BUILDING YOUR OWN RPMS

CONNECTING TO THE LAB EQUIPMENT

These steps will help you get started with the exercises:

1. Login to the desktop you are seated at as the user **`student`**, with the password **`student`**

BUILD FROM SOURCE

Historically, in the Open Source Software community, if you wanted to install a piece of software on your machine, you would download the source code archive from a repository, compile it, and install it yourself. Typically the process would look something like this:

- Download `FluffyMcAwesome-1.0.tar.gz` from provider
- `tar xf FluffyMcAwesome-1.0.tar.gz`
  Extracts archive contents into current working directory, usually creating a directory that matches the archive name, in our case, FluffyMcAwesome-1.0/
- `cd FluffyMcAwesome-1.0`
  Everything for this software package is stored relative to it's freshly unarchived directory.
- `./configure`
  Generally, there is a configure script which is used to verify the environment on your machine is sane (checks for things like the required compiler, libraries, interpreted languages, etc.) and potentially also generates a Makefile.
- `make`
  Compiles the software, but keeps all the compiled parts relative to the current working directory.
- `make install`
  Copies the compiled binaries and libraries from the local directory structure and copies them into the appropriate system directories.

The reason we use packages for software is that there are several big disadvantages in the method above.

Scale. If this software needs to be on 10 machines, this process is irritating to do. 100 machines obnoxious. 1000 machines, kill me now.

Maintenance. If you need to remove this software, is there a make uninstall? How do we keep other software from removing or overwriting the files that were provided by this piece of software? When it comes time to update, does FluffyMcAwesome-1.1 replace only those files that came with version 1.0, or does it abandon some files because they're not needed anymore? When upgrading the software, are you replacing a library or file that another piece of software relies on?

If you were to try and compile FluffyMcAwesome from source, your machine is missing several libraries required by FluffyMcAwesome. Let’s install those libraries:

```
[root@desktopX ~]# yum -y install ncurses-devel
```

ENTER THE PACKAGE

Because of the issues inherent with compiling from source, most distributions now opt for some type of package management. Red Hat Enterprise Linux uses RPM (RPM Package Manager). An RPM package is a cpio archive of file content with metadata to describe items like dependencies, tags for the file content, signatures and checksums for the payload of the package, etc. The installed RPM packages are tracked in a local database (`/var/lib/rpm`), so that when an RPM transaction is requested various checks can be performed. For example, is a new package installation trying to replace an existing file owned by another package; is a package removal going to violate the requirements for another still installed software package?

PACKAGE KNOWLEDGE QUIZ

This area is for you to scribble down some notes on things about RPMs that you find interesting or didn’t know before our game:

SETTING UP YOUR INITIAL BUILD ENVIRONMENT

We are going to be building packages as a non-privileged user, **student**. DO NOT build packages as root. During the build process, scripts and a few other things are executed on your machine. You do not want to run these types of things as root.

Lets start by creating some directory structure in the student user's home directory:

```
[student@desktopX ~]$ mkdir -p ~/rpmbuild/{SPECS,SOURCES,BUILD,RPMS,SRPMS}
```

`SPECS` – This is where our RPM `.spec` files will be placed, more on these files later.

`SOURCES` – The source code archives of our target software are put here.

`RPMS` – The compiled RPM packages will be dropped into this directory.

`SRPMS` – If we build any Source RPMs (SRPMs), they will be placed here.

`BUILD` – A directory to be used during the RPM build process; it is used as temporary storage.

There is a pre-existing RPM configuration setting that will look for a directory called `rpmbuild` in a user's home directory; the subdirectory structure is standard. Specifically, the `/usr/lib/rpm/macros` file has a macro defined, `%_topdir`.

Additionally, you will want to verify that the `rpm-build` RPM is installed on your machine.

```
[student@desktopX ~]$ yum list rpm-build
```

If the `rpm-build` RPM is not installed, as root, install it.

```
[root@desktopX ~]# yum -y install rpm-build
```

You'll also need any compilers and software that you would normally need when compiling software binaries, `gcc`, `make`, etc.

For our software, FluffyMcAwesome, we also need the `ncurses-devel` RPM installed. As root, install the `ncurses-devel` RPM.

```
[root@desktopX ~]# yum -y install ncurses-devel
```

Finally, download the source for FluffyMcAwesome and place it in the `SOURCES` directory:

```
[student@desktopX ~]$ wget
http://instructor.example.com/pub/materials/FluffyMcAwesome-1.0.tar.gz -O
~/rpmbuild/SOURCES/FluffyMcAwesome-1.0.tar.gz
```

OPENING THE .SPEC TEMPLATE

As of Red Hat Enterprise Linux 6, if you use either `vi` (with `vim-enhanced` RPM installed) or `emacs` to open a new `.spec` file, the editor will pre-populate the file with a template you can use to help generate the file's contents.

```
[student@desktopX ~]$ cd ~/rpmbuild/SPECS
[student@desktopX ~]$ vi FluffyMcAwesome.spec
```

We will be talking about the contents of the `.spec` file in the next few following sections. The template used by `emacs` is slightly different than the template provided by `vim`. The default contents of the template discussed throughout the document assume that it was opened using `vim`.

PREAMBLE

The preamble section of your `.spec` file contains a lot of basic information about the RPM that you are building. The following table provides information on the preamble items provided in the `.spec` template included by `vim` or `emacs`:

Preamble Item	Definition
`Name`	Name of the RPM you're building. For us, `FluffyMcAwesome`.
`Version`	The version of the software that is compiled. For us, 1.0 (pulled from OSS project source).
`Release`	Packager's 'version'. Pre-populated with 1%{?dist}, which will expand to 1.el6. This is a field that is there to help you, as the packager, keep your versioning straight.
`Summary`	One line descriptive blurb about the software provided by the RPM.
`Group`	Deprecated, used to classify this software according to available groups in `/usr/share/doc/rpm-*/GROUPS`.
`License`	The type of License that covers the software. Ex. GPLv2, Apache, BSD...
`URL`	Typically the URL for the OSS Project provided by the RPM.
`Source0`	Name of the archive of source code, generally in the `SOURCES` directory.
`BuildRoot`	Pre-populated in template. Used for packaging the software files into the RPM's file archive.
`BuildRequires`	Files or other software required to be on the machine before this software can be compiled into the RPM.
`Requires`	Files or other software required to be on a machine installing this RPM.
`%description`	A longer description of the software and/or features provided by the RPM.

Let's populate the preamble of our `FluffyMcAwesome.spec` file:

```
Name:           FluffyMcAwesome
Version:        1.0
Release:        1%{?dist}
Summary:        This is both Fluffy and McAwesome

Group:          Amusements/Games
License:        Copyright Red Hat, Inc. 2012
URL:            http://www.sourceforge.net/FluffyMcAwesome
Source0:        FluffyMcAwesome-1.0.tar.gz
BuildRoot:      %(mktemp -ud %{_tmppath}/%{name}-%{version}-%{release}-
XXXXXX)

BuildRequires:  ncurses-devel gcc
Requires:       /bin/bash ncurses

%description
FluffyMcAwesome is software that is both Fluffy and McAwesome.  Who doesn't
want Fluffy or McAwesome software?!?  Now you have both, congratulations.
```

%PREP

In the `%prep` stage the build environment is setup. In the default `.spec` template, it contains a call to `%setup -q` which unarchives the `Source0` archive into the `BUILD` directory. If your RPM referenced multiple sources in the preamble (`Source1`, `Source2`, etc.), they would also be unarchived in the `BUILD` directory. This portion of the `.spec` file is where it is also common to apply source code patches, or other activities to massage the source code or contents of the `BUILD` directory.

For our RPM build, we will not need to make any changes to the default `%prep` step.

%BUILD

`%build` is the portion of the `.spec` file that actually compiles the binaries that will later be pulled into our RPM contents. The default template `.spec` file has two commands in the `%build` portion, `%configure` and a `make` command. `%configure` is a macro that will run a `configure` script if one exists in the [now] unarchived source directory. If your source code does not have a `configure` script, you'll want to remove this macro call. The other command in `%build` provided by the template `.spec` is a `make` command. This will run the default target in your `{m,M}akefile` and provide any options indicated in the `%{?_smp_mflags}` macro. If your software requires more explicit build instructions, this is the section that you put the commands to compile your software, or organize it for the next step, installation.

We can use the the default `%build` section for our RPM.

%INSTALL

The default %install portion of the template .spec file includes the following:

```
%install
rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$RPM_BUILD_ROOT
```

The $RPM_BUILD_ROOT variable was set by the BuildRoot setting in the preamble, usually it's a directory created in /var/tmp. If this directory already exists, we'll remove it. Then we'll execute the install target in our software's {m,M}akefile. By using the DESTDIR argument to make, we ensure that when the make install is executed, instead of installing the files on our build machine, they will instead be placed inside our temporary $RPM_BUILD_ROOT directory. Later, when we package the files for the RPM's file archive, we will do it in a change rooted (chroot) environment relative to the $RPM_BUILD_ROOT. Essentially, what we are doing in this step is positioning the files inside $RPM_BUILD_ROOT directory so that later we can generate a file archive containing the software. When the archive is unpacked, the files will end up in the correct locations on the target system.

Looking at our software's make install target, files get placed in /usr/local/bin. This directory structure does not exist in the $RPM_BUILD_ROOT directory, so before we can run make install, we will need to generate this directory structure. Additionally, we would like to put the README file included with the software in the /usr/share/doc/FluffyMcAwesome-1.0 directory, but the make install target does not handle this file, so we'll have to place it manually into our archive. Please change the %install target in your .spec file to read:

```
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/usr/local/bin
mkdir -p $RPM_BUILD_ROOT/usr/share/doc/FluffyMcAwesome-1.0
make install DESTDIR=$RPM_BUILD_ROOT
cp README $RPM_BUILD_ROOT/usr/share/doc/FluffyMcAwesome-1.0
```

%CLEAN

%clean is executed when the RPM build process has completed. The default is to remove the $RPM_BUILD_ROOT to make sure that the temporary files we created in /var/tmp are removed from the system. You could add your own rules here if you needed to clean up other items on the build system that are byproducts of your compilation of this software.

%FILES

The %files section of the .spec file is also known as the file manifest. It will contain a list of all the files and directories that are provided by this RPM. The file manifest will be part of the database record for this software in the installed RPM database. Additionally, the files here will be checked against existing files on a target machine to ensure that there is not a collision, that is to say, that a new RPM would overwrite a file that is already on a machine provided by a different RPM.

The `%files` provided by the `.spec` template is as follows:

```
%files
%defattr(-,root,root,-)
%doc
```

`%defattr` is the listing of the default attributes that should be applied to the files included in the manifest, the expression in parenthesis next to the macro has four fields: (file permissions, user-owner, group-owner, directory permissions). In the default provided by the template `.spec`, files would be permissioned with whatever permissions are on the files when the RPM archive is created from `$RPM_BUILD_ROOT`. The files will be owned by root and the root group, and directory permissions would be the same as the default directory permissions used when the directories were added into the RPM file archive.

`%doc` is a stub to denote that as an RPM packager you can tag that some files are of specific types, such as documentation files (`%doc`), configuration files (`%config`), directories (`%dir`).

If you do not list the files that are copied into the `$RPM_BUILD_ROOT` in your file manifest, your RPM build will fail with an error similar to the following:

```
RPM build errors:
    Installed (but unpackaged) file(s) found:
   /usr/local/bin/FluffyMcAwesome
   /usr/share/doc/FluffyMcAwesome-1.0/README
```

Please change the `%files` section of the `.spec` file to read as follows:

```
%files
%defattr(-,root,root,-)
/usr/local/bin/FluffyMcAwesome
%doc /usr/share/doc/FluffyMcAwesome-1.0/README
```

To further customize the content of the file manifest, you can use the `%attr` macro. This tag is applied to individual files or directories to set permissions or ownerships on an individual file or directory. For example, if we wanted a file to be owned by the `apache` user, the `web` group, and permissioned `-rw-rw-r--` the file manifest listing for our file would look similar to:

```
%attr(0660,apache,web) /var/www/somefile
```

SCRIPTS

There are four main types of scripts we can embed in our package for manipulating the target machine, `%pre`, `%post`, `%preun`, `%postun`. `%pre` script contents are executed on the target machine before our software is unpackaged and installed. `%post` is executed after the software has been unpackaged and installed. `%preun` happens before an RPM erasure, before the software is uninstalled. `%postun` executes after the software has been removed from the target machine. Here are some examples of scripts from the `httpd` RPM (`rpm -q –scripts httpd`).

```
%pre
  # Add the "apache" user
  getent group apache >/dev/null || groupadd -g 48 -r apache
  getent passwd apache >/dev/null || \
    useradd -r -u 48 -g apache -s /sbin/nologin \
      -d /var/www -c "Apache" apache
  exit 0
```

Before the RPM is installed, using `getent`, the RPM checks for the existence of the `apache` user and group. If either is missing, it calls the appropriate command to add the respective user or group.

```
%post
  # Register the httpd service
  /sbin/chkconfig --add httpd
```

After the software has been installed, the init scripts for `httpd` are added to various runlevels by `chkcon-fig`.

```
%preun
  /sbin/service httpd stop > /dev/null 2>&1
  /sbin/chkconfig --del httpd
```

Before the software is removed from the machine, any running instances of `httpd` should be stopped. Then `chkconfig` is called to remove the `httpd` init script links from the rc directories.

We don't need any additional scripting added to our `.spec` to handle the machine state for our simple FluffyMcAwesome software.

%CHANGELOG

`%changelog` is a section of the `.spec` where we can add comments denoting what has changed in the RPM or the software packaged by this RPM. This is especially important in development environments, like Red Hat Enterprise Linux, where the OSS version doesn't change, but the package has backported fixes or modifications made to it by the package maintainer. Red Hat provided package `%changelog` entries will often mention a bugzilla ID number or CVE number associated with the fixes being imported in the package. Here are a couple of change log entries for reference:

Taken from `httpd` package (`rpm -q –changelog httpd`)

```
* Thu Oct 06 2011 Joe Orton <jorton@redhat.com> - 2.2.15-15
- core: add security fix for CVE-2011-3368 (#743659)
```

Taken from the kernel (`rpm -q —changelog kernel`):

```
* Tue Nov 08 2011 Aristeu Rozanski <arozansk@redhat.com> [2.6.32-219.el6]
- [kernel] KEYS: Fix a NULL pointer deref in the user-defined key type (David
Howells) [751190] {CVE-2011-4110}
- [scsi] fc class: fix building of Fibre Channel DUP drivers in 6.2 (Mike
Christie) [750268]
```

Notice that the `%changelog` information is slightly different between the two examples, however both of them start with a format of:

```
* WEEKDAY MONTH MONTHDAY YEAR MAINTAINERNAME <MAINTAINER EMAIL>

- DETAILS OF CHANGE
```

Add a `%changelog` entry to your `.spec` file for FluffyMcAwesome. Please change the `%changelog` section to read:

```
%changelog
* Thu Jun 28 2012 YOUR NAME <YOUR E-MAIL> - 1.0-1
- Initial build of FluffyMcAwesome 1.0 from source archive
```

RPMBUILD

Now that our `.spec` file is complete, it is time to build the RPM package for our software. To perform the compilation, we use the `rpmbuild` command, installed earlier from the `rpm-build` RPM. `rpmbuild` can be used in several ways, but we are going to use it to build the binary based RPM packages that will provide the FluffyMcAwesome software. If your `.spec` file has all the required items and is syntactically accurate, you should be able to build the `FluffyMcAwesome-1.0-1.x86_64.rpm` package by running:

```
cd ~/rpmbuild/SPECS; rpmbuild -bb FluffyMcAwesome.spec
```

You should see all the output from the `configure` script, the software compilation, and the package installation scroll by. If your `.spec` file has problems, the `rpmbuild` will likely halt with errors indicating where or what the problem is. If all went well, your package will be compiled and placed in `~/rpmbuild/RPMS/x86_64/`.

INSTALLATION

To install your brand new RPM, as root:

```
yum localinstall /home/student/rpmbuild/RPMS/x86_64/FluffyMcAwesome-1.0-
1.x86_64.rpm
```

You should now be able to run the `FluffyMcAwesome` executable and enjoy playing a Space Invaders like game!

COMPLETED FLUFFYMCAWESOME.SPEC FILE

```
Name:           FluffyMcAwesome
Version:        1.0
Release:        1%{?dist}
Summary:        This is both Fluffy and McAwesome

Group:          Amusements/Games
License:        Copyright Red Hat, Inc. 2012
URL:            http://www.sourceforge.net/FluffyMcAwesome
Source0:        FluffyMcAwesome-1.0.tar.gz
BuildRoot:      %(mktemp -ud %{_tmppath}/%{name}-%{version}-%{release}-
XXXXXX)

BuildRequires:  ncurses-devel gcc
Requires:       /bin/bash ncurses

%description
FluffyMcAwesome is software that is both Fluffy and McAwesome.  Who doesn't
want Fluffy or McAwesome software?!?  Now you have both, congratulations.

%prep
%setup -q

%build
%configure
make %{?_smp_mflags}

%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/usr/local/bin
mkdir -p $RPM_BUILD_ROOT/usr/share/doc/FluffyMcAwesome-1.0
make install DESTDIR=$RPM_BUILD_ROOT
cp README $RPM_BUILD_ROOT/usr/share/doc/FluffyMcAwesome-1.0

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,root,-)
```

```
/usr/local/bin/FluffyMcAwesome
%doc /usr/share/doc/FluffyMcAwesome-1.0/README

%changelog
* Thu Jun 28 2012 YOUR NAME <YOUR E-MAIL> - 1.0-1
- Initial build of FluffyMcAwesome 1.0 from source archive
```

REFERENCES:

```
rpmbuild(8), rpmlint(1),
Fedora Packaging Guidelines:
(http://fedoraproject.org/wiki/Packaging:Guidelines)
```