multiverse component contains unsupported software that falls under license restrictions that are not considered free.

- **backports**: In this category, you’ll find bleeding-edge software. If you want to work with the latest software available, you should definitely get it here.

When installing software with the `apt-get` utility, it will look for installation sources in the configuration file /etc/apt/sources.list. Listing 3-1 shows a part of its contents.

Listing 3-1. Definition of Installation Sources in sources.list

deb http://security.ubuntu.com/ubuntu feisty-security main restricted
deb-src http://security.ubuntu.com/ubuntu feisty-security main restricted
deb http://security.ubuntu.com/ubuntu feisty-security universe
deb-src http://security.ubuntu.com/ubuntu feisty-security universe
deb http://security.ubuntu.com/ubuntu feisty-security multiverse
deb-src http://security.ubuntu.com/ubuntu feisty-security multiverse
As you can see, the same format is used in all lines of the sources.list file. The first field in these lines specifies the package format to be used. Two different package formats are used by default: deb for binary packages (basically precompiled program files) and deb-src for packages in source file format. Next, the Universal Resource Identifier (URI) is mentioned. This typically is an HTTP or FTP URL, but it can be something else as well. For instance, it can refer to installation files that you have on an installation CD or in a directory on your server. After that you will see the name of the Ubuntu Server distribution that is used, and you will always see the current server version there. Last, every line refers to the available package categories. As you can see, most package categories are in the list by default. Only installation sources for security patches have been included in the partial listing of sources in Listing 3-1. For a complete overview, take a look at the configuration file itself.

Now that you understand how the sources.list file is organized, it follows almost automatically what should happen if you want to add some additional installation sources to this list: make sure that all required components are specified and add any line you like, referring to an additional installation source. Once an additional installation source has been added, it will be automatically checked when working on software packages. For example, if you should use the apt-get update command to update the current state of your system, the package manager will check your new installation sources as well.

A second important management component used by package managers on your server is the package database. The most fundamental package database is the dpkg database, which is managed by the Debian utility dpkg. On Ubuntu, however, the Advanced Packaging Tools (apt) set is used for package management. These tools add functionality to package management that the traditional dpkg approach typically cannot offer. Because of this added functionality, the apt tools use their own database, which is stored in /var/lib/apt. By communicating with this database, the package manager can query the system for installed software, and this enables your server to automatically solve package-dependency problems.

Every time a package is installed, a list of all installed files is added to the package database. By using this database, the package manager can even see if certain configuration files have been changed, which is very important if you want to update packages at your server!

---

**Caution** Because working with two different package management databases can be confusing, I suggest that you choose the package management system that you want to work with and stick to it. In this book, I will cover only the apt utilities.

---

**Package Management Utilities**

You can use any of several package management utilities on Ubuntu Server. The most important of these interact directly with the package database in /var/lib/apt. You would typically use the apt-get command for installation, updates, and removal of packages, and so you’ll find yourself working with that utility most of the time. You should also know of the aptitude utility, which works in two ways. You can use aptitude as a command-line utility to query your server for installed packages, but aptitude also has a menu-driven interface that offers an intuitive way to manage packages. If this still isn’t easy enough, you can use the graphical
utility Synaptic as an alternative. Before you can use that, though, you need to install a GUI. You can read more about that later in this chapter.

Another approach to managing packages is the Debian way. Because Ubuntu package management is based on Debian package management, you can use Debian package management tools like dpkg as well. However, these do not really add anything to what Ubuntu package management already offers, and so I will not cover the Debian tools in this book.

Understanding apt

Before you start working on packages on Ubuntu Server, it is a good idea to decide what tool you want to use. It's a good idea because many tools are available for Ubuntu Server and each of them uses its own database to keep track of everything installed. To prevent inconsistencies in software packages, it's best to choose your favorite utility and stick to that. In this book I'll focus on the apt-get utility, which keeps its database in the /var/lib/apt directory. This is my favorite utility because you can run apt-get as a very easy and convenient tool from the command line to perform tasks very quickly. The apt-get utility works with commands that are used as its argument, such as apt-get install something. In this example, install is the command you use to tell apt-get what you really want to do. The following four package management commands are the most important building blocks when working with apt-get:

- update: This is the first command you want to use when working with apt-get. It updates the list of packages that are available for installation. Use it to make sure that you install the most recent version of a package.
- upgrade: Use this command to perform an upgrade of your server's software packages.
- install: This is the command you want to use every time you install software. It's rather intuitive. For example, if you want to install the Xen software package, you would just type apt-get install xen.
- remove: You've probably guessed already, but you'll use this one to remove installed packages from your server.

Showing a List of Installed Packages

Before you start managing packages on Ubuntu Server, you probably want to know what packages are already installed, and you can do this by issuing the dpkg -l command. It'll generate a long list of installed packages. Listing 3-2 shows a partial result of this command.

**Note** The apt-get utility is not the most appropriate way to list installed packages because it can see only those packages that are installed with apt. If you have installed a package with dpkg (which I would not recommend), you won't see it with apt-get. So, to make sure that you don't miss any packages, I recommend using dpkg -l to get a list of all installed packages.
Listing 3-2. The dpkg -l Command Shows Information About Installed Packages.

$ dpkg -l
ii  xvidtune       1.0.1-0ubuntu1 X client - xvidtune
ii  xvinfo         1.0.1-0ubuntu1 XVideo information
ii  xwd            1.0.1-0ubuntu1 X client - xwd
ii  xwininfo       1.0.1-0ubuntu1 X client - xwininfo
ii  xwud           1.0.1-0ubuntu1 X client - xwud
ii  yelp           2.18.1-0ubuntu Help browser for GNOME 2
ii  zenity         2.18.1-0ubuntu Display graphical dialog boxes from shell sc
ii  zip            2.32-1 Archiver for .zip files
ii  zlib1g         1.2.3-13ubuntu compression library - runtime
ii  zlib1g-dev     1.2.3-13ubuntu compression library - development

The result of the dpkg command shows information about packages and their status. The first character of the package shows the desired status for a package, and this status indicates what should happen to the package. The following options are available for this status indicator:

- **i**: You’ll see this option in most cases, indicating that the package should be installed.
- **h**: This option (for “hold”) indicates that the package cannot be modified.
- **p**: This option indicates that the package should be purged.
- **r**: This option indicates that the package is supposed to be removed without removing associated configuration files.
- **u**: This option indicates that the current desired status is unknown.

The second character reveals the actual state of the package. You’ll find the following options:

- **I**: The package is installed.
- **c**: Configuration files of the package are installed, but the package itself is not.
- **f**: The package is not guaranteed to be correctly installed.
- **h**: The package is partially installed.
- **n**: The package is not installed.
- **u**: The package did install, but the installation was not finalized because the configuration script was not successfully completed.

The third character indicates any known error state associated with the package. In most cases you’ll just see a space (so, basically, you don’t see anything at all), indicating that nothing is wrong. Other options are as follows:

- **H**: The package is put on hold by the package management system. This means that dependency problems were encountered, in which case some required packages are not installed.
R: Reinstallation of the package is required.
X: The package requires reinstallation and has been put on hold.

The `dpkg` command can be used to show a list of packages that are already installed in your system, but you can also use it to display a list of packages that are available to your system. The only difference is that you have to provide some information about the package. For example, the command `dpkg -l "samba*"` would provide information about the current installation status of the Samba package. Listing 3-3 shows the result of this command.

**Listing 3-3. Dpkg Can Be Used to Display a List of Packages That Are Available.**

```
sander@RNA:~$ dpkg -l "samba*"
```

```
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
|||/ Name           Version        Description
+++-==============-==============-============================================
un  samba-common   <none>         (no description available)
```

As you can see in the output that is provided for each package, the first two positions show that the package status is currently unknown. In combination with some smart use of the `grep` command, you can even use this construction to find out what packages are available for installation on your server. In the command `dpkg -l "*" | grep ^un`, the `grep` command is used to filter out all packages that show a result that starts with the letters "un," which is very typical for a package that is not installed.

You can also use the `dpkg` utility to find out what package owns a certain file. This is very useful information. Imagine that a file is broken and you need to refresh the package's installation. To find out what package owns a file, use `dpkg --search /your/file`. The command will immediately return the name of the package that owns this file.

**Using aptitude**

On Ubuntu, a few commands are available for package management. One of these is `aptitude`. The major benefit of this command is that it is somewhat more user friendly because it can work with keywords, which are words that occur somewhere in the description of the package. For example, to get a list of all packages that have "xen" (the name of the well-known Linux virtualization product) in their description, you would use `aptitude search xen`. Listing 3-4 shows the result of this command.

**Listing 3-4. Showing Package Status Based on Keywords**

```
sander@RNA:~$ aptitude search xen
```

```
p   ubuntu-xen-desktop         - Xen software for running on servers.
p   ubuntu-xen-server          - Xen software for running on servers.
p   xen-doc-2.6.16              - Linux kernel specific documentation
p   xen-docs-3.0                - documentation for XEN, a Virtual Mac
v   xen-headers                -
```

CHAPTER 3 ■ PERFORMING ESSENTIAL SYSTEM ADMINISTRATION TASKS

52
Once you have found a package using the `aptitude` command, you can also use it to show information about the package. To do this, you’ll use the `show` argument. For example, `aptitude show xen-source` will show you exactly what the package `xen-source` is all about (see Listing 3-5). As you can see, in some cases very useful information is displayed.

**Listing 3-5. The `aptitude` show Command Shows What Is Offered by a Package.**

```
sander@RNA:~$ aptitude show xen-source
No current or candidate version found for xen-source
Package: xen-source
State: not a real package
Provided by: xen-source-2.6.16
sander@RNA:~$ aptitude show xen-source-2.6.16
Package: xen-source-2.6.16
State: not installed
Version: 2.6.16-11.1
Priority: optional
Section: universe/devel
Maintainer: Chuck Short <zulcss@ubuntu.com>
Uncompressed Size: 42.6M
```
Depends: binutils, bzip2, coreutils | fileutils (>= 4.0)
Recommends: libc-dev, gcc, make
Suggests: libncurses-dev | ncurses-dev, kernel-package, libqt3-dev
Provides: xen-source, xen-source-2.6
Description: Linux kernel source for version 2.6.17 with Ubuntu patches
This package provides the source code for the Linux kernel version 2.6.17.

You may configure the kernel to your setup by typing "make config" and following instructions, but you could get ncursesX.X-dev and try "make menuconfig" for a jazzier, and easier to use interface. There are options to use QT or GNOME based configuration interfaces, but they need additional packages to be installed. Also, please read the detailed documentation in the file /usr/share/doc/linux-source-2.6.17/README.headers.gz.

If you wish to use this package to create a custom Linux kernel, then it is suggested that you investigate the package kernel-package, which has been designed to ease the task of creating kernel image packages.

If you are simply trying to build third-party modules for your kernel, you do not want this package. Install the appropriate linux-headers package instead.

Adding and Removing Software with apt-get

The best tool to perform package management from the command line is apt-get. It provides a very convenient way to install, update, or remove software packages on your machine. It requires root permissions, so you should always start the command with sudo.

Before you do anything with apt-get, you should always use the apt-get update command first. Because apt-get gets most software packages online, it should always know about the latest available versions of those packages. The apt-get update command makes sure of this, and it caches a list of the most recent version of packages that are available on your server. Once the update is performed, you can use apt-get to install and remove software. Installation is rather easy: to install the package blah, use apt-get install blah. The advantage of the apt-get command is that it really tries to understand what you are doing. This is shown in Listing 3-6, where the apt-get command is used to install the Xen virtualization software.

Listing 3-6. The apt-get Command Tries to Understand What You Want to Do.
sander@RNA:~$ sudo apt-get install xen
Password:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Package xen is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source
However the following packages replace it:
  xen-utils-common
E: Package xen has no installation candidate

As you can see from this example, the apt-get command does have a problem in understanding what you mean by "xen." However, it does note that another package refers to Xen, and, as a result, it asks you if you want to install this other package. If you want to do that, just run apt-get again, referring to the name of this other package. Even if this works to install packages on your machine, you should always be aware that apt-get may miss the point here. So always remain alert and check if it has really installed the packages you needed.

You can also use apt-get to remove software, upgrade your system, and much more. The following list provides an overview of the most important functions of the apt-get command. Be aware that you should always run the command with root permissions, so use sudo to start apt-get (or set a root password and work as root directly).

- **Install software**: Use `sudo apt-get install package`.
- **Remove software**: Use `sudo apt-get remove package`. This option does not remove configuration files. If you need to remove those as well, use `sudo apt-get remove --purge package`.
- **Upgrade software**: To upgrade your complete operating system, use `sudo apt-get update` first so that you're sure that apt-get is aware of the most recent version of the packages. Then use `sudo apt-get dist-upgrade`.

### Making Software Management Easy with Synaptic

I know, Ubuntu Server is not supposed to be a graphical operating system, but, as you'll see, it is perfectly possible—and sometimes even preferable—to install a graphical system. A GUI makes administering your Ubuntu Server a lot easier. One of the tools that come with the graphical interface is the Synaptic package manager. As you can see in Figure 3-1, it offers a very intuitive interface to help you install and manage software packages.

In Synaptic, the Sections button is a good starting point, because clicking it allows you to see all available software, organized by software category. To see what's inside a category, click it and a list of available packages will be displayed in the right part of the Synaptic window. Clicking an individual package will provide a description of the package, allowing you to see exactly what is in it. Next, select the Mark for Installation option and click Apply. You'll then see the window in Figure 3-2, asking you if you really want to install this package. Click Apply to start the installation.
Figure 3-1. The Synaptic package management tool really makes software management easy.

Figure 3-2. Click Apply to start installation of the selected package.
Another very useful option from the Synaptic interface is the Search feature. Click Search and, from the window that’s displayed, select the software you are looking for. Click Search again and you’ll see a list with all matching packages. If you want to use these packages, mark these for installation and click Apply.

**Installing Software from Tarballs**

Most software for Ubuntu is available from the normal Ubuntu installation channels. Sometimes, however, you’ll encounter software in other formats, such as source files that are delivered in the `.tar.gz` format. These packages have been archived with the `tar` utility and then compressed with `gzip`, and so they’re known as **tarballs**.

Extraction will reveal that the tarball contains one of two types of files: source files and binary files. If the tarball contains binary files, it’s normally enough to run the installation program and install them. Just look at the name of the files in the tarball, and you’ll probably immediately see what you have to run to perform the installation. If the tarball contains source files, you first have to compile them.

Before starting to install software by compiling its source files, you need to be aware of something. Although you’ll probably end up with perfectly working software, all of the software that you install in this way is **unmanaged**. This means that it will not be updated when you update everything else on your server, simply because the software is not in the databases maintained by software management programs such as `apt-get`. Therefore, I always recommend that you try to install software using the regular Ubuntu software installation methods first. If that doesn’t work (and only then), use the method described next.

---

**Caution** The procedure described here works in many cases, but it doesn’t work all the time for the simple reason that it is all dependent on the person who created the package. I always recommend that you read the `readme` file that comes with most source files to see if the software installation has any specific instructions or requirements.

---

1. Before starting to compile the source files, you need to make sure that the required compiler is installed on your server. The command `dpkg -l | grep gcc` would be an excellent choice to do that. If you don’t see the gcc compiler, use `apt-get install gcc` to install if before you proceed.

---

**Tip** To describe this procedure, I’ve downloaded the latest version of nmap from [http://insecure.org/nmap/download.html](http://insecure.org/nmap/download.html). If you want to follow this procedure, download this file as well.

---

2. Once you have downloaded the software you want to install into your home directory, check how the file is compressed. If the file has the `.b2` extension, it has been compressed with the `bzip2` utility. To uncompress it, you need the `tar` command-line switch `-j`. If the file has the `.gz` extension, it has been compressed with the `gzip` utility.
and the tar utility needs the `-z` switch to extract it. Our example file is compressed with bzip2, so run the `tar -jxvf nmap*` command to extract the archive. In this command, the option `x` is used to extract the tar archive, the option `v` does that in a verbose way so that you'll see what happens, and the option `f nmap*` specifies that the name of the file you want to extract is anything that starts with "nmap." This creates a subdirectory in your current directory in which all source files are installed. Now activate this subdirectory with the `cd` command.

3. From the directory that was created while extracting the tarball, run the `./configure` command. This command will verify that everything required to install the selected software is present on your server. If the utility fails, it is usually because some required software component was not installed. If this is the case, you'll see an error message stating what exactly is missing. Read what software component that is, and install it before you proceed. When `./configure` runs without errors, continue with the next step.

4. Compiling software is a lot of work and involves very complex commands. However, the `make` utility is available to make the compiling process easier. This utility reads a file with the name `Makefile` that has to be present in the directory with the source files; based on the instructions in that file, it compiles the software. Depending on the software that you want to install, the compiling process can take a long time. Once it's finished, though, continue with the next step.

5. You should now have all the program files that you need. But you're not quite done, because you still have to make sure that these files are copied to the appropriate paths on your server. To do this, you must run the `make install` command as root. Type `sudo make install` and press Enter. This completes the installation of the source files for your machine, and they're ready for use.

**Configuring a Graphical User Interface**

Using a GUI on a server can be a very sensitive subject for some people. Some system administrators feel that the GUI is merely a waste of system resources and that there is no need for it on a professional server.

These people are basically right, but doing everything at the command line can be quite a chore if you have no Linux experience. Personally, I believe in freedom of choice. You want to use a graphical interface to get familiar with Ubuntu Server? That's fine with me. However, if you are an experienced Linux server administrator and you don't want to waste system resources on a useless graphical interface, please skip ahead to the section “Creating Backups.” That's fine with me as well. Do you still hesitate whether or not to install a GUI? If so, Table 3-1 lists some advantages and disadvantages for you to consider.

**Table 3-1. GUI Advantages and Disadvantages**

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makes administration easier</td>
<td>Security risks</td>
</tr>
<tr>
<td></td>
<td>Slows down your server</td>
</tr>
<tr>
<td></td>
<td>GUIs are often rather limited</td>
</tr>
</tbody>
</table>
Are you still with me? Good, because that means you want to install a GUI. As you saw in
the preceding section, installing software is easy with Ubuntu Server. When installing a graph-
ic interface, however, you need to make some choices, the first of which is what kind of
graphical interface you want to use. You basically have two different options: the window
manager and the desktop environment.

In general, a window manager is a lightweight graphical interface that manages windows
on your server, and a desktop environment is a complete graphical workspace that not only
creates windows for you, but also offers a wide range of applications. If you have worked with
Ubuntu on the desktop, you are probably familiar with the GNOME desktop manager, which
is the default graphical user environment for the desktop.

Mainly because you probably want to install a graphical desktop to make managing
Ubuntu Server easier, in this section you’ll learn how to set up the GNOME desktop. I chose
 GNOME because it is the most complete graphical desktop environment available for Linux.
Installing it is rather easy with apt-get, you just have to know what to install. To install every-
thing that is needed, enter the following command:

    sudo apt-get install xserver-xorg xfonts* gnome

This command makes sure that all required software is copied to your system. Some of
the software has to be downloaded from the Internet, and it can take a while before everything
is installed. After that, you will have a complete graphical user environment, like the one
shown in Figure 3-3.

**Figure 3-3.** It doesn’t come by default, but you certainly can manage Ubuntu Server from a
graphical interface.
Creating Backups

One thing always seems to be true about computers: one day they’ll fail. If the computer in question is a server, the failure can cause huge problems. Companies have gone bankrupt because their vital data was lost. Therefore, making decent backups of your data is essential. In this section, I’ll cover three different methods of creating backups, all of which are native Linux solutions. Apart from these solutions, quite a few commercial backup solutions are available that fit into the backup infrastructure that is often used at the enterprise level in a company. Those solutions are very specific, and I do not include them in this book. I’ll discuss two backup solutions: making file backups with `tar`, and making device backups using `dd`.

Making File Backups with `tar`

The command-line utility `tar` is probably the most popular Linux backup utility. It functions as a stand-alone utility to write backups to an archive. This archive can be tape (hence the name `tar` which stands for `tape archiver`), but it can also be anything else. For instance, `tar`-based backups are often written to a file instead of a tape, and, if this file is compressed with a compression utility like `bzip2` or `gzip`, you’ll get the famous tarball, which is a common method to deliver software installation archives. In this section, you’ll learn how to create `tar` archives and how to extract files from them. I’ll also provide some tips and tricks to help you get the most out of the `tar` utility.

Creating an Archive File

In its most basic form, `tar` is used to create an archive file. The typical command to do so is `tar -cvf somefile /somedirectory`. This `tar` command has a few arguments. First, you need to indicate what you want to do with the `tar` command. In this case, you want to create an archive. (That’s why the option `c` is used; the “c” stands for `create`.)

After that, I’ve used the option `v` (verbose). Although it’s not required, it often comes in handy because verbose output lets you see what the `tar` command is actually doing. I recommend always using this option because sometimes a `tar` job can take a really long time. (For instance, imagine creating a complete archive of everything that’s on your hard drive.) In cases such as these, it’s nice to be able to monitor what exactly happens and that’s what the option `v` is meant to do.

Next, you need to specify where you want the `tar` command to send its output. If you don’t specify anything here, `tar` defaults to the standard output (STDOUT). In other words, it
simply dumps all the data to your server’s console. This doesn’t accomplish much, so you should use the option `f` (file) to specify what file or device the output should be written to.

In this example I’ve written the output to a file, but, alternatively, you can write output to a device file as well. For example, the command `tar -cvf /dev/mt0 /somedir` will write the result of the `tar` command to the `/dev/mt0` device, which typically is your tape drive.

The last part of the `tar` command specifies exactly what you want to put into your `tar` archive. In the example, the directory `/somedir` is archived. It’s easy to forget this option, but, if you do, `tar` will complain that it is “cowardly refusing to create an empty archive.”

And you should know a couple of other things about `tar`. First, the order of arguments `does` matter. So there is a difference between `tar -cvf /somefile /somedir` and, for example, `tar -f /somefile -vc /somedir`. The order is wrong in the last part, and `tar` won’t know what you want it to do. So, in all cases, first specify what you want `tar` to do. In most cases, it’s either `c` (to create an archive), `x` (to extract an archive), or `t` (to list the contents of the archive). Then specify how you want `tar` to do that; for example, you can use `v` to tell `tar` that it should be verbose. Next, use the `f` option to indicate where you want `tar` to write the backup, and then specify what exactly you want to back up.

Creating an archive with `tar` is useful, but you should be aware that `tar` doesn’t compress one single bit of your archive. This is because `tar` was originally conceived as a tape streaming utility. It streams data to a file or (typically) a tape device. If you want `tar` to compress the contents of an archive as well, you must tell it to do so. And so `tar` has two options to compress the archive file:

- `z`: Use this option to compress the `tar` file with the `gzip` utility. This is the most popular compression utility, because it has a pretty decent compression ratio and it doesn’t take too long to create a compressed file.

- `j`: Use this option to compress the `tar` file with the `bzip2` utility. This utility compresses 10 to 20 percent better than `gzip`, but at a cost: it takes as twice as long.

So, if you want to create a compressed archive of the directory `/home` and write that backup to a file with the name `home.tar.gz`, you would use the following command:

```
tar -czvf home.tar.gz /home
```

**Note** Of course you can use the `bzip2` and `gzip` utilities from the command line as well. Use `gzip file.tar` to compress `file.tar`. This command produces `file.tar.gz` as its result. To decompress that file, use `gunzip file.tar.gz`, which gives you the original `file.tar` back. If you want to do the same with `bzip2`, use `bzip2 file.tar` to create the compressed file. This creates a file with the name `file.tar.bz2`, which you can decompress using the command `bunzip2 file.tar.bz2`.

### Extracting an Archive File

Now that you know how to create an archive file, it’s rather easy to extract it. Basically, the command-line options that you use to extract an archive file look a lot like the ones you
needed to create it in the first place. The important difference is that, to extract a file, you need
the option x (extract), instead of c (create). Here are some examples:

- `tar -xvf /file.tar`: Extracts the contents of file.tar to the current directory.
- `tar -zxvf /file.tar.gz`: Extracts the contents of the compressed file.tar to the
current directory.
- `tar -xvf /file.tar C /somedir`: Extracts the contents of /file.tar to a directory with
the name /somedir.

### Moving a Complete Directory

Most of the time, tar is used to write a backup of one or more directories to a file. Because of
its excellent handling of special files (such as stale files that are used quite often in databases),
tar is also quite often used to move the contents of one directory to another. Some people
perform this task by first creating a temporary file and then extracting the temporary file into
the new directory. This is not the easiest way because you need twice the disk space taken by
the directory whose contents you want to move: the size of the original directory plus the
space needed for the temporary file. The good news is that you don’t have to do it this way.
Use a pipe, and you can directly blow the contents of one directory to another directory.

To understand how this works, first try the command `tar -cC /var`. In this command, the
option C is used to tell tar that it should create an archive. The option C is used to archive the
contents of the directory /var and not the complete directory. This means that, in the archive
itself, you won’t see the original directory name /var. So, if there’s a file called /var/blah, you
will see blah in the archive, and not /var/blah, which would have been the case if you omitted
the option C (a leading / is always stripped from the pathname in a tar archive). Now, as you
may have noticed, in the `tar -cC /var` example, the option f /somefile.tar isn’t used to
specify where the output goes, and so all the output is sent to STDOUT, which is your console.

So that’s the first half of the command, and you ended up with a lot of output dumped
on the console. Now, in the second part of the command, you’ll use a pipe to redirect all that
output to another command, which is `tar -xC /newvar`. This command will capture the tar
archive from STDOUT and extract it to the directory /newvar (make sure that newvar exists
before you run this command). You’ll see that this method allows you to create a perfect copy of
one directory to another. So the complete command that you need in this case looks like this:

```bash
tar -cC /var . | tar -vxC /newvar
```

### Creating Incremental Backups

Based on the information in the previous section, you can probably see how to create a
backup of one or more directories. For instance, the `tar -cvf /backup.tar /var /home /srv` command creates a backup of three directories: /home, /var, and /srv. Depending on the size
of these directories, this command may take some time. Because such large backups can take
so long, it’s often useful to make incremental backups, which is a backup in which the only
files that get written to the backup are those that have changed since the last backup. To do
this, you need the option g to create a snapshot file.
An incremental backup always follows a full backup, and so you have to create the full backup first. In this full backup, you should create a snapshot file, which contains a list of all files that have been written to the backup. The following command would do that for you (make sure that /backup exists before running the command):

```
tar -czvg /backup/snapshot-file -f /backup/full-backup.tar.gz /home
```

The interesting thing about the snapshot file is that it contains a list of all files that have been written to the backup. If, two days after the full backup, you want to make a backup of only the files that have been changed in those two days, you can repeat essentially the same command. This time, the command will check the snapshot file to find out what files have changed since the last full backup, and it'll back up only those changed files. So your Monday backup would be created by the following command:

```
tar -czvg /backup/snapshot-file -f /backup/monday-backup.tar.gz /home
```

These two commands created two files: a small file that contains the incremental backup, and a large file that contains the full backup. Now, if you want to restore all files from backup, you need to restore every single file, starting with the first file that was created (typically the full backup) and ending with the last incremental backup. So, in this example, the following two commands would restore the file system back to the status at the time that the last incremental backup was created:

```
tar -xzvf /backup/full-backup.tar.gz
```

Making Device Backups Using `dd`

You won't find a more versatile utility than `tar` to create a file system-based backup. In some cases, however, you don't need a backup based on a file system; instead, you want to create a backup of a complete device, or parts of it. This is where the `dd` command comes in handy. The basic use of the `dd` command is rather easy because it takes just two arguments: `if=` to specify the input file, and `of=` to specify the output file. The arguments to those options can be either files or block devices. So, the command `dd if=/etc/hosts of=/home/somefile` can be used as a complicated way to copy a file.

**Note** `dd` is, strangely enough, short for “convert and copy.” Unfortunately, the `cc` command was already being used by something else and the developers choose to use `dd` instead.

More interesting is the use of `dd` to copy a complete device. What would you think, for example, of the command `dd if=/dev/cdrom of=/mycd.iso`? It would help you create an ISO file of the CD-ROM that’s in the drive at that moment.

You may wonder why not just copy the contents of your CD-ROM to a file with the name `/mycd.iso`. Well, the reason is, a CD-ROM, like most other devices, typically contains information that cannot be copied by a mere file copy. For example, how would you handle the boot
sector of a CD-ROM? You can't find that as a file on the device because it's just the first sector. Because dd copies sector by sector, on the other hand, it will copy that information as well.

**Tip** Did you know that it's not hard to mount an ISO file that you created with dd? The only thing that you need to know is that you have to use the -o loop option, which allows you to mount a file like any normal device. So, to mount /mycd.iso on the /mnt directory, you would need sudo mount -o loop /mycd.iso /mnt.

Making a backup of a CD-ROM with dd is one option. And any other similar device can be copied as well. How would you go about making a complete copy of your entire hard disk? It's easy, but I recommend that you first boot your server using the rescue option that you can find on the installation CD. Doing this gives you a complete Linux system that doesn't use any of the files on your server's hard disk, which ensures that no files are in use at that moment. Before you start, make sure you know what device is used by your server's hard drive. The best way to find out is by using the sudo fdisk -l command, which provides a list of all partitions found on your server, with the local hard disk coming first.

In most cases, the name of your hard drive will be /dev/sda, but it may be /dev/hda or something completely different. Let's assume that your server's hard drive is /dev/sda, and you now have to attach a second hard drive to your server. Typically, this second drive would be known as /dev/sdb. Next, you can use the dd command to clone everything from /dev/sda to /dev/sdb: dd if=/dev/sda of=/dev/sdb. This command takes quite some time to complete, and it also wipes everything that currently exists on /dev/sdb, replacing it with the contents of /dev/sda. Unfortunately, it often takes several hours to dd everything from one hard disk to another.

### Configuring Logging

The last essential system administration task covered in this chapter is logging. It's obviously very important to understand where certain information is recorded on your server. Knowing this helps you troubleshoot when something doesn't work out the way you expect. Also, understanding how logging works may help prevent your entire server from filling up with log files. On Ubuntu Server, syslog is used to configure logging. You'll learn now how to configure it and where its associated log files are written.

#### Configuring syslog

Logging on to Ubuntu Server is handled by the syslogd process. The process reads its configuration file /etc/syslog.conf and based on the instructions it finds there, it determines what information is logged to what location. You can even define different destinations for different logs. For example, information can be logged to files or a terminal, or (if it is very important) a message can be written to one or more users who are logged in at that moment. Listing 3-7 shows the default contents of /etc/syslog.conf.
Listing 3-7. Contents of syslog.conf

root@RNA:~# cat /etc/syslog.conf
# /etc/syslog.conf     Configuration file for syslogd.
#
# For more information see syslog.conf(5)
# manpage.
#
# First some standard logfiles.  Log by facility.
#
auth,authpriv.*              /var/log/auth.log
.*;auth,authpriv.none        -/var/log/syslog
#cron.*                       /var/log/cron.log
daemon.*                     -/var/log/daemon.log
kern.*                       -/var/log/kern.log
lpr.*                         -/var/log/lpr.log
mail.*                        -/var/log/mail.log
user.*                        -/var/log/user.log
uucp.*                        /var/log/uucp.log

# Logging for the mail system. Split it up so that
# it is easy to write scripts to parse these files.
#
mail.info                    -/var/log/mail.info
mail.warn                    -/var/log/mail.warn
mail.err                     /var/log/mail.err

# Logging for INN news system
#
news.crit                    /var/log/news/news.crit
news.err                     /var/log/news/news.err
news.notice                   -/var/log/news/news.notice

# Some 'catch-all' logfiles.
#
.*=debug;\
  auth,authpriv.none;\
  news.none;mail.none       -/var/log/debug

.*=info;*=notice;*=warn;\
  auth,authpriv.none;\
  cron,daemon.none;\
  mail,news.none            -/var/log/messages
# Emergencies are sent to everybody logged in.
#
*.*.emerg
#
# I like to have messages displayed on the console, but only on a virtual
# console I usually leave idle.
#
#daemon,mail.*;\
   news.=crit;news.=err;news.=notice;\
   *.=debug;*.=info;\
   *.=notice;*.=warn       /dev/tty8

# The named pipe /dev/xconsole is for the `xconsole' utility. To use it,
# you must invoke `xconsole' with the `-file' option:
#
# $ xconsole -file /dev/xconsole [...]
#
# NOTE: adjust the list below, or you'll go crazy if you have a reasonably
#       busy site..
#
#daemon.*;mail.*;\
   news.crit;news.err;news.notice;\
   *.debug;*.info;\
   *.notice;*.warn       /dev/xconsole

You can see from this listing that different rules are specified to define logging, and each
of these rules has different parts. The first part of a log definition is the facility, which provides
a basic idea of what part of the system the log message came from. The following available
facilities are predefined:

- **auth**: Generic information, related to the authentication process.
- **authpriv**: See auth.
- **cron**: Information that is related to the crond and atd processes.
- **daemon**: Generic information used by different system processes (daemons) that don't
  have a log facility of their own.
- **kern**: Everything that is related to the kernel. To log this information, a helper process
  named klogd is used. This process makes sure that information generated during the
  boot procedure is also logged.
- **lpr**: Information related to the printing subsystem.
• **mail**: Everything related to the mail system. Pay special attention to this because a misconfigured log line for the mail facility may cause lots and lots of information to be logged.

• **mark**: This is a marker that can be periodically written to the log files automatically.

• **news**: All events related to a news server (if such a server is used).

• **syslog**: Internally used by the **syslogd** process.

• **user**: Generic facility that can be used for user-related events.

• **uucp**: Messages that are related to the legacy UUCP system.

• **local0-7**: Local log facilities available for customized use. This facility can be used to assign a log facility to specific processes.

Apart from these specific facilities, a * can also be used to refer to all facilities. You can see an example of this in the last line of Listing 3-7, in which *=warn is used to handle warnings that are generated by whatever service.

For each facility, a priority is used to specify the severity of an event. Apart from *, which refers to all priorities, the following priorities can be used:

• **none**: Use this to ensure that no information related to a given facility is logged.

• **debug**: This priority is used only for troubleshooting purposes. It logs as much information as it can and is therefore very verbose. (Don't ever switch it on as a default setting.)

• **info**: This priority logs messages that are categorized as informational. Don't use this one as a default setting either because it generates lots of information.

• **notice**: Use this priority to log normal system events. This priority keeps you up to date about what specific services are doing.

• **warning**: This priority should be switched on by default for most services. It logs warnings related to your services.

• **err**: Use this priority to log serious errors that disrupt the process functionality.

• **crit**: This priority is used to log critical information that is related to programs.

• **alert**: Use this priority to log information that requires immediate action to keep the system running.

• **emerg**: This priority is used in situations in which the system is no longer usable.

These priorities are shown in increasing order of severity. The first real priority (debug) relates to the least important events, whereas the emerg priority should be reserved for the most important. If a certain priority is specified, as in *=warn, all priorities with a higher importance are automatically included as well. If you want to refer to a specific priority, you should use the = sign, as in *=warn. Using the = sign allows you to log events with a specific
priority to specific destinations only, which happens for example for the mail process, which by default has a log file for warnings, both for errors and for informational purposes.

The last part of the syslog configuration is the specification of the log destination. Most processes log to a file by default, but other possibilities exist:

- To log to a file, specify the name of the file. If you anticipate large numbers of log messages, it’s a good idea to prepend the name of the file with a -, as in `news.* -/var/log/news`. Using the hyphen ensures that messages are cached before they are written to a log file. This decreases the workload caused by logging information, but, if the system crashes and the cache isn’t written to disk, messages will be lost.

- To log to a device, just specify the name of the device that you want to log to. As can be seen from the example log file in Listing 3-7, important messages are logged to `/dev/xconsole` by default. It may also be a good idea to log important messages, such as those that have a priority of warn and higher, to an unused `tty`.

- To send alerts to users who are logged in, just specify the name of the user. In the example `*.alert root,linda`, all messages with at least an alert priority are written to the `tty` in which users linda and root are logged in at that moment.

- To send log messages to a specific log server, include the name of the server, preceded by an @. This server has to be configured as a log server by starting the log process with the `-r` option.

- For the most serious situations, use * to ensure that a message is written to all users who are logged in at that moment.

By default, syslog writes log messages to log files in the `/var/log` directory, where you can find log information that is created in many different ways. One of the most important log files that you’ll find in this directory is `/var/log/messages`. Listing 3-8 shows some lines from this file.

Listing 3-8. *Some Lines from /var/log/messages*

<table>
<thead>
<tr>
<th>Time</th>
<th>User</th>
<th>Process</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jun 7 03:14:58</td>
<td>RNA gconfd (root-5150)</td>
<td>Resolved address &quot;xml:readwrite:/root/.gconf&quot; to a writable configuration source at position 1</td>
<td></td>
</tr>
<tr>
<td>Jun 7 03:14:58</td>
<td>RNA gconfd (root-5150)</td>
<td>Resolved address &quot;xml:readonly:/etc/gconf/gconf.xml.defaults&quot; to a read-only configuration source at position 2</td>
<td></td>
</tr>
<tr>
<td>Jun 7 03:14:58</td>
<td>RNA gconfd (root-5150)</td>
<td>Resolved address &quot;xml:readonly:/var/lib/gconf/debian.defaults&quot; to a read-only configuration source at position 3</td>
<td></td>
</tr>
</tbody>
</table>

All lines in `/var/log/messages` are structured in the same way. First, you see the date and time that the message was logged. Next, you see the name of the server that the message comes from. In the example lines in Listing 3-8, you can see that the three log messages all come from the same server (RNA), and you can see the name of the process that generated the message. This process name is followed by the unique process ID and the user who runs the process. Finally, the message itself is written.
The files that are created on your server really depend on the services that are installed. Here's a list of some of the important ones:

- **apache2**: This subdirectory contains the access log and error log for your Apache web server.
- **auth.log**: Here you’ll find a list of authentication events. Typically, you’ll see when user root has authenticated to the server.
- **dmesg**: This file has a list of messages generated by the kernel. Typically, it’s quite helpful when analyzing what has happened at the kernel level when booting your server.
- **faillog**: This is a binary file that contains messages about login failures that have occurred. Use the `faillog` command to check its contents.
- **mail.***: These files contain information on what happened on the mail service that may be running at your server. These logs can become quite big if your server is a mail server because all mail activity will be logged by default.
- **udev**: In this file you can see all the events that have been generated by the hardware plug-and-play manager udev (see Chapter 6 for more information about this). The information in this file can be very useful when troubleshooting hardware problems.

### Logging in Other Ways

Many processes are configured to work with `syslog`, but some important services have their own log configuration. For example, the Apache web server handles logging itself by specifying the names of the files that information has to be logged to in the Apache configuration files. And many other similar services don’t use `syslog`, so, as an administrator, you always have to take a careful look at how logging is handled for each specific service.

---

**Tip** If you need logging from shell scripts, you can use the `logger` command, which writes log messages directly to the `syslog` procedure. It’s a useful way to write a failure in a shell script to a log file. For example, use `logger` this script completed successfully if you want to write to the log files that a script has completed successfully.

### Rotating Log Files

Logging is good, but, if your system writes too many log files, it all can become rather problematic. Log files grow quite large and can rapidly fill your server’s hard drive. As a solution to this, you can configure the `logrotate` service. This runs as a daily `cron` job, which means that it is started automatically and checks its configuration files to see if any rotation has to occur. In
these configuration files, you can configure when a new log file should be opened and, if so, what exactly should happen to the old log file: should it be compressed or just deleted? And, if it is compressed, how many versions of the old file should be kept?

You can use logrotate with two different kinds of configuration files. The main configuration file is /etc/logrotate.conf. In this file, generic settings are defined to tune how logrotate should do its work. Listing 3-9 shows the contents.

Listing 3-9. Contents of the logrotate.conf Configuration File

```bash
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# uncomment this if you want your log files compressed
#compress

# uncomment these to switch compression to bzip2
compresscmd /usr/bin/bzip2
uncompresscmd /usr/bin/bunzip2

# former versions had to have the compresscommand set accordingly
#comprressext .bz2

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d

# no packages own wtmp -- we'll rotate them here
#/var/log/wtmp {
#    monthly
#    create 0664 root utmp
#    rotate 1
#}

# system-specific logs may be also be configured here.

In this example, some important keywords are used, and Table 3-2 describes them.
Table 3-2. Options for logrotate

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weekly</td>
<td>This option specifies that the log files should be created on a weekly basis.</td>
</tr>
<tr>
<td>rotate 4</td>
<td>This option makes sure that four old versions of the file are saved. If the rotate option is not used, old files are deleted.</td>
</tr>
<tr>
<td>create</td>
<td>The old file is saved under a new name and a new file is created.</td>
</tr>
<tr>
<td>compress</td>
<td>Use this option to make sure the old log files are compressed.</td>
</tr>
<tr>
<td>compresscmd</td>
<td>This option specifies the command that should be used for creating the compressed log files.</td>
</tr>
<tr>
<td>uncompresscmd</td>
<td>Use this command to specify what command to use to uncompress compressed log files.</td>
</tr>
<tr>
<td>include</td>
<td>This important option makes sure that the content of the directory /etc/logrotate.d is included. In this directory, files exist that specify how to handle some individual log files.</td>
</tr>
</tbody>
</table>

As you have seen, the logrotate.conf configuration file includes some very generic code to specify how log files should be handled. In addition to that, most log files have a specific logrotate configuration file in /etc/logrotate.d/.

The content of the service-specific configuration files in /etc/logrotate.d is generally more specific than the content of the generic logrotate.conf. Listing 3-10 shows the configuration script for files that are written by Apache to /var/log/apache2/.

Listing 3-10. Example of the logrotate Configuration for Apache

```
/var/log/apache2/*.log {
    weekly
    missingok
    rotate 52
    compress
delaycompress
    notifempty
    create 640 root adm
    sharedscripts
    postrotate
        if [ -f /var/run/apache2.pid ]; then
            /etc/init.d/apache2 restart > /dev/null
        fi
    endscript
}
```

This example uses some more important options. Table 3-3 provides a description of these options.
### Table 3-3. Options in Service-Specific `logrotate` Files

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dateext</td>
<td>Uses the date as an extension for old versions of the log files.</td>
</tr>
<tr>
<td>maxage</td>
<td>Specifies the number of days after which old log files should be removed.</td>
</tr>
<tr>
<td>rotate</td>
<td>Used to specify the number of times a log file should be rotated before being removed or mailed to the address specified in the mail directive.</td>
</tr>
<tr>
<td>size</td>
<td>Log files that exceed the size limit are specified here.</td>
</tr>
<tr>
<td>notifempty</td>
<td>Do not rotate the log file when it is empty.</td>
</tr>
<tr>
<td>missingok</td>
<td>If the log file does not exist, go on to the next one without issuing an error message.</td>
</tr>
<tr>
<td>copytruncate</td>
<td>Truncate the old log file in place after creating a copy, instead of moving the old file and creating a new one. This is useful for services that cannot be told to close their log files.</td>
</tr>
<tr>
<td>postrotate</td>
<td>Use this option to specify some commands that should be executed after performing the <code>logrotate</code> on the file.</td>
</tr>
<tr>
<td>endscript</td>
<td>This option denotes the end of the configuration file.</td>
</tr>
</tbody>
</table>

Like the previous example for the Apache log file, all other log files can have their own `logrotate` file. Some more options are available when creating such a `logrotate` file. Check the `man` pages for a complete overview.

### Summary

As the administrator of a Linux server, you will be doing certain tasks on a regular basis. In this chapter you have read about the most important of these tasks: managing software, creating backups, scheduling services to start automatically, and configuring logging. In Chapter 4, you'll learn how to configure a secure environment on Ubuntu Server.