Notes on Creating RPM Files

29 Jul 2008

Notes on Creating RPM Files

29 Jul 2008

1 Introduction

This document contains background information on RPM files and how to create them. It used to be part of the release notes, but was split out to make the release notes easier to navigate.

2 Introducing RPM Files.

An RPM file is a file that can be installed using RedHat Package Manager on an RPM-based Linux distribution, such as RedHat Linux, Fedora, CentOS and SuSE Linux.

They can also be installed on Debian systems, using the `alien` tool.

An RPM file can be created on any Unix system with the `rpmbuild` tool installed; it does not have to be an RPM-based Linux distribution. The `rpmbuild` tool takes an RPM Spec file (suffix `.spec`) and some user-specific settings, to create either source or binary RPMs. Source RPMs contain source code and the instructions needed to build a binary RPM, binary RPMs contain binary artifacts. SmartFrog is only released as a binary RPM.

2.1.Key RPM concepts

2.1.1 A package can own a file or a directory

The RPM `.spec` file lists which package owns a specific file or directory. If a directory is owned, when the package is removed, so is every file in it.

If a file is owned, then the file is checksummed during installation, so that the OS can see if it has been overridden by any other program.

2.1.2 Any attempt to upgrade over a modified 'owned' file is an error

If you try and upgrade an RPM and one of its owned files has changed, the package manager gets upset. There is one exception -configuration files.

2.1.3 A configuration file can be edited

The only owned file that it is not an error to edit a configuration file. We mark two files as configuration files, `default.sf` and `default.ini`:

```
%config(noreplace) %{bindir}/default.ini
%config(noreplace) %{bindir}/default.sf
```

If either of these file is edited, upgrades of the SmartFrog installation *will not overwrite these files*.

3 How to write an RPM Spec File

For details on writing RPMs, the following links are handy:

- http://www.rpm.org/
- http://www.rpm.org/max-rpm/
- http://www-128.ibm.com/developerworks/linux/library/l-rpm3.html?dwzone=linux
- http://www.ibm.com/developerworks/library/l-rpm1/

© Copyright 1998-2008 Hewlett-Packard Development Company, LP

- http://www.ibm.com/developerworks/library/l-rpm2/
- http://www-128.ibm.com/developerworks/linux/library/l-rpm3.html.
  This covers the process during an upgrade

JPackage and layout rules:

- http://www.jpackage.org/develdocs.php
- http://www.pathname.com/fhs/ rules for application and directory layout

We do not follow the JPackage rules or process, which would imply a separate RPM for every single dependent artifact (such as log4J), and the use of their special command line tools (which aren't installed on some systems) to set up the classpath. There are some fundamental differences in opinion between Steve and the JPackage team about versioning, namely their unwillingness to allow more than one version of a JAR to exist on a system.

Although insecure SF installations could use JPackage, once you start trying to sign JARs, it becomes impossible to work with the JPackage world view. Signing a JAR changes its checksum, so the RPM tools would get unhappy.

Presentations:

- http://www-uxsup.csx.cam.ac.uk/talks/rpmbuild/rpmbuild.pdf
- http://www.gurulabs.com/GURULABS-RPM-LAB/GURULABS-RPM-GUIDE-v1.0.PDF

If this is meant to scare people from creating RPMs, then do not be afraid. It is meant to scare people from making blind changes to our `.spec` file(s). Note that in our build process, we first copy the `.spec` files and expanding all currently defined Ant properties. This lets us manage customisations from our build process, and from `build.properties` options.

3.1.Customising the RPM

In the directory `core/release/metadata/rpm/rpm.properties` is the properties file setting up properties that are expanded in the RPM file before `rpmbuild` is invoked. Any of these properties can be overridden in a specific `build.properties`.

The other customisation is to include signed JAR files in the RPM, so that the installations are secure. This can be done with the redistributable RPM package.

4 Adding a new package to the RPM

Do not do this as a last minute activity; it should be done the day before a release to stabilise.

1. Get the izpack package working.
2. Open a new JIRA issue.
3. edit `metadata/rpm/smartfrog.spec`. Remember that it gets copied with an Ant property expansion before being run through `rpmbuild`; If there is any variable in the RPM .`spec` file, use Ant properties, setting the default values in `metadata/rpm/rpm.properties`
4. Add a new package under the existing packages, which should look a bit like the following, albeit with a real package name.

```
%package something
Group:         ${rpm.framework}
Summary:        Example package
Requires:       %{name} = %{version}-%{release}
%description something
Insert meaningful description here
```

5. This declares the package exists and that it depends on the same version of SmartFrog.
   -If you want to depend on other RPMs, learn the `requires:` syntax.

© Copyright 1998-2008 Hewlett-Packard Development Company, LP

-If you want to release on a different schedule from SmartFrog, remove the `= %{version}-%{release}` clause, to declare that you depend on *a* version of SmartFrog, but not which version you require.

6. Add a files section

```
%files something

%{libdir}/sf-something-${smartfrog.version}.jar
%{libdir}/something-${something.version}.jar

%{linkdir}/sf-something.jar
%{linkdir}/something.jar
```

7. Add a new entry in the `%changelog` section at the bottom of the file.

You then need to edit `build.xml` to set up the artifacts and symlinks for the new package.

8. Find the target `create-link-dir`.
9. Insert the tasks to declare a new RPM package,

```
    <rpmpackage package="scripting"/>
```

10. Add symlinks for every JAR that you are including in the links dir:

```
    <ln artifact="sf-something" version="${smartfrog.version}"/>
    <ln artifact="something" version="${something .version}"/>
```

11. Run this target (Linux only) to make sure that the symlinks are created. They do not need to resolve; they are only expected to resolve on a new installation, when the JARs are in their RPM-managed locations.
12. Add new tests to the `rpm-queries-test` target. We recommend one for every non-optional JAR that is installed. Check the links too. This target runs on a remote machine, to verify the files are present, post-installation.
13. Rebuild the RPM. `ant clean rpm`
14. Test the RPM by uploading it to the remote machine and testing for the artifacts:

```
rpm-remote-test
```

15. Extend the release note templates by mentioning the new RPM and what it does.
16. Commit the changes against the JIRA issue
17. Mark the issue as fixed.

5 Testing the RPM

5.1.Expanding the RPM for local examination

On Debian systems, the alien tool can be used to examine an RPM. The target `alien-rpm` does this.

```
release.alien-rpm:
     [exec] Warning: alien is not running as root!
     [exec] Warning: Ownerships of files in the generated packages will probably be wrong.
     [exec] Directories smartfrog-3.11.slo and smartfrog-3.11.slo.orig prepared.
```

Similarly, on an RPM-based system. the target `rpm-expand` can expand an RPM.

Don't try and install things, these are not Debian/Ubuntu friendly distros, primarily because they contain assumptions about the layout of directories under `/etc` and which scripts get run on a login, assumptions that don't hold on Ubuntu.

5.2.Uploading the RPM to a machine for testing

The Ant target `rpm-upload` will upload (precreated) RPMs to a target system running `sshd`. These files can then be installed by hand or by the build itself. The process is controlled by the file `metadata/servers/rpm.$`

{rpm.server}.properties, where ${rpm.server} is set to the host name for the target upload.

Here is an example, rpm.skye.properties:

```
rpm.ssh.server=16.25.169.163
rpm.ssh.user=smartfrog
rpm.ssh.dir=rpms
rpm.ssh.keyfile=${user.home}/.ssh/rhel.private
rpm.ssh.passphrase=
rpm.ssh.verbose=true
rpm.ssh.trust=true
```

As long as the matching public key is added to .ssh/authorized_keys on the target user on the target host, this configures the build to upload to a VMWare image that can install the RPM:

```
rpm-upload:
     [echo] SCP target is 16.25.169.16
  [ssh-rpm] Connecting to 16.25.169.163:22
      [scp] Connecting to 16.25.169.163:22
      [scp] Sending: smartfrog-3.11.slo-3.noarch.rpm : 11044293
      [scp] ................................................ 50%
      [scp] ................................................ 100%
      [scp] File transfer time: 572.34 Average Rate: 19,296.57 B/s
      [scp] Sending: smartfrog-daemon-3.11.slo-3.noarch.rpm : 4992
      [scp] File transfer time: 0.26 Average Rate: 19,200.0 B/s
      [scp] Sending: smartfrog-demo-3.11.slo-3.noarch.rpm : 622554
      [scp] ********* 100%
      [scp] File transfer time: 31.58 Average Rate: 19,710.43 B/s
      [scp] done..
```

Incidentally, such very slow network speeds between a Linux host and client is a common VMWare 5 problem, which can be fixed on the host by turning off some features of Intel network cards before VMWare 5 starts:

```
ethtool -K eth0 tx off
ethtool -K eth0 sg off
ethtool -K eth0 tso off
```

VMWare 6 appears to fix this.

Once uploaded, the files can be installed. The target rpm-remote-install, does this, provided the root login on the target machine has the same authorized key as the rpm.ssh.user.

The RPMs can be uninstalled using rpm-remote-uninstall:

```
rpm-remote-uninstall:
  [rootssh] Connecting to dhcp-169-107.hpl.hp.com:22
  [rootssh] error:
  [rootssh] package smartfrog is not installed
  [rootssh] error:
  [rootssh] package smartfrog-daemon is not installed
  [rootssh] error:
  [rootssh] package smartfrog-demo is not installed
  [rootssh] Remote command failed with exit status 3
```

Failures to uninstall are ignored, as this is the also result when nothing is installed. The target rpm-remote-uninstall-strict does report an error on a failure to uninstall, so can be used to test an uninstallation:

```
ant rpm-remote-uninstall-strict
```

There is a task to do a combined install, uninstall and test :

```
ant rpm-remote-test -Drpm.server=skye
```

This does a complete remote RPM upload:

1. Forced uninstall of the old artifacts
2. Install of the new RPMs
3. A call to sfVersion to check the new version number,

© Copyright 1998-2008 Hewlett-Packard Development Company, LP

4. The `/etc/init.d/smartfrogd` daemon is walked through its lifecycle; start, stop, status, etc., including checks that state changes are idempotent.

5. A non-forced uninstall of the RPMs. If this target fails, it means that the RPM install or uninstall is broken, and the product *must not be released.*

What we don't check is that an update works. This process is tricky to get right, so should have a test. We just need to work out how. For now, do it by hand on a VMWare image with the old version installed.

5.3.Testing the initd daemon

To test that the daemon works, the target `rpm-remote-initd` installs the `smartfrog-daemon` RPM then pushes the daemon through its lifecycle, starting and stopping the daemon, checking its status, and trying to restart it. If this target is successful, it implies that there are no obvious syntax errors in the shell script `release/scripts/etc/rc.d/init.d/smartfrogd` . If the tests fail, uninstalling the daemon may be harder than normal, as it means that the `smartfrog-daemon.rpm` uninstallation script will probably fail. Uninstall the `smartfrog-daemon` RPM using `--noscripts` to work around this

5.4.Finding RPMs

To list all SmartFrog RPMs on a system, use `rpm -q:`

```
rpm -q smartfrog smartfrog-daemon smartfrog-logging smartfrog-anubis smartfrog-demo
```

This prints a list

```
[smartfrog@skye smartfrog]$ rpm -q smartfrog smartfrog-daemon smartfrog-logging smartfrog-
anubis smartfrog-demo
smartfrog-3.11.007beta-1
smartfrog-3.12.003dev-2
smartfrog-daemon-3.12.003dev-2
smartfrog-logging-3.12.003dev-2
smartfrog-anubis-3.12.003dev-2
smartfrog-demo-3.12.003dev-2
```

The Ant target `rpm-remote-list` will perform this action

```
release.rpm-remote-query:
  [rootssh] Connecting to dhcp-169-107:22
  [rootssh] smartfrog-3.12.003dev-2
  [rootssh] smartfrog-daemon-3.12.003dev-2
  [rootssh] package smartfrog-demo is not installed
  [rootssh] smartfrog-anubis-3.12.003dev-2
  [rootssh] smartfrog-logging-3.12.003dev-2
```

These can be uninstalled using `rpm -e;` use `rpm -e --noscripts` to work around problems in scripts that are stopping an RPM from uninstalling.

If you have multiple versions of the RPMs installed, then you have a problem: the scripts will not auto-uninstall them. Go in by hand, uninstall them, then clean up. Do not try and install multiple versions, if you want to avoid this problem.

5.5.Directory layout

The RPM system sets files up according to the Filesystem Hierarchy Standard (http://www.pathname.com/fhs/) , the fedora guidelines, http://fedoraproject.org/wiki/Packaging/Guidelines, and the Linux Standard Base 3.1 specification (http://refspecs.freestandards.org/LSB_3.1.0/)

- Read only files go in to `/opt/smartfrog`
- Transient files (including logs) under `/var/opt/smartfrog`.

At some point we may wish to consider placing host-specific option files in `/etc/opt`. This would make `rpmlint` complain less, but means that the file structure of RPM-installed SmartFrog would be very different from that of

© Copyright 1998-2008 Hewlett-Packard Development Company, LP

all other installations.

6 RPM Security Issues

- The (unsigned) JAR files are not signed; when the daemon is running, anyone with access to port 3800 can deploy applications with root privileges.
- The log directory is world writeable, so anyone can edit the logs created by the daemon.

© Copyright 1998-2008 Hewlett-Packard Development Company, LP