RPM Packaging Workshop

Carl George
EPEL Team Lead at Red Hat

@carlwgeorge@fosstodon.org
@carlwgeorge.bsky.social
@carlwgeorge:fedora.im
carl@redhat.com

What is RPM?

- Native package format used by:
  - Fedora Linux
  - CentOS Stream
  - Red Hat Enterprise Linux
  - many others
- Consumed by package managers such as DNF

Why package with RPM?

- Easily install, reinstall, remove, and upgrade software
- Query and verify installed packages
- Metadata to describe package properties and relationships with other packages
- Digitally signed packages to validate authenticity
- Distribute packages in DNF repositories
- Pristine sources to ease future maintenance

What is an RPM package?

- Special archive containing files and metadata
- Two main types
  - **Binary RPMs** (a.k.a. packages, rpms) contain files to be installed on the target system
  - **Source RPMs** (a.k.a. source packages, srpms) contain software source code and instructions for building binary RPMs

What is an RPM spec file?

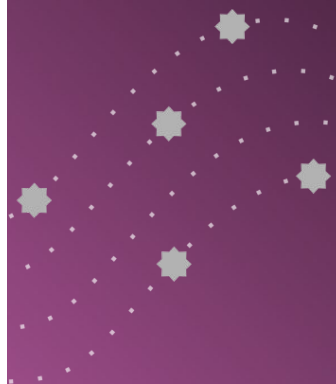
- Recipe for building a package
- Preamble that defines metadata about the package
- Build scriptlets corresponding to various stages of the build process
- File list and changelog
- Conditionals for flexibility between operating systems, operating system versions, architectures, etc.

RPM spec files

Spec file preamble

```
Name:           example
Version:        2.5
Release:        1%{?dist}
Summary:        Example software
License:        GPL-3.0-or-later
URL:            https://example.com
Source:         %{url}/example-%{version}.tar.gz

%description
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore.
```



Spec file build scriptlets

```
%prep
%autosetup

%conf
%configure

%build
%make_build

%install
%make_install
```

Spec file sections

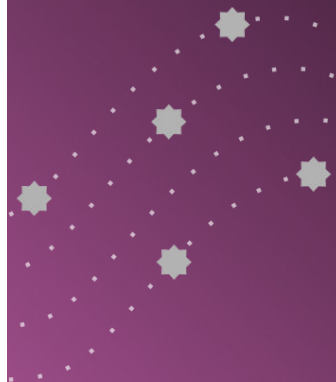
```
%files
%{_bindir}/example
%{_datadir}/example
%config(noreplace) %{_sysconfdir}/example.conf
%doc README.md CHANGELOG.md
%license LICENSE

%changelog
* Thu Feb 26 2026 Carl George <carlwgeorge@fedoraproject.org> - 2.5-1
- Update to version 2.5
```

RPM macros

RPM macros

- Variables for text substitution in a spec file
  - Syntax: `%example` or `%{example}`
- Some macros accept parameters to influence the output
- Can be defined inside the spec file or on the system
  - `/usr/lib/rpm/macros.d/macros.*`
  - `/etc/rpm/macros.*`
  - `~/.rpmmacros`



RPM macros

- Can be conditional to only expand when the macro is defined
  - `%{?dist}`
- Another conditional form is to insert text when defined
  - `%{?rhel:--disable-feature}`
- Can be explored outside of the build process
  - `rpm --eval '%example'` → evaluate a specific macro
  - `rpm --showrc` → print all defined macros

Common macros

- Filesystem paths
  - `%{_bindir}` → `/usr/bin`
  - `%{_datadir}` → `/usr/share`
  - `%{_sysconfdir}` → `/etc`
- Operating system properties
  - `%{rhel}` → `10`
  - `%{dist}` → `.el10`
  - `%{el10}` → `1`

Common macros

- Build process helpers
  - `%autosetup` ⟶ extract source code archives and apply patches
  - `%configure` ⟶ `./configure` with packaging-specific options
  - `%make_build` ⟶ `make` with packaging-specific options
  - `%make_install` ⟶ `make install` with packaging-specific options

Common macros

- Python build process helpers
  - `%pyproject_wheel` → wheel-based Python build
  - `%pyproject_install` → wheel-based Python install
- Test suite helpers
  - `%pytest` → `pytest`
  - `%tox` → `tox`

Initialize lab

Initialize lab

Open this link to load the lab environment.

Login with **any email** and the password `BuildRH!` and then click the **Lab User Interface** link.

red.ht/rpmlab

Workspace setup

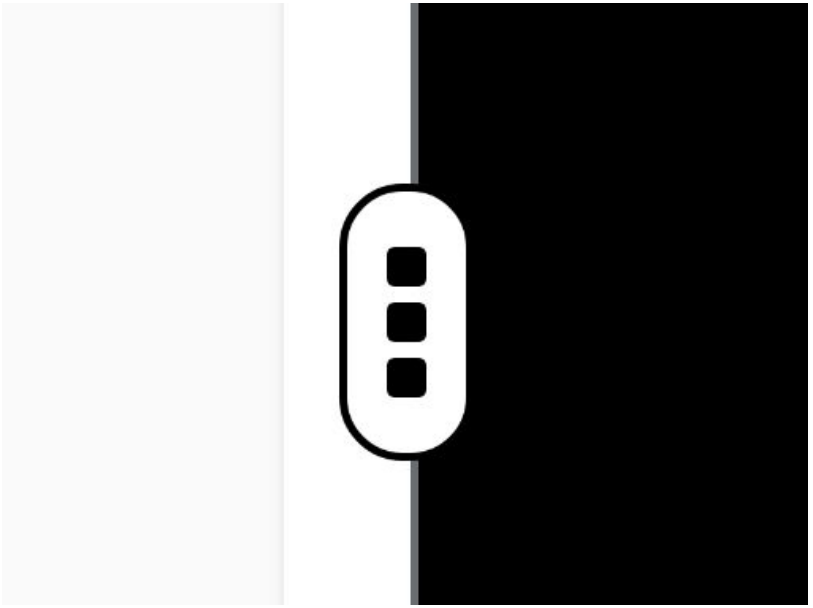
Workspace setup

- `rpmdev-setuptree` (from the `rpmdevtools` package) creates several directories
  - `~/rpmbuild/BUILD`
  - `~/rpmbuild/RPMS`
  - `~/rpmbuild/SOURCES`
  - `~/rpmbuild/SPECS`
  - `~/rpmbuild/SRPMS`

Lab tips

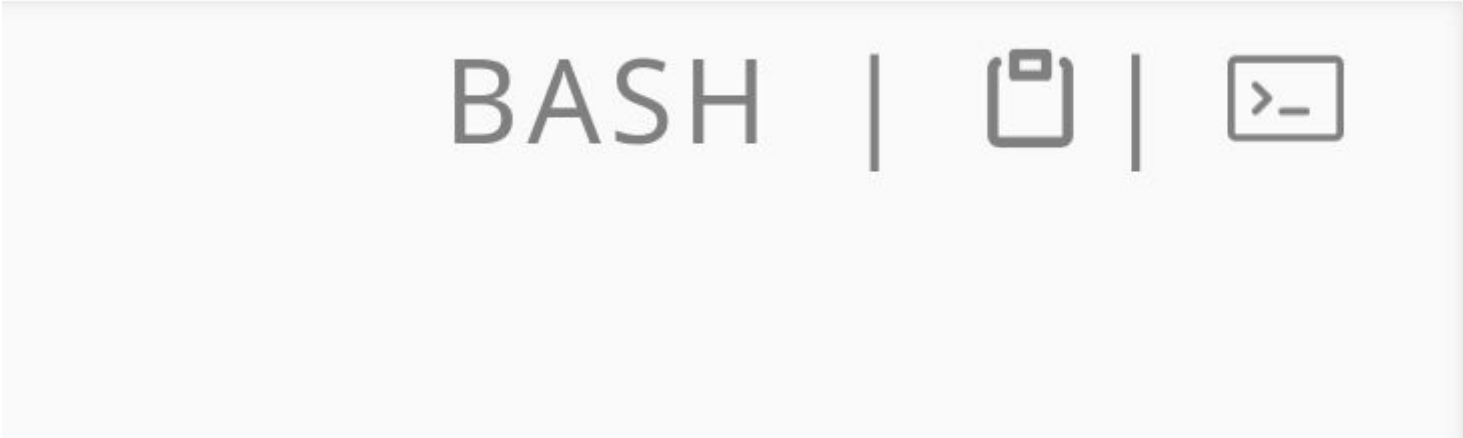
Lab tips

- The instructions and terminal panes can be resized using the widget with three dots



Lab tips

- Block with commands have icons at the top right
  - Clipboard icon to copy command
  - Terminal icon to run command



Lab: workspace setup

Lab: workspace setup

Your first challenge is to set up your packaging workspace.

Once you have completed the challenge, click the “**Next**” button.

RPM spec files, part 2

Spec file preamble

```
Name:           example
Version:        2.5
Release:        1%{?dist}
BuildArch:      noarch
Summary:        Example software
License:        GPL-3.0-or-later
URL:            https://example.com
Source:         %{url}/example-%{version}.tar.gz
```

Spec file preamble

- `Name:` → name of the package, should match the spec file name
- `Version:` → version of the software being packaged
- `Release:` → used to distinguish between different builds of the same software version
- Together these properties form an identifier known as the NVR
  - `gawk-5.1.0-6.el9`
  - `tzdata-2025c-1.el10`
  - `virt-what-1.27-4.fc43`

Spec file preamble

- `BuildArch:` ⟶ defaults to the build system architecture, can be set to `noarch` for packages with no architecture-specific files
- `Summary:` ⟶ short one-line summary of the software, 80 characters or less
- `License:` ⟶ identifier for the license of the software
- `URL:` ⟶ URL for more information about the software

Spec file preamble

- `Source:` → file name or URL of file needed to build the package, such as a source code archive or default configuration files
- Can be used multiple times
- Optionally suffixed with numbers
  - `Source0:`
  - `Source1:`
- Extracted in `%prep`, usually by `%autosetup`
  - Top-level set by `-n` flag, e.g. `%autosetup -n %{name}-%{version}`

Spec file preamble

```
%description
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.  Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.
```

Spec file preamble

- `%description` → longer description of the package
  - Can span multiple lines
  - Each line should be 80 characters or less

Spec file build scriptlets

```
%prep
%autosetup

%conf
%configure
```

Spec file build scriptlets

- `%prep` ⟶ commands to prepare the source code for building, such as unpacking archives and applying patches
- `%conf` ⟶ commands to configure the software for building

Spec file build scriptlets

```
%build
%make_build

%install
%make_install
```

Spec file build scriptlets

- `%build` → commands to build the software
- `%install` → commands to copy the desired build artifacts into a directory tree within the `%{buildroot}`

Spec file sections

```
%files
%{_bindir}/example
%{_datadir}/example
%config(noreplace) %{_sysconfdir}/example.conf
%doc README.md CHANGELOG.md
%license LICENSE
```

Spec file sections

- `%files` ⟶ list of files and directories that will be installed on the target system
  - Referenced by absolute path within the `%{buildroot}`
  - Each item can be preceded by an attribute

Spec file sections

- Common attributes in `%files`
  - `%config(noreplace)` → mark as a configuration file and prevent it from being overwritten on updates
  - `%dir` → own just the directory itself, but not its contents
  - `%doc` → mark as a documentation file
  - `%license` → mark as a license file

Spec file sections

- Most attributes only accept absolute paths
- The `%doc` and `%license` attributes also accept relative paths, which will copy the files from the `%{_builddir}` into the `%{buildroot}`
  - `%doc` ⟶ copy files into `/usr/share/doc/%{name}/`
  - `%license` ⟶ copy files into `/usr/share/licenses/%{name}/`

Spec file sections

```
%changelog
* Thu Feb 26 2026 Carl George <carlwgeorge@fedoraproject.org> - 2.5-1
- Update to version 2.5

* Mon Dec 01 2025 Carl George <carlwgeorge@fedoraproject.org> - 2.4-2
- Rebuild for libspelling soname bump

* Sat Jul 19 2025 Carl George <carlwgeorge@fedoraproject.org> - 2.4-1
- Update to version 2.4
```

Spec file sections

- `%changelog` → record of changes that have happened to the package between different versions and releases
  - Reverse chronological order

Creating spec files

- From scratch
- Copy a similar spec file and adjust as needed
- Automatic templates from a text editor
- `rpmdev-newspec` (from the `rpmdevtools` package) will create a new spec file from templates

Creating changelog entries

- From scratch
- Copy another changelog entry and adjust as needed
- Text editor plugins
- `rpmdev-bumpspec` (from the `rpmdevtools` package) will create new changelog entries and simultaneously adjust the version and/or release tags

Building RPMs

Building RPMs

- RPMs are built with the `rpmbuild` command
  - `rpmbuild` expects the directory structure from `rpmdev-setuptree`
- Various build modes
  - `-bs` → build an SRPM from a spec file and sources
  - `-bb` → build an RPM from a spec file and sources
  - `-ba` → build both an SRPM and an RPM from a spec file and sources
  - `--rebuild` → build an RPM from an SRPM

Inspecting RPMs

- `rpm` can query an uninstalled RPM file by using the `--package` flag
- Combine with other flags to inspect specific properties
  - `--info`
  - `--list`
  - `--requires`
  - `--provides`
  - `--conflicts`
  - `--changelog`

Lab: packaging bello

Lab: packaging bello

Your next challenge is to package `bello`, a program written in Bash.

Once you have completed the challenge, click the "**Next**" button.

RPM spec files, part 3

Spec file preamble

```
Source:         %{url}/example-%{version}.tar.gz
Patch:          bug-fix.patch
```

Spec file preamble

- `Patch:` → file name or URL of patch to apply to the source code
- Can be used multiple times
- Optionally suffixed with numbers
  - `Patch0:`
  - `Patch1:`
- Applied in `%prep`, usually by `%autosetup`
  - Patch prefix level is handled by `-p` flag, e.g. `%autosetup -p 1`

Spec file preamble

```
BuildRequires:  make
Requires:       curl >= 7.76.1
Recommends:     findutils
Supplements:    tree
```

Spec file preamble

- `BuildRequires:` → other packages needed to build this package
- `Requires:` → other packages needed to install this package
- `Recommends:` → weak requires, installed by default but can be removed or skipped
- `Supplements:` → reverse recommends

Installing build requirements

Installing build requirements

- `rpmbuild` needs the build requirements listed in the spec file to be installed on the build host
- Can be installed manually or with `dnf builddep`

Lab: packaging cello

Lab: packaging cello

Your next challenge is to package `cello`, a program written in C.

Once you have completed the challenge, click the "**Next**" button.

RPM spec files, part 4

Spec file preamble

```
Epoch:          1
Version:        2.5
Release:        1%{?dist}
```

Spec file preamble

- `Epoch:` ⟶ optional integer for overriding normal sorting order
  - Evaluated before version
  - Last resort to correct upgrade path
  - Can never be removed
  - Handle an upstream version scheme change
    - `2026.02` > `1.0.0`
    - `2026.02` < `1:1.0.0`
  - Handle forced downgrades
    - `5.6.1` < `1:5.4.6` < `1:5.6.2`

Spec file preamble

```
Conflicts:      alternate-example
Obsoletes:      legacy-example < 2
Provides:       legacy-example = %{version}-%{release}
```

Spec file preamble

- `Conflicts:` → other packages that cannot be installed at the same time
- `Obsoletes:` → used to replace one package with another
- `Provides:` → allows other packages to refer to this package by another name

Spec file build scriptlets

```
%check
%pytest
```

Spec file build scriptlets

- `%check` → commands to test the software, such as unit tests

Lab: packaging pello

Lab: packaging pello

Your next challenge is to package `pello`, a program written in Python.

Once you have completed the challenge, click the “**Next**” button.

Subpackage preamble

Subpackage preamble

```
%package devel
Summary:        Development files for %{name}
Requires:       %{name}%{?_isa} = %{version}-%{release}

%description devel
The %{name}-devel package contains libraries and header
files for developing applications that use %{name}.
```

Subpackage files

```
%files devel
%{_includedir}/example
%{_libdir}/example.so
%{_libdir}/pkgconfig/example.pc
```

Spec file subpackage preamble

- `%package <name>` ⟶ starts a preamble section for a subpackage
- `%description <name>` ⟶ description for a subpackage
- `%files <name>` ⟶ list of files for a subpackage

Mock

Mock

- There are drawbacks to using `rpmbuild` directly
  - Build requirements installed directly on build host
  - Build requirements that happen to already be installed on the build host are easy to forget to include in the spec file
  - Can only build RPMs targeting the same operating system and operating system version as the build host

Mock

- Mock is a tool that builds RPMs in isolated chroots
  - Uses `rpmbuild` internally
  - Build requirements are installed in the chroot, not the build host
  - Helps identify missing build requirements
  - Can build RPMs targeting a different operating system and operating system version as the build host
  - Chroots are automatically created and removed
- Widely used (koji, copr, fedpkg, and more)

Lab: building with mock

Lab: building with mock

Your final challenge is to build the `pello` package again, but using the `mock` tool this time.

Once you have completed the instructions, click the "**Next**" button.

Become a Fedora/EPEL packager

Interested in doing more? Consider becoming a Fedora and EPEL package maintainer.

red.ht/fedorapackager

Share the lab with your friends

Access the hands-on portion of the lab in the future.

red.ht/rpm

That’s all folks

@carlwgeorge@fosstodon.org
@carlwgeorge:matrix.org
carl@redhat.com