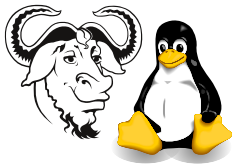

Applying OS Updates

1 Aim

After successfully working through this exercise, You will:

- understand the importance of applying updates to the operating system of servers
- understand how to apply the updates to Red Hat Linux.
- understand how to use RPM and `rpmfind` to resolve software package dependencies
- understand how to automate the update process

2 Background

All operating systems contain bugs. Even Microsoft's OS, despite what Bill may say (See `http://www.cantrip.org/nobugs.html`), contains bugs. See for example, `http://www.gcn.com/archives/gcn/1998/july13/cov2.htm`. And Linux contains bugs. Now crackers are just waiting for a new bug to be uncovered. The company that provides the OS also will provide updates to their OS, thanks to CERT: `http://www.cert.org/`, who eventually publish information about the bug even if the company does not. As a system administrator, it is a vitally important part of your duty to apply updates to important or sensitive computers in your company.

Here in the ICT Department, I have implemented a system that automatically downloads the current updates for Red Hat Linux from the Internet every day. You will use these updates and apply them to your computer. The updates are software packages.

You will use the Network File System (NFS) to *mount* the network drive containing these updates, and then use the Red Hat Package Manager (RPM) to apply these updates to your machine.

The RPM package manager does far more than allow you to install updates. It can also check that the installed software packages are correctly installed, and verify that none of the programs have been changed by a cracker (as long as RPM itself is unchanged!) See chapter 25, *Package Management with RPM*, of the *The Official Red Hat Linux Customization Guide*, available from the Red Hat web site at `http://www.redhat.com/docs/manuals/linux/`, and also on the documentation CDROM (burn a copy on our CD burner).

2.1 The RPM Package Manager

The RPM package manager has a number of basic capabilities: It can *install* software, *update* software packages, *remove* packages, *build* software packages, *query* and *verify* software packages.

Please see section 6.10 on page 185 in the *Linux Training Materials Project* (the workshop notes we have used in the lab) for more information about using RPM to query and verify software packages.

Building RPM Packages If anyone is interested in building your own RPM packages, you are very welcome to see Nick, he may even give you a book about it! The online book is called *Maximum RPM*, and is good reading for those who are interested.

If you find an RPM package for another version of Linux, it is a good idea to rebuild it, since that will solve library dependencies. For this, you use the *source* RPM package. To do this is simple. For example, suppose you download the latest Emacs package from `ftp://rawhide.redhat.com/pub/redhat/linux/rawhide/SRPMS/SRPMS/emacs-21.1-3.src.rpm` and want to use it in Red Hat 7.2 instead of the older version there. All you need do is:

```
$ sudo rpm --rebuild emacs-21.1-3.src.rpm
```

You will see that it requires the `gettext` package, so you can download and rebuild that also. The code compiles, and eventually creates the binary RPMs, which you can then install:

```
$ sudo rpm -Uhv /usr/src/redhat/RPMS/i386/gettext-0.10.40-3.i386.rpm
$ sudo rpm -Uhv /usr/src/redhat/RPMS/i386/emacs-*21.1-3.i386.rpm
```

Hmm, nice, Emacs version 21 is better!

Table 1 on the next page shows the basic RPM operations that we will use today. Table 2 on page 4 shows more information about how to make queries using RPM.

3 Procedure

Note that updating the kernel requires that you configure `lilo` with the new kernel, or the machine will not boot. We will use two methods; a manual method in which you manually resolve software package dependencies, and an automatic method using `rpmfind`

There are three steps in the manual method:

- Mount the network drive
- If there are kernel updates that need to be applied, apply them first, installing any dependent packages first.
- Apply the remaining updates, first those specialised to the Pentium III, and secondly, the remaining updates.
- Finally configure `lilo`.

Here is how we do it.

3.1 Mounting the network drive

1. Type:

   ```
   $ sudo mount nicku.org:/var/ftp/pub /mnt/ftp
   ```

Command	*Action*	*Description*
`rpm -i`	install	install the software package; do not uninstall any previous version of the software. Use this when installing kernels; normally, use the `upgrade` action.
`rpm -U`	upgrade	if an older copy of the package has been installed, uninstall it and replace it with this newer package. Do not delete the configuration files from the old package. Otherwise, it is the same as the `install` action, and is what I use most often.
`rpm -F`	freshen	install this package only if an older version of this software package has already been installed. Otherwise, do nothing. Useful when upgrading software packages.
`rpm -e`	remove	will completely erase the software package.
		Useful "helper" options
`-h`	hashes	Print fifty hashes ('#') to show progress of installation
`-v`	verbose	print the name of each package as it is installed. Useful when installing more than one package at a time. In practice, I always use this option, as well as the `-h` option.
`--force`	force `:-)`	For updates, go ahead and force the update, even if the same package or a newer one is already installed. Be very careful in using this; understand what you are doing. It can also override dependencies, and you can end up with a system that does not work if you use this option carelessly.

Table 1: The RPM options that we use to install and upgrade software.

> Now you have mounted the directory `/var/ftp/pub` from the machine `ictlab` over the network to the local directory `/mnt/ftp`, using the NFS protocol. If you change to your local directory `/mnt/ftp`, you will be accessing the directory `/var/ftp/pub` on `ictlab` over the network.
>
> We say that `ictlab` *exports* the directory `/var/ftp/pub` by NFS, and that your machine has *mounted* this on the local directory `/mnt/ftp`.

3.2 Installing Updates Manually

The Organisation of the Updates: The updates, as organised on Red Hat's ftp sites and mirrors, is in a group of directories:

```
$ ls
athlon  i386  i486  i586  i686  ia64  images  noarch  SRPMS
```

Most binary packages we need are in the `i386` directory. Some platforms benefit from packages that are compiled specifically for their architecture, so if you have a P-II or P-III, install the packages from the `i686` directory rather than the same packages from the `i386` directory. Table 3 on the next page summarises what is in each of the directories.

1. Now change to a directory on that network drive:

command	effect
`rpm -qa | less`	list all installed software packages
`rpm -q apache`	show the version of the apache package, if it is installed. Also use this to determine if a package is installed or not.
`rpm -qa | grep apache`	show all installed packages that have *apache* in their name
`rpm -ql apache`	list all files in the apache package
`rpm -qld apache`	list all documentation files in the apache package
`rpm -qlc apache`	list all configuration files in the apache package
`rpm -qi apache`	display information about the package
`rpm -V apache`	verify that the apache package is correctly installed
`rpm -qf /etc/passwd`	determine which package the `/etc/passwd` file belongs to

Table 2: This is a brief list of RPM query commands. I have used the `apache` package as an example.

directory	*What it's for*
`athlon`	Contains packages for ADM's Athlon processors
`i386`	Most of the packages you install come from here
`i486`	If a 486 will benefit from an updated package compiled for the 486, then it will be placed here
`i586`	Packages optimised for the Pentium
`i686`	Packages optimised for the Pentium-II
`ia64`	Packages compiled for the new 64-bit processors from Intel
`images`	These are updated installation disk images to put on a floppy disk using the `dd` command
`noarch`	These software packages contain no architecture specific code, and are used on all platforms
`SRPMS`	The source RPM packages

Table 3: The directories in the updates directory.

```
$ cd /mnt/ftp/redhat-7.1/updates/i686
```

2. Let's try to install the kernel:

```
$ sudo rpm -ihv kernel-2*.rpm
Password:
error: failed dependencies:
        modutils >= 2.4.6 is needed by kernel-2.4.9-6
        mkinitrd >= 3.2.2 is needed by kernel-2.4.9-6
        tux < 2.1.0 conflicts with kernel-2.4.9-6
```

It didn't work, since the `kernel` package *depends* on the software packages `modutils`, `mkinitrd` and `tux`, so these must be installed first.

The command `rpm` is important to you. There is a whole book about this command, called *Maximum RPM*, freely available on the documentation CDROM as the package `maximum-rpm-1.0-0.20010810.noarch.rpm`.

It is important to use the `-i` option when installing kernel packages. Otherwise, your system would lose the modules that it is currently using, and you would *have* to run `lilo` or your system would not boot.

The three options:

- `-i` **i**nstall the package. Install the package in any case, and *do not* uninstall any older version of the package if any exist.
- `-h` Print **h**ashes (#) to indicate progress.
- `-v` **v**erbosely print the name of each package as install it.

Then change to the directory with these packages and install them:

```
$ cd ../i386
$ sudo rpm -Fhv modutils* mkinitrd* tux*
error: failed dependencies:
        filesystem >= 2.1.0 is needed by mkinitrd-3.2.6-1
```

- `-F` **F**reshen the package. Install the package *only* if an older version of the same package is already installed.

3. Okay, so we have one more dependency. The package `filesystem` is in the `noarch` directory, so we install it together with the others:

```
$ rpm -Fhv modutils* mkinitrd* tux* ../noarch/filesystem*
```

At last, it worked! After all a packages dependencies are satisfied, it can be installed. You may have read that there is an option to `rpm` called `--force`, but do not use this unless necessary; a package installed without the packages it depends on will not work properly.

The package `filesystem` may not install correctly, since it cannot create directories in `/usr/local`. Note that currently, the directory `/usr/local` is managed by the automounter, so it cannot create directories there while the automouter is running. If you are doing this as a local user (but perhaps not if you are logged in with your LDAP account), you can turn the automounter off temporarily to install the `filesystem` package, then turn it back on when you have finished:

```
$ sudo service autofs stop
$ sudo rpm -Uhv ../noarch/filesystem*
$ sudo service autofs start
```

4. Now go back and install the kernel:

```
$ cd ../i686
$ sudo rpm -ihv kernel-2*.rpm
```

It is important to use the `-i` option when installing kernel packages. Otherwise, your system would lose the modules that it is currently using, and you would *have* to run `lilo` or your system would not boot.

`-i` **i**nstall the package. Install the package in any case, and *do not* uninstall any older version if any exist.

So at last, we have installed the kernel!

5. If you are using the excellent `grub` boot loader (default in RH 7.2), then you don't need to do anything else with `grub`, since the 7.2 kernel RPMs update `/boot/grub/grub.conf` automatically.

6. If and only if you are using `lilo`, to use the new kernel, you need to add it to your `/etc/lilo.conf`, then run `lilo`.

(a) I added a section to my `lilo.conf`, and changed the label for my old kernel, so that my own `lilo.conf` looks like this:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
default=linux

image=/boot/vmlinuz-2.4.9-12
        label=linux
        read-only
        root=/dev/hda9

image=/boot/vmlinuz-2.4.3-2
        label=linux243-2
        read-only
        root=/dev/hda9

other=/dev/hda1
         label=windows
```

(b) Now run lilo:

```
$ sudo lilo
Added linux *
Added linux243-2
Added windows
```

It's a good idea to leave the old kernel there as a boot option so that you can use it if the new kernel does not work for you for any reason.

Well now you have applied the update to the kernel, and to some other related packages.

Note: do not apply the other updates as described here. Use `rpmfind` instead, as explained in section 3.3 on the following page. I have provided this as a guide on how you *could* (but won't!) install the remaining updates manually.

There are some other updates to apply, and again, if you were continuing to use the manual method, you would need to sort out the dependencies. You would begin by

installing any other (non-kernel) packages from the `i686` directory. Finally (after installing all the packages from the `i686` directory), you would change to the `i386` directory and apply all the remaining updates with

```
$ sudo rpm -Fhv *.rpm ../noarch/*.rpm
```

3.3 Applying Updates Automatically: Setting up `rpmfind`

You might say, there has to be a better way. And there is: `rpmfind`. It is possible to automate the update process totally using `rpmfind`.

An Overview of Configuring `rpmfind`: There are a few steps to configuring `rpmfind`. You only need to configure it once; you can then run it regularly. This is an outline of what we will do here:

- install `rpmfind` if it's not already installed;
- create a local account for yourself (or use one you created previously);
- add this user to the `sudoers` file;
- update the login script of this account to include the directories `/sbin` and `/usr/sbin`;
- log into this new account;
- import Red Hat's public key that matches the key used to digitally `sign` the packages.
- finally, create a configuration file for `rpmfind`, and edit it;

The documentation for `rpmfind` is available online at `http://rpmfind.net/linux/rpmfind/` and `http://rpmfind.net/linux/rpmfind/autoupgrade.html`. You will need to refer to it.

Setting up `rpmfind`

1. Make sure that rpmfind is installed on your computer.

   ```
   $ rpm -q rpmfind
   rpmfind-1.6-5
   ```

   > In this case, it is installed. If you get a message like, "`package rpmfind is not installed`", then, well, the package needs to be installed.

   In Red Hat 7.x, it is part of the main Red Hat distribution. and is on one of the two installation disks.

   You can install it like this:

   ```
   $ cd /mnt/ftp/redhat-7.1/RedHat/RPMS
   $ sudo rpm -Uhv rpmfind-1.6-5.i386.rpm
   ```

2. Now create a local account for yourself, as described in section 7.14 on page 207 of the *Linux Training Materials Project*. You may use one of the accounts you created previously.

`rpmfind` will create a configuration file in your home directory. It is better to create this on a local account than on your network account, since the `rpmfind` configuration applies to your local machine.

3. Add your new account to the sudoers file as described in the handout on `sudo`.

4. Change the `PATH` on your new account to include the directories `/sbin` and `/usr/sbin` (see section 7.2 on page 195 of the *Linux Training Materials Project*.

```
$ sudo sh -c "echo 'export PATH=$PATH:/usr/sbin:/sbin' >> ~nickl/.bash_profile"
```

This appends the line

`export PATH=$PATH:/usr/sbin:/sbin`

to the end of the login script for the user `nickl`. Obviously, use the name of your own local account instead of `nickl`.

5. Log out and log into your new account.

6. Now you will import Red Hat's public key into GNU Privacy Guard. I assume you have `ictlab:/var/ftp/pub` mounted on `/mnt/ftp`.

```
$ gpg --import /mnt/ftp/redhat-7.2/i386/RPM-GPG-KEY
gpg: Warning: using insecure memory!
gpg: /home2/nickl/.gnupg: directory created
gpg: /home2/nickl/.gnupg/options: new options file created
gpg: you have to start GnuPG again, so it can read the new options file
$ gpg --import /mnt/ftp/redhat-7.2/i386/RPM-GPG-KEY
gpg: Warning: using insecure memory!
gpg: /home2/nickl/.gnupg/secring.gpg: keyring created
gpg: /home2/nickl/.gnupg/pubring.gpg: keyring created
gpg: key DB42A60E: public key imported
gpg: /home2/nickl/.gnupg/trustdb.gpg: trustdb created
gpg: Total number processed: 1
gpg:               imported: 1
```

GNU Privacy Guard is a very important tool for encrypting and signing data. It is often used for encrypting email, but even more often for verifying the `integrity` of data. When data is *signed* using `gpg`, the receiver can verify that there has been absolutely no change to the original data since it was signed by the person from whom the data came from.

We will use `gpg` to verify that the packages we install are genuine packages from Red Hat, and that they are not false packages containing Trojan programs created by bad people who want to crack your computer systems.

`gpg` is a free version of Pretty Good Privacy (`pgp`). `pgp` is described well in a book of that name. You can read more about `gpg` from `http://www.gnupg.org/`. `gpg` uses powerful public key encryption (together with other encryption technologies) to encrypt any data you want. Other people can decrypt it without you sending them any secret information. It is really quite remarkable, and very useful.

7. You need to run `rpmfind` for the first time to create the configuration file in your home directory.

   Here, we ask `rpmfind` to list its options:

```
$ rpmfind -h
```

This will take some time, since we are behind a firewall and `rpmfind` will be unable to reach the Internet servers until we set a proxy. That will be the next step.

8. Now `rpmfind` has created a configuration file in your home directory: ∼`/.rpmfind` Edit this file:

```
$ emacs .rpmfind &
```

9. Here is the output of `diff -u` showing the changes I made to my ∼`/.rpmfind` file:

```
--- /home2/nickl/.rpmfind~      Fri Jan  4 08:36:47 2002
+++ /home2/nickl/.rpmfind       Fri Jan  4 08:41:00 2002
@@ -31,10 +31,10 @@
 ; Where to lookup for autoupgrades
 ; multiple local or FTP directories can be specified
 ;
-autoupgradeURL=
+autoupgradeURL=/mnt/ftp/rh-7.1-updated/RedHat/RPMS

 ; your HTTP proxy/cache if any: http://myhttpproxy/
-httpProxy=
+httpProxy=http://sheep.vtc.edu.hk:8080/

 ; your FTP proxy/cache if any: ftp://myftpproxy/
 ; If authentication is needed use ftpProxyUser and ftpProxyPasswd
@@ -43,7 +43,7 @@
 ftpProxyPasswd=

 ; Default mode upgrade, latest or lookup (default)
-mode=lookup
+mode=upgrade

 ; Run in automatic mode: yes or no (default)
 auto=no
@@ -52,7 +52,7 @@
 nodelete=no

 ; Be paranoid when checking signatures: yes or no (default)
-paranoid=no
+paranoid=yes

 ; Run in overwrite mode: yes or no (default)
 overwrite=no
```

3.4 Running `rpmfind`

1. Now you are ready to run `rpmfind`:

```
$ sudo rpmfind --autoupgrade
```

> I found that it become stuck with some unresolved dependencies. You may need to resolve these manually! If so, install the set of packages that depend on each other, then run `rpmfind --autoupgrade` again until all the updates are applied. I found that the version of `rpmfind-1.7-2` with Red Hat Linux 7.2 works much better.

You are finished. Your system is much more secure against attack by crackers and malicious programs such as the Ramen Worm (`http://www.redhat.com/support/alerts/ramen_worm.html`).

At home, you can either:

- In our lab, burn a CDROM containing the updates and take them home, or
- download them from a Red Hat Mirror such as `ftp://rufus.w3.org/linux/redhat/updates/7.2/`

  > Note that a list of Red Hat mirror sites is available from `http://www.redhat.com/mirrors.html`.

3.5 How to Automatically Download the Updates?

For sites with a number of Linux machines: a site where you have many Linux computers, a scheme like this is a good way to maintain updates.

Here is a single line script to download updates for Red Hat 7.2 and Red Hat 7.1 to the local directories ~`ftp/pub/redhat-7.2/updates` and ~`ftp/pub/redhat-7.1/updates`.

Note that if you are using Red Hat 7.1, you should apply updates for Red Hat 7.1.

```
#! /bin/sh

rsync -avz \
    --delete \
    rufus.w3.org::linux/redhat/updates/7.2/en/os/ \
    ~ftp/pub/redhat-7.2/updates

rsync -avz \
    --delete \
    rufus.w3.org::linux/redhat/updates/7.1/en/os/ \
    ~ftp/pub/redhat-7.1/updates
```

If you saved this script as `/root/bin/rsync-redhat-7.x-updates`, then this `crontab` entry for root will run it at 7.31 am and 5.31 pm:

```
31 7,17 * * * /root/bin/rsync-redhat-7.x-updates
```

You would create this crontab entry for the user `root` with the command:

```
$ sudo crontab -e
```

This solution uses the important `rsync` protocol, described in my handout on burning CDROMs. The advantages of `rsync` include that it only transfers only the differences between files. It provides a very efficient way of synchronising files between two networked computers.

Refer to section **??** on page ?? in the *Linux Training Materials Project* for more about `cron` and the `crontab`. Also do

```
$ man 5 crontab
```

to see what each of the fields in the `crontab` entry represent.

For only one Linux computer: You can simply use `rpmfind` to directly download and install the updates itself. There are other choices too; the `up2date` program is incredibly convenient for interactive updates. First run the program `rhn_register` one time, then after that run `up2date`. Another good choice is the Red Carpet program that comes with the Ximian Gnome desktop. See `http//www.ximian.com` for more details.

3.6 Running rpmfind automatically

You can have `rpmfind` automatically update your computer by adding this to root's `crontab`:

```
# at 8.03 am every day, install new updates if they are available.
3 8 * * *       /usr/bin/rpmfind --autoupgrade
```

For this to work, the path `autoupgradeURL` in your ∼`/.rpmfind` file should point to the directories where the updates are downloaded, probably mounted by NFS, and either mounted permanently, or perhaps mounted on demand by the automounter.

It is important to ensure that root has a secure ∼`/.rpmfind` configuration. In particular, you should have GNU Privacy Guard set up for root with the Red Hat public key, and `rpmfind` should have the `paranoid=yes` setting to make sure that you are installing trusted packages and not those created by crackers.

3.7 Remote upgrade with rpmfind

One problem system administrators face is upgrading a production system. Normally, upgrading the OS (say from Red Hat 7.0 to 7.2) would involve taking the machine out of production for a couple of hours, rebooting it from a kickstart boot disk or CDROM.

I have use `rpmfind` to upgrade our laboratory server from Red hat 7.1 to Red hat 7.2 while it was online, and with no physical access to the computer. I upgraded it from my home on the weekend. This is not recommended by Red Hat, and you need to know what to do if you have any problems, but it was smooth for our machine, and is viable. The Debian Linux distribution has a much better system for doing updates and upgrades called `apt-get`. `rpmfind` is not as good, but it does the job.