

# Package Management

# Package Management

- As we spend more time with Linux, we see that its software landscape is extremely dynamic. Things are constantly changing.
- Most of the top-tier Linux distributions release new versions every six months and many individual program updates every day.
- To keep up with this blizzard of software, we need good tools for package management.
- Package management is a method of installing and maintaining software on the system

# Packaging Systems

- Different distributions use different packaging systems, and as a general rule, a package intended for one distribution is not compatible with another distribution.
- Most distributions fall into one of two camps of packaging technologies:
  - ✓ the Debian *.deb* camp
  - ✓ the Red Hat *.rpm* camp
- There are some important exceptions such as Gentoo, Slackware, and Arch, but most others use one of these two basic systems as shown in the Table

Packaging System	Distributions (Partial Listing)
Debian Style (.deb)	Debian, Ubuntu, Linux Mint, Raspberry Pi OS
Red Hat Style (.rpm)	Fedora, CentOS, Red Hat Enterprise Linux, OpenSUSE

# How a Package System Works

- The method of software distribution found in the proprietary software industry usually entails buying a piece of installation media such as an “install disk” or visiting a vendor’s web site and downloading a product and then running an “installation wizard” to install a new application on the system.
- Linux doesn't work that way. Virtually all software for a Linux system will be found on the Internet.
- Most of it will be provided by the distribution vendor in the form of package files, and the rest will be available in source code form that can be installed manually

# Package Files

- The basic unit of software in a packaging system is the package file. A package file is a compressed collection of files that comprise the software package.
- A package may consist of numerous programs and data files that support the programs.
- In addition to the files to be installed, the package file also includes metadata about the package, such as a text description of the package and its contents.
- Additionally, many packages contain pre- and post-installation scripts that perform configuration tasks before and after the package installation.
- Package files are created by a person known as a package maintainer, often (but not always) an employee of the distribution vendor.
- The package maintainer gets the software in source code form from the upstream provider (the author of the program), compiles it, and creates the package metadata and any necessary installation script

# Repositories

- While some software projects choose to perform their own packaging and distribution, most packages today are created by the distribution vendors and interested third parties.
- Packages are made available to the users of a distribution in central repositories that may contain many thousands of packages, each specially built and maintained for the distribution.
- A distribution may maintain several different repositories for different stages of the software development life cycle.
- A distribution may also have related third-party repositories. These are often needed to supply software that, for legal reasons such as patents or DRM anti-circumvention issues, cannot be included with the distribution

# Dependencies

- Programs are seldom “standalone”; rather they rely on the presence of other software components to get their work done.
- Common activities, such as input/output for example, are handled by routines shared by many programs.
- These routines are stored in what are called shared libraries, which provide essential services to more than one program.
- If a package requires a shared resource such as a shared library, it is said to have a dependency.
- Modern package management systems all provide some method of dependency resolution to ensure that when a package is installed, all of its dependencies are installed, too.

# High and Low-level Package Tools

- Package management systems usually consist of two types of tools.
  - Low-level tools which handle tasks such as installing and removing package files
  - High-level tools that perform metadata searching and dependency resolution

Distributions	Low-Level Tools	High-Level Tools
Debian style	dpkg	apt, apt-get, aptitude
Fedora, Red Hat Enterprise Linux, CentOS	rpm	dnf, yum

# Common Package Management Tasks

- Many operations can be performed with the command line package management tools.
- the term `package_name` refers to the actual name of a package rather than the term `package_file`, which is the name of the file that contains the package.
- Also, before any package operations can be performed, the package repository needs to be queried so that the local copy of its database can be synchronized.
- Red Hat's `dnf` program does this automatically and updates the local database if too much time has elapsed since the last update.
- On the other hand, Debian's `apt` program must be run with the `update` command to explicitly update the local database. This needs to be done every so often.
- In the examples below, the `apt update` command is done before any operations, but in real life this only needs to be done every few hours to stay safe.

# Finding a Package in a Repository

- Using the high-level tools to search repository metadata, a package can be located based on its name or description.

Style	Command(s)
Debian	<code>apt update; apt search <i>search_string</i></code>
Red Hat	<code>dnf search <i>search_string</i></code>

- For example, to search a dnf repository for the emacs text editor, we can use this command:  
`dnf search emacs`

# Installing a Package from a Repository

- High-level tools permit a package to be downloaded from a repository and installed with full dependency resolution.

Style	Command(s)
Debian	<code>apt update; apt install <i>package_name</i></code>
Red Hat	<code>dnf install <i>package_name</i></code>

- For example, to install the emacs text editor from an apt repository on a Debian system, we can use this command:  
`apt update; apt install emacs`

# Installing a Package from a Package File

- If a package file has been downloaded from a source other than a repository, it can be installed directly (though without dependency resolution) using a low-level tool.

Style	Command(s)
Debian	<code>dpkg -i <i>package_file</i></code>
Red Hat	<code>rpm -i <i>package_file</i></code>

- For example, if the `emacs-22.1-7.fc7-i386.rpm` package file had been downloaded from a non-repository site, it would be installed this way:

```
rpm -i emacs-22.1-7.fc7-i386.rpm
```

# Removing a Package

- Packages can be uninstalled using either the high-level or low-level tools. The high-level tools are shown in Table.

Style	Command(s)
Debian	<code>apt remove <i>package_name</i></code>
Red Hat	<code>dnf erase <i>package_name</i></code>

- For example, to uninstall the emacs package from a Debian-style system, we can use this command:

```
apt remove emacs
```

# Updating Packages from a Repository

- The most common package management task is keeping the system up-to-date with the latest versions of packages. The high-level tools can perform this vital task in a single step.

Style	Command(s)
Debian	<code>apt update; apt upgrade</code>
Red Hat	<code>dnf update</code>

- For example, to apply all available updates to the installed packages on a Debian-style system, we can use this command:  
`apt update; apt upgrade`

# Upgrading a Package from a Package File

- If an updated version of a package has been downloaded from a non-repository source, it can be installed, replacing the previous version.

Style	Command(s)
Debian	<code>dpkg -i <i>package_file</i></code>
Red Hat	<code>rpm -U <i>package_file</i></code>

- For example, to update an existing installation of emacs to the version contained in the package file `emacs-22.1-7.fc7-i386.rpm` on a Red Hat system, we can use this command:

```
rpm -U emacs-22.1-7.fc7-i386.rpm
```

# Listing Installed Packages

- The table lists the commands we can use to display a list of all the packages installed on the system.

Style	Command(s)
Debian	<code>dpkg -l</code>
Red Hat	<code>rpm -qa</code>

# Determining Whether a Package is Installed

- The table lists the low-level tools we can use to display whether a specified package is installed.

Style	Command(s)
Debian	<code>dpkg -s <i>package_name</i></code>
Red Hat	<code>rpm -q <i>package_name</i></code>

- For example, to determine whether the emacs package is installed on a Debian style system, we can use this command:

```
dpkg --status emacs
```

# Displaying Information About an Installed Package

- If the name of an installed package is known, we can use the commands in the Table to display a description of the package.

Style	Command(s)
Debian	<code>apt show <i>package_name</i></code>
Red Hat	<code>dnf info <i>package_name</i></code>

- For example, to see a description of the emacs package on a Debian-style system, we can use this command:  
`apt-cache show emacs`

# Finding Which Package Installed a File

- To determine what package is responsible for the installation of a particular file, we can use the commands in the Table.

Style	Command(s)
Debian	<code>dpkg -S file_name</code>
Red Hat	<code>rpm -qf file_name</code>

- For example, to see what package installed the `/usr/bin/vim` file on a Red Hat system, we can use the following:
- `rpm -qf /usr/bin/vim`