RPMs for Devs

martin.uy

Open by default.

Agenda

- Goals
- How do open source projects work?
- Introduction to RPMs
- Building RPMs
- A quick example: building, deploying and debugging the Linux kernel

Disclaimer

- I’m not an RPM nor a packaging expert, just a developer
- This is not a comprehensive RPM talk but more of a straight forward recipe
- This talk is based on Fedora Linux
  - It may be different in other distributions and package formats, but hopefully you will find some commonality

Goals

- Setup a quick development and debugging environment for any open source project (Linux kernel, NSS, Java, etc.)
  - Incremental builds
  - Debug with symbols and source code
  - Support building multiple projects in the same environment
- Only one recipe to rule them all: hide project specifics when building and installing
- Don’t taint our current environment with build dependencies
  - And don’t mix packages from different versions!

Open Source projects

- There is a community (upstream)
  - Generally sponsored by a company or a foundation
  - Governed by its authorities, structures and rules
  - There is usually a code repository, bug systems, mailing lists and IRC
- Source code is always available but not every community provide binary builds (for every architecture and operating system)
- Communities are open but some require signing agreements to accept major contributions

Open Source projects

THE LINUX FOUNDATION
KVM
redhat
WildFly
OpenJDK
ORACLE

Open Source projects

- There are Linux distributions (downstream)
  - Sponsor engineers to contribute to upstream communities
  - Take source code from upstream community repositories (vanilla source) and make a few changes to build, package and distribute
  - Why a “few changes”?
    - Integrate to their environment (i.e.: files or directories layout, crypto certificates location, configuration and admin tools, etc.)
    - Apply enhancements not in upstream yet (or that upstream has rejected for some reason)
    - Remove code due to license issues or platform-specific
    - Change build parameters
    - Other

Open Source projects

- Red Hat has a policy of “upstream first”, with some exceptions:
    - Service Level Agreements (SLAs) may require to deliver faster than what it takes for upstream to accept a contribution
- These exceptions are still open source: you can get them from the RPM source code repository which is publicly available
- Exceptions tend to be minimal (maintenance cost) and generally don’t modify APIs (it’s not a fork)

Open Source projects

- To contribute code to an open source project, you need to generate patches against master branch
- Building is different for each project:
  - dependencies?
  - ./configure? parameters?
  - make? cmake?
    - all?
    - build?
    - install?
  - README?
- You really need to read documentation for developers (or ask) and setup a proper environment.

Introduction to RPMs

- Binary format for source code and binaries packaging
- Created by Erik Troan and Marc Ewing (Red Hat), in 1997
- Used in many Linux distributions (and a few non-Linux ones)
- For multiple architectures
- Dependencies evaluation (to build or install)
- Delta RPMs (to speed up upgrades)
- Signature for integrity checks

Introduction to RPMs

- Linux distributions generally provide RPMs packages to download (from a package manager or HTTP)
  - Source packages (.src.rpm)
  - Binary packages (for each supported architecture)
  - Devel packages (headers)
  - Debug info packages (debug symbols stripped from built binaries and source code)

    Availability of these classes of packages depends on each case. I.e.: A “devel” package may not make sense for interpreted code.
- In addition, Fedora provides public read-access to its RPMs GIT repository
  - This repository is where RPM changes occur (package maintainers)

Introduction to RPMs

- RPMs download
  - http://mirror.globo.com/fedora/linux/development/rawhide/Everything/
- Git
  - https://src.fedoraproject.org/rpms/<package-name>.git
- fedpkg (Fedora) is a useful tool to work with RPMs and their repositories

Introduction to RPMs

- What does an SRPM package contain?
  - Source tarballs (vanilla sources) and “source” index file for integrity checking
  - RPM patches
  - SPEC file
    - Recipe (makefile-like) to build an RPM from an unpacked SRPM
  - Package dependent and auxiliary scripts
    - I.e.: script to build the source tarball from upstream repository

Introduction to RPMs

- SPEC file
  - Package information (multiple RPM packages may be generated)
    - Name
    - Description
    - License
    - Architectures
    - Version
  - Package dependencies
  - RPM patches (source code diffs)

Introduction to RPMs

- SPEC file
  - Stage instructions
    - %prep → extract source tarballs and apply RPM patches
    - %build → build patched source
    - %install → deploy to a BUILDROOT (final directories and files layout)
    - %clean → do cleanup
    - %post → do post-processing
    - %check → run smoke tests on built binaries
  - Changelog

Building RPMs

- fedpkg tool
  - `fedpkg clone -a <package-name>`
  - `fedpkg switch-branch <your-branch>`
  - `fedpkg sources`
  - `fedpkg srpm`
- Choose a branch equal to your deploy target (i.e. f25 for Fedora 25). This will simplify dependencies.
- At this point, package RPM source has been obtained and an SRPM (with vanilla sources inside) has been built from it.

Building RPMs

- Mock
  - Tool for building packages in a *chroot* environment
  - Safely and automatically manages build dependencies
  - Internally uses “dnf/yum” to get dependencies and “rpmbuild” tools to work with RPM packages.
  - Available in Fedora and CentOS. Can be built for RHEL.
  - Build for multiple distros and arches. I.e.: environment configuration to build for “Fedora 25 x86_64”.

Building RPMs

- Initialize a mock environment
  - `sudo /usr/sbin/usermod -a -G mock $(whoami)`
  - `mock -r fedora-25-x86_64 --rootdir=<path-to-chroot> --init`
- Install build dependencies in a mock environment
  - `mock -r fedora-25-x86_64 --rootdir=<path-to-chroot> --installdeps <path-to-srpm>`
- Mock can be used to build (`mock build`) but we will do it manually to get more control.

Building RPMs

- Prepare to build the package
  - `cd <path-to-chroot>/builddir`
  - `mkdir <package-name>_build`
  - `cd <package-name>_build`
  - `mkdir original`
  - `cp <path-to-srpm> original`
  - `mock -r fedora-25-x86_64 --rootdir=<path-to-chroot> --shell`

Building RPMs

- Prepare to build the package

  -
    ```
    mock -r fedora-25-x86_64 --rootdir=<path-to-chroot> --shell
    ```

    -
      ```
      export CURRENT_BUILD_PACKAGE=<package-name>
      ```

    -
      ```
      rpm --define "_topdir /builddir/${CURRENT_BUILD_PACKAGE}_build" -i /builddir/${CURRENT_BUILD_PACKAGE}_build/original/<package-srpm-file>
      ```
- At this point, the SRPM is unpacked in the build environment.
- Save original SPEC file

  -
    ```
    cp <path-to-chroot>/builddir/<package-name>_build/SPECS/<package-name>.spec <path-to-chroot>/builddir/<package-name>_build/SPECS/<package-name>.spec.bak
    ```

Building RPMs

- To increase speed, any build directory (`/builddir/${CURRENT_BUILD_PACKAGE}_build/BUILDROOT or BUILD`) can be replaced by a directory on *tmpfs* through symbolic linking.
  - Instead of slow HDD I/O, everything is written in memory
  - Requires large memory space available
- However, persisting build artifacts in BUILD directory may be interesting for incremental builds.

Building RPMs

- Prepare to build the package

  -
    ```
    mock -r fedora-25-x86_64 --rootdir=<path-
    to-chroot> --shell
    ```

    -
      ```
      export CURRENT_BUILD_PACKAGE=<package-name>
      ```

    -
      ```
      rpmbuild --define "_topdir /builddir/$
      {CURRENT_BUILD_PACKAGE}_build" -bp
      --target=`uname -m` /builddir/$
      {CURRENT_BUILD_PACKAGE}_build/SPECS/$
      {CURRENT_BUILD_PACKAGE}.spec 2> /builddir/$
      {CURRENT_BUILD_PACKAGE}_build/SPECS/$
      {CURRENT_BUILD_PACKAGE}_build_err.log |
      tee /builddir/$
      {CURRENT_BUILD_PACKAGE}_build/SPECS/$
      {CURRENT_BUILD_PACKAGE}_build_out.log
      ```

Building RPMs

- Prepare to build the package
  - At this point, prepare stage (%prep) has been executed. Vanilla source has been unpackaged and RPMs patches applied on top of it. This is the code that is going to be built.
  - Edit SPEC file:
    - Add “exit 0” after “%prep” line
    - Find any instruction that removes or cleanups files and comment it. I.e.: “make -s mrproper” in kernel.spec

Building RPMs

- Track source changes (optional)
  - `cd <path-to-chroot>/builddir/<package-name>_build/BUILD/<package-name>`
  - `rm -rf .git`
  - `git init`
  - `git add .`
  - `git commit -m 'dev_baseline_source'`
  - `git tag -a dev_baseline_source -m "dev_baseline_source"`

Building RPMs

- Build

  -
    ```
    mock -r fedora-25-x86_64 --rootdir=<path-to-chroot> --shell
    ```

    -
      ```
      export CURRENT_BUILD_PACKAGE=<package-name>
      ```

    -
      ```
      rpmbuild --define "_topdir /builddir/${CURRENT_BUILD_PACKAGE}_build" -bb --target=`uname -m` /builddir/${CURRENT_BUILD_PACKAGE}_build/SPECS/${CURRENT_BUILD_PACKAGE}.spec 2> /builddir/${CURRENT_BUILD_PACKAGE}_build/SPECS/${CURRENT_BUILD_PACKAGE}_build_err.log | tee /builddir/${CURRENT_BUILD_PACKAGE}_build/SPECS/${CURRENT_BUILD_PACKAGE}_build_out.log
      ```
  - RPMs will be written to <path-to-chroot>/builddir/<package-name>_build/RPMS

Building RPMs

- Incremental builds
  - Modify source code and re-run build command.
  - Objects that were not affected by file changes, are not re-built speeding up the whole process.

A quick example: kernel

- Before executing “prep” stage, modify kernel.spec file:
  - %define buildid .dev (no blank space before nor after “%”)
  - Disable signing (for x86_64)
    - %global signkernel 0
    - %global signmodules 0
  - In %build:
    - Comment “make -s mrproper” prepending a “#”
- Add the following options to “prep” and “build” rpmbuild commands:
  - --without debuginfo --without debug --without perf --without cross_headers --without headers --without doc --without tools

A quick example: kernel

- Before executing “build” rpmbuild command, modify <path-to-chroot>/builddir/kernel_build/BUILD/<kernel>/<kernel-2>/configs/<kernel> (i.e.: kernel-4.9.14-x86_64.config)
  - Change:
    - CONFIG_RANDOMIZE_BASE=n
    - CONFIG_RANDOMIZE_MEMORY=n
    - CONFIG_MODULE_SIG=n
    - CONFIG_MODULE_SIG_ALL=n
    - CONFIG_MODULE_SIG_UEFI=n
    - CONFIG_MODULE_SIG_SHA256=n
    - CONFIG_KEXEC_BZIMAGE_VERIFY_SIG=n
    - CONFIG_KEXEC_VERIFY_SIG=n

A quick example: kernel

- Extras
  - Eclipse is a good IDE for kernel dev + debugging (as a gdbserver front-end) in my experience
    - Source debugging is a bit tricky though, due to compiler optimization
  - QEMU is a good hypervisor for kernel debugging. It has a gdbserver stub. Had a few issues debugging boot stage.
    - Run QEMU image with “-s” parameter and attach gdb to port 1234.

References

- https://github.com/rpm-software-management/mock/wiki