How to Modify a Package in the Open Build Service

Quilting with OSC

SUSE Linux Enterprise Server

Josef Moellers, Senior Developer SUSE Linux Enterprise Network Services, SUSE

This document leads you through the process of modifying a software package in the Open Build Service (OBS) using the **osc** and **quilt** tools. It also discusses simple error cases, based upon the author's own experiences, but it does not attempt to be a full manual or to cover all options. The steps described here should work well, but if you encounter any difficulties, you should consult the manuals or ask an expert for help.

This document does not intend to provide a guide for the Open Build Service. If you want to learn more about OBS, visit the project's Web page at http://openbuildservice.org/ and read the specific documentation there http://openbuildservice.org/help/

Publication Date: March 13, 2018

Contents

1 Introduction: Open Build Service and the Tools `osc` and `quilt`

The Open Build Service command line client `osc` is a tool developed to interact with OBS servers. It provides many functions including:

- Creating, modifying and deleting the information (meta data) about projects and packages and downloading or uploading their code and packaging
- Managing the repositories or targets that projects or packages build against
- Preparing and running local builds of packages
- Making and managing requests to modify packages in projects (so called *Submit Requests*)

`quilt` is a software utility for managing a series of changes to the source code of any computer program. Such changes are often called "patches" or "patch sets". `quilt` takes an arbitrary number of patches and turns them into a single patch. In doing so, `quilt` makes it easier for other programmers to test and evaluate the different changes before they are permanently inserted into the source code. In short, `quilt` allows you to easily manage many patches by keeping track of the changes each patch makes. Patches can be applied, un-applied, refreshed, and more.

To understand how `osc` and `quilt` work, a few terms and technologies need to be described more in detail:

- OBS manages *repositories*, such as **devel:tools**, **SUSE:SLE-15:GA**, or **home:<username>**. *Repositories* are sometimes also called *projects*. In this documentation, projects are abbreviated as **PRJ**.
- A *repository* contains a set of *packages*, such as **pam_krb5**, **ntp**, **wget**, **zypper**. In this document, packages are abbreviated as **PKG**. Keep in mind that a package can (and usually will) appear in more than one repository, but for all practical purposes each one is a copy in its own right.
- A package consists of a set of files, which contains:
  - at least a `spec` file,
  - an archive of *pristine sources* – these are the "original" sources for the package without any patches applied – possibly accompanied by a `signature` file,

- a `changes` file,
- possibly a set of patches which are to be applied in a given sequence, the order being defined by the `spec` file, and
- sometimes additional files, which are not part of the sources or which control some aspects of the building process, for example an `rpmlintrc` file.

While there exists only **one** Open Build Service, it maintains two separate collections, or APIs (as they are called in the documentation) of repositories:

- the *external* build service at https://api.opensuse.org. It can be used by everyone. When you log in to this Web site, you can view all projects available in the external build service.
- the *internal* build service where the SUSE products are maintained. Only SUSE employees can access this Web site using their log in credentials. Here you can view the existing projects in the internal build service.

Note: External Build Service

By default, the **`osc`** command works on the external build service only. To make it work on the internal build service, you need to add the special option with argument `-A int`. As this paper focuses on the external API of the Open Build Service, only the plain **`osc`** command is used.

This document guides you through the process of modifying a package by updating it to a new version, adding an existing upstream patch or by building a new patch.

You might need to add a patch to several versions of the respective package in more than one project. The reason is that, if you fix a bug, you often need to fix it in various SUSE Linux Enterprise releases. For example, if you fix a bug in SUSE Linux Enterprise 11, you might also need to fix it in SUSE Linux Enterprise 12. For this purpose we will use two tools:

- **`osc`** to manage the package(s) and
- **`quilt`** to manage the patch(es)

Using two tools occasionally requires switching from one tool to the other. At the “junctions”, each of the tools will “enter the realm” of the other tool. But in good Unix tradition, each tool performs its own job, while being aware of the other tool. The commands for both tools come

with a very extensive set of subcommands and very good manual pages. Options explained or mentioned in this document are usually placed after the subcommand. In consequence, for a correct command, type :

```
osc mbranch -c <PKG>
```

and NOT

```
osc -c mbranch <PKG>
```

The former command requests to check out files from the created branches, while the latter would specify to use an alternate configuration file **"mbranch"** with **<PKG>** as the subcommand.

Both tools come with a built-in help system. To display the information for a given subcommand, type:

```
osc help <subcommand>
```

or

```
quilt <subcommand> -h
```

2 Repositories and Projects

This chapter focuses on *repositories*. You will have to upload packages to *repositories* and move packages from one *repository* to another. The term *repository* is often used as a synonym for *project*. But strictly speaking a *project* is a set of packages and a *repository* is the location where the files for these packages are stored.

Note: Colons

When you see colons (":") in a repository (or project) name, notionally replace them by slashes ("/"), to make the file name look more familiar.

A project can be

- a software project (for example **Apache**, **Base:System**, or **Linux-PAM**)
- a home project (for example **home:jmoellers**)

- a branch (for example **home:jmoellers:branches:Linux-PAM**)
- a product project (for example **SUSE:SLE-12-SP3:GA)**

Important

Product project is not an official term!

You can list all projects with the command `osc ls /`. For the external build service, this command returns a rather long list of available projects (at the time of writing, almost 45,000 projects were available).

Each package included in a product usually goes through at least three repositories:

1. the *development repository*, or in short *devel repo*
2. *Factory*
3. the *product repository*

Often a fourth repository is needed, which is the *branch* that you create to work on the product. In these cases a package passes trough the following repositories:

- the *development repository*, or in short *devel repo*
- your *branch*
- once again the *devel repo*
- *Factory*
- the *product repository*

2.1 Upstream

Upstream is not a *repository* in the classical meaning, but usually it is the very basic source of a package. It could be a Git repository that you clone, or a Web site from where you download a TAR archive.

When you fix a bug, you should submit the patch to *upstream* to ensure it can be included into future versions of the package. This has the advantage that next time you upgrade the package from *upstream*, you might be able to drop one or more local patches. In addition, you contribute

your efforts to the community. However, getting the patch accepted might require some patience and persistence, because some upstream maintainers have very special requirements regarding how to fix a bug. But usually most *upstream* maintainers are grateful for any fixes.

2.2 Development Repository

The *development repository*, or in short *devel repo*, is where you keep a package locally, after it has been downloaded or cloned from *upstream*. Most *development repositories* are located on the external build service. Each *development repository* has one or more maintainers and one or more bugowners.

Only the maintainer(s) and the bugowner(s) can submit to this repository. Other contributors must branch the repository and then request the patch to be accepted by the maintainer(s) and/or bugowner(s). You can, however, check out from the *development repo*, for example, if you only want to have a look at the source code or if you need to analyze a problem without first creating a branch (which takes up some space on the build servers).

You can find the *devel repo* of a package by typing

```
osc develproject <PRJ> <PKG>
```

The command:

```
osc develproject openSUSE:Factory cvs
```

will display the following output

```
devel:tools:scm
```

Note: Short Version

You can abbreviate **develproject** with the short version **dp**.

2.3 Factory

The *Factory* project is the rolling development code base for the openSUSE distribution Tumbleweed. *Factory* is mainly used as an internal term for openSUSE's distribution developers, and the target project for all contributions to openSUSE's main code base. There is a constant flow of packages going into *Factory*. There is no package or feature freeze; therefore, the *Factory* repository is not guaranteed to be fully stable and is not intended to be used as a distribution itself.

The core system packages receive automated testing via openQA, the automated test tool for Linux operating systems (for more information, see http://open.qa/). When automated testing is completed and the repository is in a consistent state, it is synchronized to the download mirrors. Then it is published as the openSUSE Tumbleweed distribution, the rolling release version of the openSUSE project, featuring the newest technology (see https://www.opensuse.org/#Tumbleweed). It is of course also the basis for openSUSE Leap, the regular annual release from the openSUSE project, with security and stability being the main focus.

In addition, *Factory* is used as the base repository containing the code stream for the development of the next major version of SUSE Linux Enterprise and all related enterprise-class products. Therefore it is all the more important to keep this repository up to date.

Note: Community Project

openSUSE:Factory is part of the openSUSE community project. This means it resides outside of the official SUSE product projects, and is not regulated or controlled by SUSE. Package maintainers are usually members of the openSUSE community.

A major rule has been established which is called **"Factory First"** . In theory, this means that you need to submit any package that you want to see included in any version of openSUSE or SUSE Linux Enterprise to *Factory*. In practice, you will submit your package or patch to the *devel repo* first and then you will submit to *Factory* from there. As you branch and check out from the *devel repo*, `osc` will automatically submit changes to the *devel repo* master. Once you have submitted or updated a package in its *devel repo*, you (or the maintainer of the *devel repo*) must submit the package to **openSUSE:Factory**.

Important: Factory First Policy

After you have submitted or updated a package in the *devel repo*, you always need to submit it to the next step! As long as the package is updated or patched only in its *devel repo*, the new version of the package will not be included with any new release. Only when it is submitted to **openSUSE:Factory**, it is available for incorporation into future product releases. This policy is called **"Factory First"** !

From **openSUSE:Factory**, the package will later be submitted to a *distribution repository*, for example to **SUSE:SLE-15:GA**. Usually this is done by the package maintainers, but occasionally you might be asked to submit your new or updated package there.

2.4 Home Project

The *Home Project* is your private repository. It is called **home:<username>** (for example **home:jmoellers**). In this repository, you store your own projects, and you create your *branches*.

2.5 Branch

When you create a *branch* from an existing project, for example from a *devel repo* of which you are not the maintainer or bugowner, this is stored under **home:<username>:branches**. In the following example:

```
home:jmoellers:branches:network:utilities/wget
```

home:jmoellers:branches: stands for the branch, **network:utilities** stands for the **devel project**, and **wget** is the product.

Some maintainers or bugowners branch from their own devel repos, to ensure clean development processes, and to allow for proper documentation.

The `osc` command records from which repository the branch was created. When you submit changes back to your branch, it also makes a request to the maintainers of the package to accept and submit the changes into the original repository.

2.6 Product Projects

For the purpose of this document, the term *Product Projects* refers to the repositories containing the packages that are bundled into a SUSE product. When a new product is built, its repository usually is named **SUSE:<productname>:GA** (for example **SUSE:SLE-15:GA**). At a later stage of the lifecycle, after the General Availability (GA) of the product, when the service packs for a product need to be created, a repository **SUSE:<productname>:Update** is set up. This is the repository to which you now should submit your patches. The next service pack of a product will be built from this repository.

Note: GA Production Repository

You cannot directly submit to the GA production repository. As the **Factory First** rule applies, you submit to *Factory*. Then the product release team will fetch the package from there. You can, however, submit to the **Update** repositories when you fix a bug in an older release.

Each distribution and each service pack has its own repository (for example **SLE-xx-SPyy**). If you fix a bug in a package's *devel repo* (after which you submit the patch to **openSUSE:Factory**), you may need to also fix it in the *distribution repos*.

This requires the following steps:

1. Creating branches for all versions which are currently in *maintenance mode*:

   ```
   osc mbranch
   ```

2. Checking the branches out to your local disk. The previous step and this step can be combined into one step:

   ```
   osc  mbranch -c
   ```

3. Creating one or more patches and modifying the associated files (for example the **spec** file)

4. Submitting the changes back to the *product repo*

The last step will not automatically update the package in the *product repo*. The package maintainer will do that after some necessary tests.

3 Package

The next paragraphs focus on packages. A package consists of:

- a `spec` file describing how the binary package is to be built from the sources,
- the *pristine sources*, means the unmodified base source archive which you usually download from the upstream git repository, possibly accompanied by a signature file,
- a `changes` file containing a human readable list of the changes introduced with each patch, and
- (possibly) a set of patches to be applied, for example functional enhancements, adaptations to SUSE specifics or bug fixes

Other files might be included in a package, but they are not relevant for the purpose of this document. You can find all files of a package in the *source RPM*.

The command **`osc ls <PRJ>`** lists all packages of a project or repository.

Example:

```
osc ls devel:tools:scm
```

This command lists all packages in the **s**ource **c**ode **m**anagement section of the **devel**opment **tools**: from **EasyMercurial** through **bugzilla**, **cvs**, **git**, **osc**, **rcs**, **trac**, down to **xmlto**.

With the command **`osc search`** (or short **`osc se`**) you can find the project(s) a package is in:

Example:

```
osc se bugzilla
```

Besides listing the home projects of contributors who have created branches of the **bugzilla** package, this command returns the following information:

```
devel:tools:scm       bugzilla
```

4 Getting the Package

At a certain point in time, you need to modify a package. You

1. either have worked out a fix for a bug yourself,
2. or found an upstream patch that you want to include,
3. or you want to upgrade a package to a new version.

If you want to update a *package*, you should use the existing one as the skeleton. You also you must keep and amend the `changes` file. If you want to write a patch or use an existing patch, you need the current version of the package because you simply add your patch to the package.

At this stage, you should also have a clear idea of the directory structure you want to use. One option is to store all your patches and packages under your **HOME** directory. But you can also consider to create subdirectories by topics for the work you currently have in progress. In any case, the meta data that are stored by **`osc`** ensure that the package is submitted to the correct directory.

4.1 Development Repository – Maintainer or Bugowner

You can always check out the sources from the *devel repo*. But as long as you are not the maintainer or bugowner of that repository, you usually cannot submit your changes directly. The tool `osc` knows where a package comes from and will automatically submit it back to the correct repository.

After you moved to the directory where you want to work on the sources, you can check out the current revision of the package by running the command:

```
osc co <PRJ> <PKG>
```

where <PRJ> ist the project the package is part of and <PKG> is the package's name.

Example:

```
osc co devel:languages:perl perl-RPM-VersionSort
```

This command checks out a complete directory tree and creates **devel:languages:perl/perl-RPM-VersionSort**. Inside the **perl-RPM-VersionSort** directory, all required files are available: `spec` file, `source` archive, `changes` file, patches etc.

In case you have created several working directories and you try to check out a package you already have a working copy of, you will get an error message:

```
error: 'PRJ/PKG' is already an initialized osc working copy
```

Change to the indicated directory and update the content running the command

```
cd <PRJ>/<PKG>; osc update
```

or short

```
cd <PRJ>/<PKG>; osc up
```

This will check out any changed files and tell you which revision is the current one. Everything is now set and you may proceed to *Section 5, "Working on the Sources"*.

4.2 Development Repository - Non-Maintainer or Non-Bugowner

There are at least two situations where you **must not** check out directly from the *devel repo*:

1. If you need to patch an already released package, the *devel repo* usually holds a more current version than the released package. You must not update the release version of such a package (the procedure *devel repo* → *factory* → *product repo* will not work).
2. If you are not the maintainer or bugowner of the package, you cannot submit your changes back to the *devel repo*.

In these cases you need to create a *branch* first. Then you need to check out the sources from this branch.

4.2.1 Branching

The branch you work on will be created on the build server. When you have finished your work and submitted your changes, remove the branch on the build server to free up resources. If you need the branch for additional tasks, you can remove it also at a later point in time.

Create a branch with the command

```
osc branch <PRJ> <PKG>
```

This command creates the subdirectory **home:<username>:branches:<PRJ>/<PKG>** on the server and populates it with the source files (spec file, pristine sources, changes file, current set of patches, etc.) of the package.

If the patch also needs to be applied to maintained versions, this means versions already released as a product, use the command **`osc mbranch <PKG>`** to branch the projects. The **"m"** in **"mbranch"** stands for **"multiple"** , and not as often wrongly assumed for **"maintenance"** .

Note: Add Package Name

Make sure you add the *package* name and **not** the *project* name to the command **`osc branch`**. **`mbranch`** automatically detects in which releases the package is under maintenance and creates the appropriate directory structure. Both commands **`osc branch`** and **`osc mbranch`** provide a list of the files they have created a branch for, and allow you to easily see which releases you need to work on.

4.2.2 Check Out

The commands **`osc branch`** and **`osc mbranch`** only create a branch on the build server. To create a branch on your local system including copies of the packages you want to work on, check them out with the command

```
osc co home:<username>:branches:<PRJ><PKG>
```

Remember you check out a package from the *devel repo* in the same way.

This command creates the directory structure **home:<username>:branches:<PRJ>/<PKG>** in the current directory on your local machine and populates it with all files.

If you created your new subdirectory by mistake within the wrong local directory, you can remove the newly created one with the command **`rm -rf`**. Then move to the correct directory where the new subdirectory should be located, and check out again.

4.2.3 Branch and Check Out

When you update or modify a package, you usually need to perform both steps, branching the repository, and checking out the package. You can easily combine both steps into one command by typing

```
osc branchco <PRJ> <PKG>
```

or by using the short command version

```
osc bco <PRJ> <PKG>
```

If you have used the **`osc mbranch`** command for branching, you cannot combine the two steps **`mbranch`** and **`checkout`** into one command. However, you can specify the `--checkout` option by typing

```
osc mbranch —-checkout <PKG>
```

or in short

```
osc mbranch -c <PKG>
```

Note: Add Package Name

Make sure you add the package name and **not** the project name to the command, as this command will branch and check out all projects where the respective package is under maintenance.

5 Working on the Sources

The sources are now available on your local system and you can start modifying them. Change to the package's working directory. If you checked out from the *devel repo* or from a *distribution repo*, type the command

```
cd <PRJ>/<PKG>
```

If you have branched and checked out of that branch, type the command

```
cd home:<username>:branches:<PRJ>/<PKG>
```

Inside the current working directory, you should find at least the following files:

1. the spec file `<PKG>.spec`,
2. the pristine sources, for example `<PKG>-<vers>.tar.bz2`,
3. the changes file `<PKG>.changes`,
4. any patches,
5. possibly some other files that belong to this package like an `rpmlintrc` file.

Your next steps depend on whether you want to

1. upgrade the package to a new version, which is explained in *Section 5.1, "Upgrading"*, or
2. patch the package to add functionality or fix a bug, which is explained in *Section 5.2, "Patching"*.

5.1 Upgrading

When you upgrade the package to a new version of the sources, you must also check if any of the old patches are obsoleted by the new version and if they still apply without (major) problems. You might hit upon an obsolete or faulty patch when one or more patches have been submitted and accepted upstream and integrated into the sources. Existing patches sometimes do not apply without problems when the relevant source sections have changed considerably, thus confusing the patching program. Fixing issues caused by faulty patches can become an iterative and painful process.

5.1.1 Pristine Sources

As you created a complete set of new sources, the first step should be to replace the old sources by the new ones. Do not forget to remove the old sources to avoid any “hiccup” within the build service. You might however want to save the old sources to a subdirectory of your HOME directory for reference and fallback. In addition, if the sources come with a checksum (for example an MD5SUM) or a signature, you should include it.

5.1.2 Spec File

Modify the spec File to reflect the new release (version, release, source name). First, disable all existing patches, as some of them might be obsolete or have problems being applied (for example if the sources have changed considerably to disrupt the patch program). You can also temporarily move all patches to another directory and bring them back in when required.

5.1.3 Build the Source Tree

Set up the source tree by running the command:

```
quilt setup <PKG>.spec
```

As there are no patches (yet) available, this command mainly unpacks the TAR archive.

Note: Asterisk Wild Card

Often you can use the command **`quilt setup *.spec`** with the asterisk wild card, because usually there is only one spec file. As **`quilt setup`** accepts only a single non-option argument, no damage occurs if more than one spec file exists. However, a warning message is emitted. This means you can redo the command with the desired spec file.

5.1.4 Check the Patches

Now, one by one, inspect the patches that you removed previously. Determine whether they are still needed. As these are ordinary “patch” type patches, you can use the command

```
patch –dry-run -p1 < <wherever>/patchfile
```

to see if the patch applies well. As some patches were built one directory level deeper, you might need to change to that directory and exchange the **`-p1`** with **`-p0`**.

If the patch works well, use the command

```
quilt import <wherever>/patchfile
```

to import the patch and add it to the `series` file.

With the command **`quilt push`** (no further arguments) you actually apply the patch. Add the patch to the spec file, while keeping the original numbering, and continue with *Section 6, "Build the Package"*.

If you prefer, you can re-introduce all the patches you removed earlier and build the package subsequently. You can also add one patch at a time and build the package after each addition.

If a patch is not accepted, you most probably need to re-create it. *Section 5.2, "Patching"* details how to build a new patch. When you have finished adding all the patches, proceed to *Section 6, "Build the Package"*.

5.2 Patching

The previous paragraphs describe how to update a package to a new release or version. But usually, you need to work on an existing package and add a patch to fix a bug or implement a feature.

If an upstream patch is already available you can use that patch (see *Section 5.1.4, "Check the Patches"*). This can be the case if, for example, you need to fix a security vulnerability and someone else has already posted a patch for it.

In other cases, you need to create the patch yourself. This can be the case if you

- cannot apply an existing patch,
- have analyzed a problem and found a fix yourself,
- or implemented a new feature.

Start by checking out the sources as described in section *Section 4, "Getting the Package"*. Then proceed to work on the sources.

5.2.1 Setup

At this stage, you have branched and checked out the *package*. It now resides in its directory, and consists of the `spec` file, the TAR archive, any patches, the `changes` file and some additional files.

The TAR archive contains the *pristine sources*. These are the unmodified sources cloned from a Git repository or downloaded from the project's upstream Web site. Most packages also already come with a set of patches. Extract the source files from the TAR archive and apply these patches first, as specified in the `spec` file, before starting to build your own patch.

`Quilt` offers subcommands that will do this for you.

Initialize the source tree from the TAR archive. Unpack the TAR archive and prepare for the application of the `%prep` section of the `spec` file with the command

```
quilt setup <packagename>.spec
```

You can also try to use the asterisk wild card `*.spec`, but if a package has more than one `spec` file, using the wild card does not work. If this is the case, **`quilt`** alerts you.

While the command **`quilt setup`** prepares the sources, it does not apply the patches. It merely creates a `series` file which contains the list of patches from the `spec` file in the correct order. To apply the patches, change to the source directory and run the command **`quilt push -a`**.

This command applies the patches in the order specified in the `series` file. The output of this command shows you which patches are applied to which files, and if everything went smoothly during the application. You are now prepared to add your own patch.

Note: Symbolic Link

Occasionally, especially if there are no patches to be applied, you need do create a symbolic link from the source directory to the checkout directory by typing **`ln -s .. patches`**.

5.2.2 Building the patch

When you build a new patch, you first need to name it. To do so, in the root of the source tree (this is where you just ran the command **`quilt push -a`**), run the command

```
quilt new <patchname>
```

For a guide on how to name the patch, see the instructions on the openSUSE Web site at https://en.opensuse.org/openSUSE:Packaging_Patches_guidelines#Patch_naming

Note: Name Your Patch

Do not forget to name your new patch, else any changes you make will be added to the last patch!

To keep everything under control, you **must not** use your favorite editor to modify a file (although it is perfectly fine to peek into a file using your editor). From the root of the source tree, run the command:

```
quilt edit dir/file
```

This command does some "bookkeeping"; for example it saves the original version of the file. Then it invokes your editor (from the *EDITOR* environment variable) on the respective file. Subsequently it adds some more information so that the changes later end up in your patch.

You can run the command **`quilt edit`** on several files at once:

```
quilt edit dir1/file1 dir2/file2
```

You can also run **`quilt edit`** multiple times on the same file, reverting and amending previous changes. Only the differences between the original version and the end version are recorded in the patch. The first time you run this command the original file is saved. Each time you leave the editor, the current state of the file is compared to this copy and the patch is created from the differences. If you, for example, make thousands of changes, which, in effect, only change one line, then the patch will contain only the changes to this single line.

When you build a patch, you should follow these rules:

1. Make sure you include all required changes in one patch. Do not combine multiple fixes in one patch.
2. Create multiple patches, if required.

This will make it easier to remove a single patch later, for example if the changes are accepted upstream.

5.2.3 Including the Patch

When you have finished the modifications of the sources, run the command **`quilt refresh`** to make sure that the patch is stored in the same directory as all other patches.

Then change to the checkout directory by typing **`cd ..`** and prepare for the inclusion of the patch.

5.2.3.1 Build Service

Now make your patch known to the build service with the command **`osc ar`**.

You can also use the command `osc add <patchname>` but the former one helps you to remove files no longer needed.

5.2.3.2 Spec File

Next, add the patch to the spec file. Add at least two lines:

```
Patch<n>: <patchname>
```

and

```
%patch<n> -p1
```

The first line belongs to the header section, usually right after any **Source<n>:** lines. Above the **Patch< n>:** line you should also add a comment line describing what the patch tries to accomplish. Make sure to align the patch name with any other columns above, for example the **Source<n>:** lines.

The second line goes into the **%prep** section, again appended to the end of any list of existing **%patch<n>** lines. Don't forget that the command `quilt edit` creates a patch with a patch level of **-p1**.

Note: Adding Lines to Spec File

Whenever you add these two lines, keep to the given style, aligning columns and appending lines to already existing sets of lines of the same type. Also, do not fill holes in the patch numbers: your patches number is the number of the last patch **plus one**. If your patch is the first one, then it is named **Patch1**.

5.2.3.3 Changes

Another important file is the `changes` file. The naming convention usually is `<package>.changes`. This file is maintained through the command `osc vc`.

This command opens an editor on the given `changes` file. As there is not much space to fill in your changes, try to limit the number of items you want to include here. This is especially difficult if you update a package where a lot of changes were made upstream. In this case, mention only the most important changes and refer to the respective change log on the Web.

In the change file, you **must** mention the following information:

- A BUGZILLA reference if a bug was fixed with this patch. To help others find the bug, prepend the number with a shorthand for bugzilla: **bsc (suse.com)**, **bnc (novell.com)**, **boo (opensuse.org)** followed by a hash sign (for example **bsc#1234567**)
- A FATE reference if you add a feature. Like the BUGZILLA references, this consists of the word "fate" followed by a hash, followed by the FATE number (for example **fate#123456**).
- The patch name
- The Common Vulnerabilities and Exposures (CVE) ID if the patch closes a security issue (for example **CVE-1234-5678**).

For more information about how to create a `changes` file, refer to the openSUSE Wep page at https://en.opensuse.org/openSUSE:Creating_a_changes_file_(RPM).

If you have more than one BUGZILLA or FATE reference, because the bug was reported or the feature was requested for more than one release, you must add all references. To avoid the rejection of your patch, make sure that you include the requested information in the `changes` file.

5.2.4 Commit

Important: Embargoed Bugs

Occasionally you will be asked to fix a bug that is EMBARGOED. These bugs usually concern security violations (CVEs). This means no information has been released yet about this bug. Never commit anything to OBS while bugs are still embargoed.

If your local machine runs one of the architectures the package will need to be built for, you can build it locally with the command **`osc build`**.

This command creates a build environment on your local machine, installs tools and packages required to build the package and actually builds it. You can use this command also to keep all the information local. After the package is built, you can immediately get it and test it locally.

At this stage, you have:

1. added your patch(es) to the `spec` file (<**some editor**> `*.spec`)
2. added an entry to the `changes` file (with the command **`osc vc`**)
3. made your patches known to **`osc`** (with the command **`osc ar`**)

To send the changed files to the build server and build the package, run the command **`osc commit`** without further arguments. This command:

1. starts an editor to allow you to add a message; even if you have already added information to the `changes`, an additional commit message is helpful
2. synchronizes the files on the build server with the local files by:
    a. adding or removing files according to the command **`osc ar`** and
    b. transferring modified files
3. triggers one or more builds for various architectures and/or releases

6 Build the Package

When you use the command **`osc commit`** to submit your changes, the build service starts to build the package for various projects and architectures. Depending upon the load, this may take a while. You can check the progress by running the command **`osc r`**.

The output shows three columns containing:

1. the project (for example **openSUSE_Leap_42.2** or **openSUSE_Factory**)
2. the architecture (for example **x86_64** or **ppc64**)
3. the status (for example **scheduled**, **building**, **succeeded**, or **failed**)

If the status includes an asterisk (for example **succeeded***), this means the status is outdated. You should always double check if a status is outdated. If you are in doubt, use the command **`osc buildlog`** or **`osc bl`** to check the status. If the status is **building**, **succeeded**, or **failed**, look at the build log using the **`osc bl <PRJ> <ARCH>`** subcommand for further information (<PRJ> is the project and <ARCH> is the machine architecture).

In the early stages of the build process, the `osc r` command does not return any output. In such cases repeat the command. If `osc r` consistently includes an asterisk, run the `osc bl` command to verify if the status is really outdated.

Note: Dates and Times

Dates and times in the build log output are displayed in Coordinated Universal Time (UTC).

The build log is also very useful if a build has failed: the output contains information about what caused the failure. Use this information to fix the cause and re-commit.

7 Test

When the package has been built (you can find the list of packages built at the end of the build log output), you should test your changes. Depending on the kind of package you have been working on, you can

- use a physical machine from the Orthos pool or
- set up a virtual machine

You obtain the packages using the command

```
osc getbinaries <PRJ> <ARCH>
```

If you also want to get the source RPM, add the option `--sources`.

The files are saved in the directory **binaries** in the current directory. Use the option `-d DIR/--destdir=DIR` to specify a different destination. The command displays a list of fetched files.

8 Submit

After you have tested the package successfully, you usually want to release it. This task is called **submit**.

If you haven't done so already, add a comment to the changelog with the command `osc vc`

To submit the package, type **`osc sr`**. This command requires no further arguments as the changes are submitted to where the package was checked out from beforehand. Also, if you submit to your branch, a request is sent to the maintainer or to the bugowner of the package to include the changes into the repository where you branched from. If you are the maintainer or bugowner and had checked out from the *devel repo*, this command updates the *devel repo*. If you branched via the command **`osc mbranch`** and you submit now to one of the branches, the command **`osc sr`** will detect this and will "request to open a NEW maintenance incident instead". If you submitted from an *mbranch* or to your *personal* branch, your work is done: others will follow up and finish the work.

If you submitted to the *devel repo*, to fulfill the **Factory first** policy, you should then submit the changes to *Factory*/> with the command

```
osc sr <PRJ>/<PKG> openSUSE:Factory
```

This will not immediately submit the package to *Factory*, but it will be marked for submission. When the package is accepted in the *devel repo*, it will automatically be submitted to *Factory*, where it again needs to be accepted.

All requests can be inspected at

- the *Open Build Service* at https://build.opensuse.org/request/show/<RequestID (https://build.opensuse.org/request/show/<RequestID>) for normal submit requests, or at
- the *Internal Build Service* for maintenance requests.

After a certain period of time your new package will be accepted.

9 Propagate a patch

Sometimes you need to patch a package in more than one repository.

A useful technique to do so is to start at one end of the sequence of releases. This could be the oldest or the newest release, for example the version in the *devel repo*. Now work your way down to the other end, to the newest or the oldest release. At the start end, you build the patch and test it. Then you change to the next (newer or older) package instance and use the patch from the previous step as the basis (use **`patch –-dry-run`**, **`quilt import`**, **`quilt push`**). In theory this approach assumes that, at each step, the package will not have changed much so

the patch from the previous step can be applied, albeit with some correction. Sometimes this does not work; then you need to re-write the patch from scratch. But usually this method works, saves some time and prevents errors.

10 Conclusion

In summary, the tasks you usually perform to modify a package and submit a patch are the following:

1. **`osc bco <PRJ> <PKG>`**
2. **`cd home:<username>:branches:…`**
3. **`quilt setup *.spec`**
4. **`cd <sources>`**
5. **`quilt push -a`**
6. write your patch/modify the package
7. **`quilt refresh`**
8. **`cd ..`**
9. **`osc ar`** (*Don't forget this step!*)
10. **`EDITOR *.spec`** (*add the patch*)
11. **`osc vc`** (*add description*)
12. **`osc commit`**
13. **`osc r`**
14. **`osc bl <REPOSITORY> <PKG>`**
15. **`osc submit`**

After having followed all the steps above, you have submitted your first patch. If you have learned new things, don't hesitate to share your knowledge with others. If you stumbled across a problem not covered in this paper, or if you found another solution that saves some work, don't hesitate to send your feedback to `doc-team@suse.com` and help enhance this document.

11 Legal Notice

This article is part of a series of documents called "SUSE Best Practices". The individual documents in the series were contributed voluntarily by SUSE's employees and by third parties.

The articles are intended only to be one example of how a particular action could be taken. They should not be understood to be the only action and certainly not to be the action recommended by SUSE. Also, SUSE cannot verify either that the actions described in the articles do what they claim to do or that they don't have unintended consequences.

Therefore, we need to specifically state that neither SUSE LLC, its affiliates, the authors, nor the translators may be held liable for possible errors or the consequences thereof. Below we draw your attention to the license under which the articles are published.

GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
 Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts". line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.