Chapter 24
Package Management

Introduction

- The two most common software management systems were started by two of the most popular Linux distributions:
    - Red Hat (RPM Package Management)
    - Debian (Debian Package Management)
- Another Linux distribution, **SUSE**, uses the package management system developed by Red Hat as well as its own package management system called **Zypper**.
- Package management systems:
    - Provide ability to download, install, configure, query, update and remove software.
    - Ensures requirements of software called *dependencies* are maintained

RPM Package Management

- Originally called Red Hat Package Management.
- RPM files are available in two formats:
  - Source file (`.src.rpm`) used to compile on any architecture
  - Binary file (`.rpm`) pre-compiled for a specific architecture
- Compiled `.rpm` naming convention:

```
package_name-version-release.architecture.rpm
```

```
x3270-x11-3.3.6-10.5.el6.i686.rpm
```

RPM Queries

- The basic level command for managing RPM files is the `rpm` command
- The `rpm` command is able to perform many queries related to software packages.
- The `rpm` command can query information about:
  - Packages installed on the system.
  - RPM files that are either on the filesystem or reachable by an URL.
- The difference between doing a query about an installed package versus an RPM file:
  - If not installed, add the `-p FILE` option to the query.
  - To query an installed package, use only the package name.

RPM Queries

- To perform any RPM query, use the `-q` option with the `rpm` command. To query for basic package information, use the `-i` option:

```
[sysadmin@localhost ~]$ rpm -qi bash
Name : bash
Version : 4.2.46
Release : 31.el7
Architecture: x86_64
Install Date: Tue Dec 4 14:38:13 2018
Group : System Environment/Shells
Size : 3667773
License : GPLv3+
Signature : RSA/SHA256, Mon Nov 12 14:21:49 2018, Key ID 24c6a8a7f4a80eb5
Source RPM : bash-4.2.46-31.el7.src.rpm
Build Date : Tue Oct 30 17:09:33 2018
Build Host : x86-01.bsys.centos.org
Relocations : (not relocatable)
Packager : CentOS BuildSystem <http://bugs.centos.org>
Vendor : CentOS
URL : http://www.gnu.org/software/bash
Summary : The GNU Bourne Again shell
Description :
The GNU Bourne Again shell (Bash) is a shell or command language interpreter that is compatible with the
Bourne shell (sh). Bash incorporates useful features from the Korn shell (ksh) and the C shell (csh).
Most sh scripts can be run by bash without modification.
```

RPM Queries

- To perform a similar query on the RPM file for bash, the `-p` option would be added along with the `-q` and `-i` options:

```
[sysadmin@localhost ~]$ rpm -qip bash-4.2.46-31.el7.src.rpm
```

- To query whether a package is installed, a user can use the –q option:
  - If the package is installed, the output will be:

```
 [sysadmin@localhost ~]$ rpm -q bash
bash-4.2.46-31.el7.src.rpm
```

  - If the package is not installed, the output will be:

```
 [sysadmin@localhost ~]$ rpm -q pickle
package pickle is not installed
```

- To display a list of all installed RPMs:

```
[sysadmin@localhost ~]$ rpm -qa
```

RPM Queries

- The following table shows different types of queries that can be performed:

Option	Purpose
`-a`	List all of the installed packages currently on the system
`-c`	Display a list of configuration files that belong to the package
`-d`	List the documentation files that belong to the package
`-i`	Display the package information
`-K`	Check the package integrity
`-l`	List all files in the package
`--provides`	List the capabilities this package provides

RPM Queries

Option	Purpose
`-R`	List the capabilities that this package requires
`--scripts`	List the scripts that are used before and after installation of the package
`-s`	Display the state of each package file as normal, not installed or replaced

RPM Scripts

- RPM files may contain commands to execute during installation or removal.
- As installation or removal of an RPM normally requires root privileges, these scripted commands will be executed as the root user which can be dangerous.
- To inspect the scripts use:

```
rpm -qp --scripts package-version-release-arch.rpm
```

```
[sysadmin@localhost ~]$ rpm -qp --scripts x3270-x11-3.3.6-10.5.el6.i686.rpm
```

RPM Integrity

- RPM files are *signed* by the private key of the distributor, who publishes the corresponding public key.
- Red Hat-derived distributions store public keys in the `/etc/pki/rpm-gpg` directory.
- An administrator can import these keys into the RPM database with the command:

```
[sysadmin@localhost ~]$ rpm --import /etc/pki/rpm-gpg/*
```

- The `rpm` command can verify the integrity of an RPM signed by the matching private key:

```
[sysadmin@localhost ~]$ rpm -qpK x3270-x11-3.3.6-10.5.el6.i686.rpm
x3270-x11-3.3-4.el7.x86_64.rpm: rsa sha1 (md5) pgp md5 OK
```

  - Notice the only output that is capitalized is `OK`, meaning that the package is not corrupted.
  - If output states `NOT OK`, it is a corrupted package.

Installing Packages with RPM

- Unlike queries, installation of an RPM normally requires root privileges.
- Using the `rpm –i` command to install an RPM package will fail if the dependencies for that package are not installed.
- The error message when installation fails can be helpful to locate the missing dependencies.
- For example, to install the `x3270-x11-3.3.6-10.5.el6.i686.rpm` package, execute:

```
[sysadmin@localhost ~]$ rpm -i x3270-x11-3.3.6-10.5.el6.i686.rpm
```

- If there were any missing dependencies, the `rpm` command would fail and display an error message like the following:

```
error: Failed dependencies:
```

Erasing Packages with RPM

- Remove packages with the `-e` option for the `rpm` command.
- You can not remove a package that another package depends on, attempting to do so will result in an error:

```
[sysadmin@localhost ~]$ rpm -e x3270 libicu
error: Failed dependencies:
       x3270 = 3.3.6 is needed by (installed) x3270-x11-3.3.6-10.5.el6.i686
       libicuuc.so.42 is needed by (installed)      x3270-x11-3.3.6-10.5.el6.i686
```

- All the packages can be removed with a single `rpm` command by specifying all package names as an argument:

```
[sysadmin@localhost ~]$ rpm -e x3270-x11 x3270 libicu
```

Updating Packages with RPM

- Use `-U` option to either update to a new version or to install:

```
[sysadmin@localhost ~]$ rpm -U x3270-4.3.6-10.5.el6.i686.rpm
```

- Use the `-F` (*freshen*) option to only update if package is already installed:

```
[sysadmin@localhost ~]$ rpm -F x3270-4.3.6-10.5.el6.i686.rpm
```

- Kernel packages should never be updated; always use `-i` to install.

Using rpm2cpio

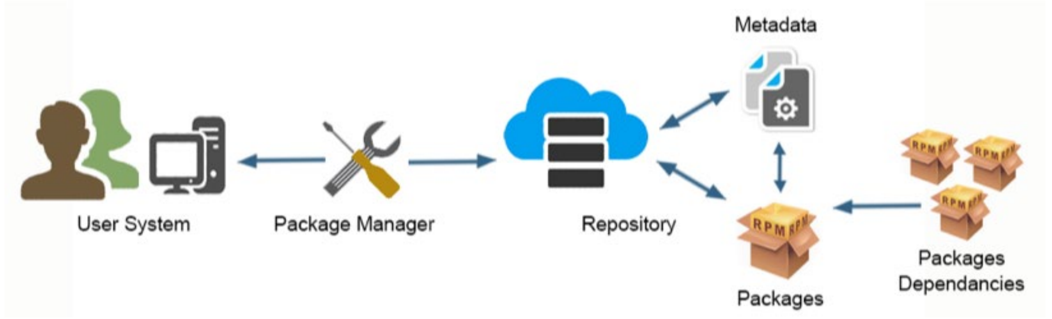
- Use `rpm2cpio` to extract individual files from a package or list contents.
- The `rpm2cpio` command takes package as input and produces a `cpio` format as output.
- For example, to list the contents of an `.rpm` file use :

```
[sysadmin@localhost ~]$ rpm2cpio telnet-server-0.17-47.el6_3.1.i686.rpm | cpio -t
```

- Why use the `rpm2cpio` command?
    - If a key package file is accidentally removed from the system, reinstalling the entire package may be overkill if only a single file is needed.
    - Using the `rpm2cpio` command, the file that is missing can be extracted and copied back into the appropriate directory.

Managing Packages With yum

- The `yum` command is another software management command.
- Advantages of using `yum`:
  - It can automatically resolve dependencies.
  - It can download packages from a repository server.
  - It can query package information from repository server.



Managing Packages With yum

- The `yum` command can be configured by modifying:
  - The `/etc/yum.conf` file.
  - The `/etc/yum.repos.d` directory.
- Use the `provides` subcommand to `yum` to determine which package owns a file:

```
[sysadmin@localhost ~]$ yum provides /usr/lib/libicuuc.so.42
```

- Search for key terms by using the `search` subcommand to `yum`:

```
[sysadmin@localhost ~]$ yum search terminal
```

Installing Packages with yum

- To install package with `yum`, use:

```
yum install package
```

- When installing, a prompt will appear:

```
Install 3 Package(s)
Total download size: 5.5 M
Installed size: 20 M
Is this ok [y/N]:
```

- To install package without prompts:

```
yum –y install package
```

- Multiple packages can be installed:

```
[root@localhost ~]# yum install telnet telnet-server ftp vsftpd
```

Installing Packages with yum

- A *yum group* is a collection of packages that work together to create a large piece of software.
- To list all of the groups that are installed and available on a system:

```
[root@localhost ~]# yum grouplist
```

- To see details about a specific group, use:

```
yum groupinfo group_name
```

- To install a group of packages:

```
yum groupinstall group_name
```

```
[root@localhost ~]# yum groupinstall "Office Suite and Productivity"
```

Removing Packages With yum

- When removing a package that has other packages that depend upon it, then the yum command will offer to remove the dependencies.
- To remove a package use either of the commands below:

```
yum remove package
```

```
yum erase package
```

- Use the `-y` option to remove without prompts (not recommended).
- The `yum` command will remove all dependency packages as well.

Updating Packages With yum

- To list packages that need to be updated use:

```
yum list updates
```

- To update one or more individual packages, the following syntax can be used:

```
yum update package
```

Installing Packages with DNF

- Dandified Yum, or DNF, is the next upcoming version of the `yum` command.
- DNF is designed to solve package management dependency issues that were present in `yum`.
- Another benefit of DNF over `yum` is a clearly documented Application Programming Interface (API).
- The `dnf` command can be executed using the following syntax:

```
dnf [OPTIONS] <COMMAND> [<ARGUMENTS>...]
```

Installing Packages with DNF

- To list all available packages on the system, use:

```
[root@localhost ~]# yum dnf list --available
```

- To install a package, the `install` option can be used:

```
dnf install package
```

```
[root@localhost]# dnf install cowsay.noarch
```

- To list the recently installed packages using the `list` command with the `--installed` option:

```
[root@localhost]# dnf list --installed
```

Debian Package Management

- Debian's package management system is based upon the format for the Debian distribution.
- The Debian package file names end in the `.deb` extension.
- Used on several distributions, including Ubuntu and Mint.
- The `.deb` files are binary files compiled to execute on a particular computer architecture.
- The source packages, which contain the original source code, have an extension of `.dsc`.

Debian Package Management

- Before working with packages, run the `apt-get update` command to ensure that the system has a current list of the packages.
- Debian package files follow this naming convention:

```
package-name_version-release_architecture.deb
```

Package name:	`joe_3.7-2.3_i386.deb`
Version:	`joe_3.7-2.3_i386.deb`
Release:	`joe_3.7-2.3_i386.deb`
Architecture:	`joe_3.7-2.3_i386.deb`

Installing Software Packages with dpkg

- The `dpkg` command can be used for installing, removing, and querying packages.
- To install a software package in a Debian-based distribution, use the `-i` option with the `dpkg` command.

```
sysadmin@localhost:~$ dpkg -i joe_3.7-2.3_i386.deb
```

- Debian packages also may have dependencies.
- The `dpkg` tool does not provide any automatic way to resolve dependency issues.

Removing Software With dpkg

- There are two options available to remove software using `dpkg`:
  - The `-r` option removes most of the software but leaves the configuration files in case it is reinstalled.
  - The `-P` (purge) option removes all of the software and configuration files.

Listing Packages with dpkg

- Use the `-l` option to list all currently installed packages:

```
sysadmin@localhost:~$ dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name Version Architecture Description
+++-===============-============-============-================================
ii adduser 3.113+nmu3ub all add and remove users and groups
ii apt 1.0.1ubuntu2 amd64 command-line package manager
ii apt-file 2.5.2ubuntu1 all search for files within Debian pa
...
```

- The output above contains the following information:
  - Desired status of the package; `i` for installed, `u` for unknown, `r` for remove and `h` for hold.
  - Actual status of the package; `i` means installed and `n` means not installed.
  - The package name, version (combined with the release) and the description of the package.

Configuring Packages With dpkg

- Debian package management configures packages when they are installed.
- If the package needs to be reconfigured at some later point, use the `dpkg-reconfigure` command:

```
dpkg-reconfigure [OPTIONS...] PACKAGES...
```

```
root@localhost:~# dpkg-reconfigure tzdata
```

Searching for Packages With apt-cache

- The system must have the appropriate repositories configured in the `/etc/apt/sources.list` file.

- Users can search for packages by using the `apt-cache search` command:

```
apt-cache search KEYWORD
```

```
sysadmin@localhost:~$ apt-cache search medusa
```

- To display package dependencies, use:

```
apt-cache depends PACKAGE
```

```
sysadmin@localhost:~$ apt-cache depends apache2
apache2
  PreDepends: dpkg
  Depends: lsb-base
...
```

Searching for Packages With apt-cache

- To display packages that need to be updated, use:

```
apt-get update
```

- To display package status, use:

```
apt-cache show PACKAGE
```

Installing/Updating Packages With apt-get

- Unlike `dpkg`, the `apt-get` command automatically resolves dependencies.
- To download and install a package as well as its dependencies:

```
apt-get install PACKAGE
```

- The `apt-get install` command can also update a package, if that package is installed.
- To only update use the `apt-get --only-upgrade install` command.
- To update all packages of the system:
  - `apt-get update`
  - `apt-get upgrade`

Removing/Purging Packages With apt-get

- Like `dpkg`, the `apt-get` command is able to either remove or purge a package.
- To remove a package, but keep the configuration files in case it is reinstalled:

```
apt-get remove PACKAGE
```

- To remove a package, including the configuration files:

```
apt-get purge PACKAGE
```

Verifying Files With Checksums

- Verifying the integrity of files is critical for users, administrators, and programmers.
- A *checksum* is a small piece of complementary data used to verify the integrity of a file.
- The file's creator generates a checksum by applying a cryptographic hashing algorithm to the file's contents.
- Recipients who obtain a copy of the file can verify integrity by repeating the same hashing algorithm on the copy.

md5sum

- The `md5sum` command is based on the MD5 (message-digest 5) algorithm.
- It creates a 128-bit hash using the original file.
- To create a checksum with the `md5sum` command, use:

```
md5sum [OPTIONS]... [FILE]...
```

- For example, a hash is created for a file using the `md5sum` command:

```
[sysadmin@localhost]$ md5sum anyfile.txt > anyfile.md5
[sysadmin@localhost]$ ls
anyfile.md5 anyfile.txt
[sysadmin@localhost]$ cat anyfile.md5
d41d8cd98f00b204e9800998ecf8427e anyfile.txt
```

- The `md5sum` command can also be used to authenticate the file(s) with the *check* `–c` option.

sha256sum

- The `sha256sum` command creates a 256-bit checksum number that can be used to verify a file.

- To create a checksum with the `sha256sum` command, use:

```
sha256sum [OPTIONS]... [FILE]...
```

- For example, a hash is created for a file using the `sha256sum` command:

```
[sysadmin@localhost]$ sha256sum anyfile.txt > anyfile.sha256
[sysadmin@localhost]$ cat anyfile.sha256
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855 anyfile.txt
```

- The `sha256sum` command can also be used to authenticate the file(s) with the *check* `-c` option.

```
[sysadmin@localhost]$ sha256sum -c anyfile.sha256
anyfile.txt: OK
```

sha512sum

- The `sha512sum` command creates a more secure 512-bit checksum number for verifying files.
- To create a checksum with the `sha512sum` command, use:

```
sha512sum [OPTIONS]... [FILE]...
```

- For example, a hash is created for a file using the `sha512sum` command:

```
[sysadmin@localhost]$ sha512sum anyfile.txt > anyfile.sha512
[sysadmin@localhost]$ cat anyfile.sha512
cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0
ff8318d2877eec2f63b931bd47417a81a538327af927da3e anyfile.txt
```

- Can also be used to authenticate the file(s) with the *check* `-c` option.

```
[sysadmin@localhost]$ sha512sum -c anyfile.sha512
anyfile.txt: OK
```

SUSE Package Management

- The **ZYpp/libzypp** package management engine primarily runs on openSUSE, SUSE Linux Enterprise, and Ark.
- ZYpp/libzypp is the background package management engine.
- The `zypper` command is the front-end command line tool.
- *Repositories* are a shared network resource or local directory that provides a location for searching for software to install and update.
- Software that is not part of the official openSUSE or SUSE Linux Enterprise distributions can be found in repositories.

SUSE Package Management

- The `zypper` command is used to query, install, remove, update, manage repositories, and more.

- To use the `zypper` command:

```
zypper [--GLOBAL-OPTS] COMMAND [--COMMAND-OPTS] [COMMAND-ARGUMENTS]
```

- To learn more about the `zypper` command's syntax and usage:

```
localhost:~ # zypper help
```

Searching for Packages With zypper

- Before using `zypper`, refresh the information that `zypper` has about the repositories:

```
localhost:~ # zypper ref
```

- To find a package to install, use the *search* `se` command:

```
zypper se PACKAGE
```

```
localhost:~ # zypper se gvim
Loading repository data...
Reading installed packages...
S | Name | Summary        | Type
--+------+---------------+--------
  |gvim  | A GUI for Vi  | package
```

- The first column, `S`, is the status column; if package was installed, there would be an `i` in this column.

Installing Packages With zypper

- The zypper command can be used with the *install* in command to install packages:

```
zypper in PACKAGE_NAME
```

```
localhost:~ # zypper in gvim
```

- Now that the gvim package is installed, there is an i in the Status column:

```
localhost:~ # zypper se gvim
Loading repository data...
Reading installed packages...
S | Name | Summary        | Type
--+------+----------------+--------
i |gvim  | A GUI for Vi | package
```

- To re-install (and force the overwriting of) the package use the in command again:

```
localhost:~ # zypper in gvim
```

Managing Repositories With zypper

- To query the software repositories on a system, use the `zypper` command with the *list repositories* `lr` option:

```
localhost:~ # zypper lr
```

- To add a repository to install additional software from, use the *add repository* `–ar` option and the repository URL:

```
localhost:~ # zypper ar -f http://packman.inode.at/suse/openSUSE_Leap_15.1/ packman
Adding repository 'packman'............................................[done]
```

- The `zypper ref` command would be run again after to update the system.

Updating Packages With zypper

- The `zypper` command can also be used to update the packages in a repository.

```
zypper –list-updates PACKAGE_NAME
```

- To see a listing of all the packages that are available for update from your repositories:

```
localhost:~ # zypper list-updates -t package
Loading repository data...
Reading installed packages...
No updates found.
```