Chapter 1

Managing Software

1.1 Understanding Meta Package Handlers

In the old days, the `rpm` command was used to install packages, and it was incapable of resolving dependencies (i.e., auto-installing other packages/programs that were needed to make a package work). The syntax needed for rpm is : `rpm -ivh packageName.rpm`. (**i**=install, **v**=verbose, **h**=show hashes about progress).

Nowadays, due to the `yum` package installer, this is no longer an issue. It works with repositories, which are installation sources for a bunch of packages, and the command works by downloading indexes for the repositories. The `yum` meta-package handler needs only the rpm name to install it.

```
1 # yum install blah.rpm
```

At this point the yum command searches the indexes for any dependencies. If any are found, they're downloaded from the repository as well, before the original package is installed.

1.2 Setting up Yum repositories

A standard RHEL installation is hooked up to RHN (Red Hat Network), the RedHat repository, and all patches and updates are downloaded from it. It's the installation source and primary repository for most packages available on RHEL.

1.2.1 yum repolist

This command shows us the list of repositories which our system is configured to use. Unless RHN is connected to, the RHEL 7 Server can't use this (and other) repo commands.

1.2.2 Custom Repository

To convert an existing folder to a yum repository, we need to first go to the `/etc/yum.repos.d` directory, and then create a file named: `repoName.repo` where *repoName* is the name of

our custom repository. It's critical that the file name ends with `.repo` as otherwise yum won't be able to recognize it. The contents of the repoName.repo file should be:

```
[repoName]
name=repoName
baseurl=file:///home/somu/repo
gpgcheck=0
```

The first line is called the label. The second line defines the name of the repository. The third line, defines the URI (Uniform Resource Identifier) where the repository is located. If it's on the internet, protocols such as `ftp://` can be used, but in our case since it's on the local filesystem, we use the `file://` protocol. Further, the path of the repository folder is `/home/somu/repo`, which is what the *baseurl* is set to. The fourth line turns off the GPG file integrity checking (not suggested for real environments).

createrepo

A final step is to generate the indexes required by yum to use the repository. For this, we use the `createrepo` command.

```
# createrepo /downloads
Spawning worker 0 with 4 pkgs
Workers Finished
Saving Primary metadata
Saving file lists metadata
Saving other metadata
Generating sqlite DBs
Sqlite DBs complete
```

Next we can check if the repo was successfully added by running `yum repolist`.

```
# yum repolist
Loaded plugins: fastestmirror, langpacks
repo id                              repo name                              status
base/7/x86_64                        CentOS-7 - Base                        9,591
extras/7/x86_64                      CentOS-7 - Extras                        283
repoTestLabel                        repoTest                                   0
updates/7/x86_64                     CentOS-7 - Updates                     1,134
repolist: 11,008
```

1.3 Using the yum command

The `yum` command is a package manager and a meta package handler. The following are some of the yum commands:

1.3.1 yum search

`yum search` searches the given repositories for a suitable package.

```
# yum search nmap
Loaded plugins: fastestmirror, langpacks
```

```
Loading mirror speeds from cached hostfile
* base: centos.excellmedia.net
* extras: centos.excellmedia.net
* updates: centos.excellmedia.net
======================== N/S matched: nmap ========================
nmap-frontend.noarch : The GTK+ front end for nmap
nmap-ncat.x86_64 : Nmap's Netcat replacement
nmap.x86_64 : Network exploration tool and security scanner

Name and summary matches only, use "search all" for everything.
```

1.3.2 yum install

`yum install` installs the package passed as argument to it, after installing all the required dependencies. When the `-y` option is used, Yum doesn't wait for a (Y/N) reply after showing the dependency list, and proceeds to download and install the package.

```
# yum install -y nmap
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
* base: centos.excellmedia.net
* extras: centos.excellmedia.net
* updates: centos.excellmedia.net
Resolving Dependencies
--> Running transaction check
---> Package nmap.x86_64 2:6.40-7.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
Package        Arch             Version                   Repository      Size
================================================================================
Installing:
nmap           x86_64           2:6.40-7.el7              base           4.0 M

Transaction Summary
================================================================================
Install  1 Package

Total download size: 4.0 M
Installed size: 16 M
Downloading packages:
No Presto metadata available for base
nmap-6.40-7.el7.x86_64.rpm                                 | 4.0 MB   06:38
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : 2:nmap-6.40-7.el7.x86_64                                     1/1
Verifying  : 2:nmap-6.40-7.el7.x86_64                                     1/1

Installed:
nmap.x86_64 2:6.40-7.el7

Complete!
```

Some programs may have a script that needs to be run to setup and configure it. In such cases, yum does it for us.

1.3.3 yum list

The `yum list` command is used to list the packages installed on a system, filtered on a specific criteria.

Options	Description
yum list all	Lists all available and installed packages
yum list installed	Only list the installed packages
yum list avialable	Only list the available packages

1.3.4 yum provides

Sometimes we don't know which package to install. For example, if we want to install and use *semanage*, an important utility to set up SELinux, we have to use the `yum search semanage` command to find all the info about the packages that offer it.

```
# yum search semanage
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
* base: centos.excellmedia.net
* extras: centos.excellmedia.net
* updates: centos.excellmedia.net
============================ N/S matched: semanage ============================
libsemanage-python.x86_64 : semanage python bindings for libsemanage
libsemanage.i686 : SELinux binary policy manipulation library
libsemanage.x86_64 : SELinux binary policy manipulation library
libsemanage-devel.i686 : Header files and libraries used to build policy
: manipulation tools
libsemanage-devel.x86_64 : Header files and libraries used to build policy
: manipulation tools
libsemanage-static.i686 : Static library used to build policy manipulation tools
libsemanage-static.x86_64 : Static library used to build policy manipulation
: tools

Name and summary matches only, use "search all" for everything.
```

The above are the results that contain the string '*semanage*' in their names/descriptions, but may not contain the semanage binary that we require. For such cases, where we know the name of the binary utility, but don't know which package contains it, we use the `yum provides` command. The `*/semanage` is used to indicate it needs to search some file pattern.

```
# yum provides */semanage
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
* base: centos.excellmedia.net
* extras: centos.excellmedia.net
* updates: centos.excellmedia.net
libsemanage-devel-2.5-8.el7.i686 : Header files and libraries used to build
: policy manipulation tools
Repo        : base
```

```
Matched from:
Filename    : /usr/include/semanage

libsemanage-devel-2.5-8.el7.x86_64 : Header files and libraries used to build
: policy manipulation tools
Repo        : base
Matched from:
Filename    : /usr/include/semanage

policycoreutils-python-2.5-17.1.el7.x86_64 : SELinux policy core python
: utilities
Repo        : base
Matched from:
Filename    : /usr/sbin/semanage
Filename    : /usr/share/bash-completion/completions/semanage

policycoreutils-python-2.5-17.1.el7.x86_64 : SELinux policy core python
: utilities
Repo        : @anaconda
Matched from:
Filename    : /usr/sbin/semanage
Filename    : /usr/share/bash-completion/completions/semanage
```

1.3.5 yum remove

`yum remove <packageName>` checks the system to see if any installed packages are dependent upon the package we're trying to remove. If so, it removes the specified package and the dependent packages, unless one (or more) of them are protected. For example, `yum remove bash` fails as it'd have to remove Systemd and yum packages since they are heavily dependent on bash! Again, any `yum remove` command requires a prompt to be answered, which can be bypassed with `yum remove -y`.

```
# yum remove -y nmap
Loaded plugins: fastestmirror, langpacks
Resolving Dependencies
--> Running transaction check
---> Package nmap.x86_64 2:6.40-7.el7 will be erased
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
Package        Arch             Version                  Repository       Size
================================================================================
Removing:
nmap           x86_64           2:6.40-7.el7             @base            16 M

Transaction Summary
================================================================================
Remove  1 Package

Installed size: 16 M
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Erasing    : 2:nmap-6.40-7.el7.x86_64                                     1/1
```

```
Verifying  : 2:nmap-6.40-7.el7.x86_64                                 1/1

Removed:
nmap.x86_64 2:6.40-7.el7

Complete!
```

1.4 Using rpm queries

Any software installed on our RHEL Servers are tracked in an rpm database, which supports queries to find out status and other information about packages. Rpm queries are most useful for SysAdmins when we need to find out more information about a package or software. For example, if we need to find out how to configure a time synchronization service called `chronyd`, first we find out where it is located.

```
# which chronyd
/usr/sbin/chronyd
```

Now that we know where the binary for the chrony daemon is located, we perform an rpm query on it, to find out which package it comes from:

```
# rpm -qf /usr/sbin/chronyd          # Query the package owning <file>
chrony-3.1-2.el7.centos.x86_64
```

Now that we know what package it comes from, we can list everything that the package `chrony` contains:

```
# rpm -ql chrony         # Query list
/etc/NetworkManager/dispatcher.d/20-chrony
/etc/chrony.conf
/etc/chrony.keys
/etc/dhcp/dhclient.d/chrony.sh
/etc/logrotate.d/chrony
/etc/sysconfig/chronyd
/usr/bin/chronyc
/usr/lib/systemd/ntp-units.d/50-chronyd.list
/usr/lib/systemd/system/chrony-dnssrv@.service
/usr/lib/systemd/system/chrony-dnssrv@.timer
/usr/lib/systemd/system/chrony-wait.service
/usr/lib/systemd/system/chronyd.service
/usr/libexec/chrony-helper
/usr/sbin/chronyd
/usr/share/doc/chrony-3.1
/usr/share/doc/chrony-3.1/COPYING
/usr/share/doc/chrony-3.1/FAQ
/usr/share/doc/chrony-3.1/NEWS
/usr/share/doc/chrony-3.1/README
/usr/share/man/man1/chronyc.1.gz
/usr/share/man/man5/chrony.conf.5.gz
/usr/share/man/man8/chronyd.8.gz
/var/lib/chrony
/var/lib/chrony/drift
/var/lib/chrony/rtc
/var/log/chrony
```

To see only the configuration files, instead of all files related to the package, we use:

```
# rpm -qc chrony         # Query config
/etc/chrony.conf
/etc/chrony.keys
/etc/logrotate.d/chrony
/etc/sysconfig/chronyd
```

To find the documentation for the package, we use:

```
# rpm -qd chrony         # Query documentation
/usr/share/doc/chrony-3.1/COPYING
/usr/share/doc/chrony-3.1/FAQ
/usr/share/doc/chrony-3.1/NEWS
/usr/share/doc/chrony-3.1/README
/usr/share/man/man1/chronyc.1.gz
/usr/share/man/man5/chrony.conf.5.gz
/usr/share/man/man8/chronyd.8.gz
```

To view all packages installed on our system, we can use:

```
# rpm -qa          # Query all
```

This command is especially useful to find out which version of a package is installed.

```
# rpm -qa | grep openjdk
java-1.8.0-openjdk-headless-1.8.0.151-1.b12.el7_4.x86_64
java-1.8.0-openjdk-1.8.0.151-1.b12.el7_4.x86_64
```

Pre and Post install Scripts

Many packages include pre and post installation scripts that we may need to find out about. If that is the case, we can use:

```
# rpm -q --scripts java-1.8.0-openjdk
postinstall scriptlet (using /bin/sh):

update-desktop-database /usr/share/applications &> /dev/null || :
/bin/touch --no-create /usr/share/icons/hicolor &>/dev/null || :
exit 0
postuninstall scriptlet (using /bin/sh):

update-desktop-database /usr/share/applications &> /dev/null || :
if [ $1 -eq 0 ] ; then
/bin/touch --no-create /usr/share/icons/hicolor &>/dev/null
/usr/bin/gtk-update-icon-cache /usr/share/icons/hicolor &>/dev/null || :
fi
exit 0
posttrans scriptlet (using /bin/sh):

/usr/bin/gtk-update-icon-cache /usr/share/icons/hicolor &>/dev/null || :
```

This step become critical when working on a production server, especially for security purposes since installing a package requires administrative (root) privileges. If the package is

from an unverified source, we should know what exactly the package installation script does before executing it.

For 3^{rd} party, downloaded packages, that we might not have installed yet, we need to use the `rpm -qp` command instead. Thus, to list the contents of said 3rd party package, we use:

```
# rpm -qpl <packageName>.rpm
# rpm -qp --scripts <packageName>.rpm
```

The second line shows us the scripts (pre and post install) that'll be used by the downloaded (and NOT yet installed) package.

1.4.1 Installing a local rpm file

To perform the installation of an rpm file that we've downloaded from the internet, and it's not in a repository, we use `yum localinstall`.

To download said rpm, we can use a tool like `wget <rpmURL>`.

```
# ls -l
total 4056
-rw-r--r--. 1 root root 4152356 Nov 25  2015 nmap-6.40-7.el7.x86_64.rpm
# yum localinstall nmap-6.40-7.el7.x86_64.rpm
Loaded plugins: fastestmirror, langpacks
Examining nmap-6.40-7.el7.x86_64.rpm: 2:nmap-6.40-7.el7.x86_64
Marking nmap-6.40-7.el7.x86_64.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package nmap.x86_64 2:6.40-7.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
Package     Arch          Version             Repository                      Size
================================================================================
Installing:
nmap        x86_64        2:6.40-7.el7        /nmap-6.40-7.el7.x86_64         16 M

Transaction Summary
================================================================================
Install  1 Package

Total size: 16 M
Installed size: 16 M
Is this ok [y/d/N]: y
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : 2:nmap-6.40-7.el7.x86_64                                     1/1
Verifying  : 2:nmap-6.40-7.el7.x86_64                                     1/1

Installed:
nmap.x86_64 2:6.40-7.el7
```

```
38
39  Complete!
```

1.4.2 repoquery

The repoquery is similar to the rpm query, but instead of querying an installed or not-yet-installed but locally available package, it directly queries the repositories, without even needing to download them! However, the `--scripts` option isn't yet supported by the command.

```
1   # repoquery -ql yp-tools
2   /usr/bin/ypcat
3   /usr/bin/ypchfn
4   /usr/bin/ypchsh
5   /usr/bin/ypmatch
6   /usr/bin/yppasswd
7   /usr/bin/ypwhich
8   /usr/sbin/yppoll
9   /usr/sbin/ypset
10  /usr/sbin/yptest
11  /usr/share/doc/yp-tools-2.14
12  /usr/share/doc/yp-tools-2.14/AUTHORS
13  /usr/share/doc/yp-tools-2.14/COPYING
14  /usr/share/doc/yp-tools-2.14/ChangeLog
15  /usr/share/doc/yp-tools-2.14/NEWS
16  /usr/share/doc/yp-tools-2.14/README
17  /usr/share/doc/yp-tools-2.14/THANKS
18  /usr/share/doc/yp-tools-2.14/TODO
19  /usr/share/doc/yp-tools-2.14/nsswitch.conf
20  /usr/share/locale/de/LC_MESSAGES/yp-tools.mo
21  /usr/share/locale/sv/LC_MESSAGES/yp-tools.mo
22  /usr/share/man/man1/ypcat.1.gz
23  /usr/share/man/man1/ypchfn.1.gz
24  /usr/share/man/man1/ypchsh.1.gz
25  /usr/share/man/man1/ypmatch.1.gz
26  /usr/share/man/man1/yppasswd.1.gz
27  /usr/share/man/man1/ypwhich.1.gz
28  /usr/share/man/man5/nicknames.5.gz
29  /usr/share/man/man8/yppoll.8.gz
30  /usr/share/man/man8/ypset.8.gz
31  /usr/share/man/man8/yptest.8.gz
32  /var/yp/nicknames
```

1.4.3 Displaying information about a package

`repoquery -qi <packageName>` can display information about the package.

```
1   # repoquery -qi awesum
2
3   Name        : awesum
4   Version     : 0.6.0
5   Release     : 1
6   Architecture: noarch
7   Size        : 150637
8   Packager    : Darren L. LaChausse <the_trapper@users.sourceforge.net>
9   Group       : Applications/Security
```

```
URL          : http://awesum.sf.net/
Repository   : Ex11Repo
Summary      : Awesum is an easy to use graphical checksum verifier.
Source       : awesum-0.6.0-1.src.rpm
Description :
Awesum is a graphical checksum verification utility. It is written in Python
and uses the PyGTK toolkit. Awesum is very easy to use and includes support
for both MD5 and SHA checksum algorithms. Unlike many checksum verification
utilities, Awesum features a progress bar which makes working with large files
(such as CD-ROM ISO images) much more bearable.
```
