How To Survive With Many Patches
or
Introduction to Quilt*

Andreas Grünbacher, SuSE Labs
agruen@suse.de

February 22, 2012

Abstract

After looking at different strategies for dealing with software packages that consist of a base software package on top of which a number of patches are applied, this document introduces the script collection *quilt,* which was specifically written to help deal with multiple patches and common patch management tasks.

1 Introduction

In the old days, vendor specific software packages in the open source world consisted of a file with the official version of the software, plus a patch file with the additional changes needed to adapt the package to specific needs. The official software package was usually contained in a package.tar.gz file, while the patch was found in package.diff. Instead of modifying the official package sources, local changes were kept separate. When building the software package, the tar archive was extracted, and the patch was applied.

Over time, the patch file ended up containing several independent changes. Of those changes, some were integrated into later versions of the software, while other add-ons or adaptations remain external. Whenever a new official version was integrated, the patch needed to be revised: changes that were already integrated in the official version needed to be split from changes that were not.

A big improvement was to allow multiple patches in a vendor package, and this is also how patches are handled today: a number of patches is applied on top of each other. Each patch usually consists of a logically related set of changes. When some patches get integrated upstream, those patches can simply be removed from the vendor specific package. The remaining patches

*Quilt is a GPL licensed project hosted on GNU Savannah. Some ideas for this document were taken from *docco.txt* in Andrew Morton's patch management scripts package [1]. The text in the examples was taken from *A Midsummer Night's Dream* by William Shakespeare.

frequently continue to apply cleanly. Some of the remaining patches may have to be maintained across a range of upstream versions because they are too specific for the upstream software package, etc. These patches often get out of sync, and need to be updated.

For the majority of packages, the number of patches remains relatively low, so maintaining those patches without tools is feasible. A number of packages have dozens of patches, however. At the extreme end is the kernel source package (kernel-source-*2.4.x*) with more than 1 000 patches. The difficulty of managing such a vast number of patches without tools can easily be imagined.

This document discusses different strategies of dealing with large sets of patches. Patches are usually generated by the *diff* utility, and applied with the *patch* utility. Different patch file formats are defined as part of the specification of the *diff* utility in POSIX.1 [3]. The most commonly used format today, *unified diff,* is not covered by POSIX.1, however. A good description of patch file formats is found in the *GNU diff* info pages [4].

The question we try to answer in this document is how patches are best kept up to date in face of changes both to the upstream software package, and to the patches that precede them. After looking at some existing approaches, a collection of patch management scripts known as *quilt* is described [2], starting with basic concepts, and progressing towards more advanced tasks.

2 Existing Approaches

The minimal solution for updating a patch is to apply all preceding patches. Then, a copy of the resulting source tree is created.[1] The next patch in the sequence of patches (which is the one to be updated) is applied to only one of these source trees. This source tree is then modified until it reflects the desired result. The new version of the patch is distilled by comparing the two source trees with *diff*, and writing the result into a file.

This simple approach is rather error prone, and leaves much to be desired. Several people have independently written scripts that automate and improve upon this process.

A version control system like *CVS* or *RCS* may be a reasonable alternative in some cases. The version control system is brought in the state of the working tree with a number of patches applied. Then the next patch is applied. After the working tree is updated as required, a diff between the repository copy and the working tree is created (with *cvs diff*, etc). In this scenario the version control system is used to store and compare against the old repository version only. The full version control overhead is paid, while only a small fraction of its functionality is needed. Switching between different patches is not simplified.

[1] The two copies can also be hard-linked with each other, which significantly speeds up both the copying and the final "diffing". If hard links are used, care must be taken that the tools used to update one copy of the source tree will create new files, and will not overwrite shared files. Editors like *emacs* and *vi*, and utilities like *patch*, support this.

One of the most advanced approaches is Andrew Morton's patch management scripts [1]. The author of this document found that none of the available solutions would scale up to the specific requirements of the SUSE kernel-source package, and started to improve Andrew Morton's scripts until they became what they are now [2].

3 Quilt: Basic Concepts and Operation

The remainder of this document discusses the script collection *quilt*.

With quilt, all work occurs within a single directory tree. Since version 0.30, commands can be invoked from anywhere within the source tree (the directory tree is scanned upwards until either the .pc or the patches directory is found). Commands are of the form "quilt cmd", similar to CVS commands. They can be abbreviated as long as the specified part of the command is unique. All commands print some help text with "quilt cmd -h".

Quilt manages a stack of patches. Patches are applied incrementally on top of the base tree plus all preceding patches. They can be pushed on top of the stack (quilt push), and popped off the stack (quilt pop). Commands are available for querying the contents of the series file (quilt series, see below), the contents of the stack (quilt applied, quilt previous, quilt top), and the patches that are not applied at a particular moment (quilt next, quilt unapplied). By default, most commands apply to the topmost patch on the stack.

When files in the working directory are changed, those changes become part of the working state of the topmost patch, provided that those files are part of the patch. Files that are not part of a patch must be added before modifying them so that quilt is aware of the original versions of the files. The quilt refresh command regenerates a patch. After the refresh, the patch and the working state are the same.

Patch files are located in the patches sub-directory of the source tree (see Figure 1). The QUILT_PATCHES environment variable can be used to override this location and quilt will remember this location by storing its value in the .pc/.quilt_patches file. The patches directory may contain sub-directories, which is useful for grouping related patches together. patches may also be a symbolic link instead of a directory.

A file called series contains a list of patch file names that defines the order in which patches are applied. Unless there are means by which series files can be generated automatically (see Section 5.8), they are usually provided along with a set of patches. In series, each patch file name is on a separate line. Patch files are identified by pathnames that are relative to the patches directory; patches may be in sub-directories below the patches directory. Lines in the series file that start with a hash character (#) are ignored. When quilt adds, removes, or renames patches, it automatically updates the series file. Users of quilt can modify series files while some patches are applied, as long as the applied patches remain in their original order.

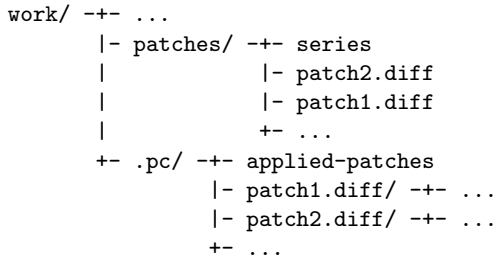

Figure 1: Quilt files in a source tree.

Different series files can be used to assemble patches in different ways, corresponding for example to different development branches.

Before a patch is applied (or "pushed on the stack"), copies of all files the patch modifies are saved to the .pc/*patch* directory.[2] The patch is added to the list of currently applied patches (.pc/applied-patches). Later when a patch is regenerated (quilt refresh), the backup copies in .pc/*patch* are compared with the current versions of the files in the source tree using *GNU diff*.

Documentation related to a patch can be put at the beginning of a patch file. Quilt is careful to preserve all text that precedes the actual patch when doing a refresh.

The series file is looked up in the root of the source tree, in the patches directory, and in the .pc directory. The first series file that is found is used. This may also be a symbolic link, or a file with multiple hard links. Usually, only one series file is used for a set of patches, so the patches sub-directory is a convenient location.

While patches are applied to the source tree, the .pc directory is essential for many operations, including taking patches off the stack (quilt pop), and refreshing patches (quilt refresh). Files in the .pc directory are automatically removed when they are no longer needed, so usually there is no need to clean up manually. The QUILT_PC environment variable can be used to override the location of the .pc directory.

4 An Example

This section demonstrates how new patches are created and updated, and how conflicts are resolved. Let's start with a short text file:

```
Yet mark'd I where the bolt of Cupid fell:
It fell upon a little western flower,
Before milk-white, now purple with love's wound,
And girls call it love-in-idleness.
```

[2]The patch name with extensions stripped is used as the name of the sub-directory below the .pc directory. *GNU patch*, which quilt uses internally to apply patches, creates backup files and applies the patch in one step.

New patches are created with quilt new. A new patch automatically becomes the topmost patch on the stack. Files must be added with quilt add before they are modified. Note that this is slightly different from the CVS style of interaction: with CVS, files are in the repository, and adding them before committing (but after modifying them) is enough. Files are usually added and immediately modified. The command quilt edit adds a file and loads it into the default editor. (The environment variable EDITOR specifies which is the default editor. If EDITOR is not set, *vi* is used.)

```
$ quilt new flower.diff
Patch flower.diff is now on top
$ quilt edit Oberon.txt
File Oberon.txt added to patch flower.diff
```

Let's assume that the following lines were added to Oberon.txt during editing:

```
The juice of it on sleeping eye-lids laid
Will make a man or woman madly dote
Upon the next live creature that it sees.
```

The actual patch file is created (and later updated) with quilt refresh. The result is as follows:[3]

```
$ quilt refresh
$ cat patches/flower.diff
Index: example1/Oberon.txt
===================================================================
--- example1.orig/Oberon.txt
+++ example1/Oberon.txt
@@ -2,3 +2,6 @@
 It fell upon a little western flower,
 Before milk-white, now purple with love's wound,
 And girls call it love-in-idleness.
+The juice of it on sleeping eye-lids laid
+Will make a man or woman madly dote
+Upon the next live creature that it sees.
```

Now let's assume that a line in the text has been overlooked, and must be inserted. The file Oberon.txt is already part of the patch flower.diff, so it can immediately be modified in an editor. Alternatively, quilt edit can be used; it simply opens up the default editor if the file is already part of the patch.

After the line is added, we use quilt diff -z to see a diff of the changes we made:

```
$ quilt diff -z
Index: example1/Oberon.txt
===================================================================
--- example1.orig/Oberon.txt
+++ example1/Oberon.txt
@@ -2,6 +2,7 @@
```

[3]Timestamps in patches are omitted from the output in the examples.

```
 It fell upon a little western flower,
 Before milk-white, now purple with love's wound,
 And girls call it love-in-idleness.
+Fetch me that flower; the herb I shew'd thee once:
 The juice of it on sleeping eye-lids laid
 Will make a man or woman madly dote
 Upon the next live creature that it sees.
```

A diff of the topmost patch can be generated with quilt diff without arguments. This does not modify the actual patch file. The changes are only added to the patch file by updating it with quilt refresh. Then we remove the patch from the stack with quilt pop:

```
$ quilt refresh
Refreshed patch flower.diff
$ quilt pop
Removing flower.diff
Restoring Oberon.txt

No patches applied
```

Next, let's assume that Oberon.txt was modified "upstream": The word *girl* did not fit very well, and so it was replaced with *maiden.* Oberon.txt now contains:

```
Yet mark'd I where the bolt of Cupid fell:
It fell upon a little western flower,
Before milk-white, now purple with love's wound,
And maidens call it love-in-idleness.
```

This causes flower.diff to no longer apply cleanly. When we try to push flower.diff on the stack with quilt push, we get the following result:

```
$ quilt push
Applying flower.diff
patching file Oberon.txt
Hunk #1 FAILED at 2.
1 out of 1 hunk FAILED -- rejects in file Oberon.txt
Patch flower.diff does not apply (enforce with -f)
```

Quilt does not automatically apply patches that have rejects. Patches that do not apply cleanly can be "force-applied" with quilt push -f, which leaves reject files in the source tree for each file that has conflicts. Those conflicts must be resolved manually, after which the patch can be updated (quilt refresh). Quilt remembers when a patch has been force-applied. It refuses to push further patches on top of such patches, and it does not remove them from the stack. A force-applied patch may be "force-removed" from the stack with quilt pop -f, however. Here is what happens when force-applying flower.diff:

```
$ quilt push -f
Applying flower.diff
patching file Oberon.txt
```

```
Hunk #1 FAILED at 2.
1 out of 1 hunk FAILED -- saving rejects to file Oberon.txt.rej
Applied flower.diff (forced; needs refresh)
```

After re-adding the lines of verse from flower.diff to Oberon.txt, we update the patch with quilt refresh.

```
$ quilt edit Oberon.txt
$ quilt refresh
Refreshed patch flower.diff
```

Our final version of Oberon.txt contains:

```
Yet mark'd I where the bolt of Cupid fell:
It fell upon a little western flower,
Before milk-white, now purple with love's wound,
And maidens call it love-in-idleness.
Fetch me that flower; the herb I shew'd thee once:
The juice of it on sleeping eye-lids laid
Will make a man or woman madly dote
Upon the next live creature that it sees.
```

5 Further Commands and Concepts

This section introduces a few more basic commands, and then describes additional concepts that may not be immediately obvious. We do not describe all of the features of quilt here since many quilt commands are quite intuitive; furthermore, help text that describes the available options for each command is available via quilt *cmd* -h.

The quilt top command shows the name of the topmost patch. The quilt files command shows which files a patch contains. The quilt patches command shows which patches modify a specified file. With our previous example, we get the following results:

```
$ quilt top
flower.diff
$ quilt files
Oberon.txt
$ quilt patches Oberon.txt
flower.diff
```

The quilt push and quilt pop commands optionally take a number or a patch name as argument. If a number is given, the specified number of patches is added (quilt push) or removed (quilt pop). If a patch name is given, patches are added (quilt push) or removed (quilt pop) until the specified patch is on top of the stack. With the -a option, all patches in the series file are added (quilt push), or all applied patches are removed from the stack (quilt pop).

5.1 Patch Strip Levels

Quilt assumes that patches are applied with a strip level of one (the -p1 option of *GNU patch*) by default: the topmost directory level of file names in patches is stripped off. Quilt remembers the strip level of each patch in the series file. When generating a diff (quilt diff) or updating a patch (quilt refresh), a different strip level can be specified, and the series file will be updated accordingly. Quilt can apply patches with an arbitrary strip level, and produces patches with a strip level of zero or one. With a strip level of one, the name of the directory that contains the working tree is used as the additional path component. (So in our example, Oberon.txt is contained in directory example1.)

5.2 Importing Patches

The quilt import command automates the importing of patches into the quilt system. The command copies a patch into the patches directory and adds it to the series file. For patch strip levels other than one, the strip level is added after the patch file name. (An entry for a.diff with strip level zero might read "`a.diff -p0`".)

Another common operation is to incorporate a fix or similar that comes as a patch into the topmost patch. This can be done by hand by first adding all the files contained in the additional patch to the topmost patch with quilt add,[4] and then applying the patch to the working tree. The quilt fold command combines these steps.

5.3 Sharing patches with others

For sharing a set of patches with someone else, the series file which contains the list of patches and how they are applied, and the patches themselves are all that's needed. The *.pc* directory only contains quilt's working state, and should not be distributed. Make sure that all the patches are up-to-date, and refresh patches when necessary. The –combine option of quilt diff can be used for generating a single patch out of all the patches in the series file.

5.4 Merging with upstream

The concept of merging your patches with upstream is identical to applying your patches on a more recent version of the software.

Before merging, make sure to pop all your patches using quilt pop -a. Then, update your codebase. Finally, remove obsoleted patches from the series file and quilt push the remaining ones, resolve conflicts and refresh patches as needed.

[4] The *lsdiff* utility, which is part of the *patchutils* package, generates a list of files affected by a patch.

5.5 Forking

There are situations in which updating a patch in-place is not ideal: the same patch may be used in more than one series file. It may also serve as convenient documentation to retain old versions of patches, and create new ones under different names. This can be done by hand by creating a copy of a patch (which must not be applied), and updating the patch name in the series file.

The quilt fork command simplifies this: it creates a copy of the topmost patch in the series, and updates the series file. Unless a patch name is explicitly specified, quilt fork will generate the following sequence of patch names: patch.diff, patch-2.diff, patch-3.diff,...

5.6 Dependencies

When the number of patches in a project grows large, it becomes increasingly difficult to find the right place for adding a new patch in the patch series. At a certain point, patches will get inserted at the end of the patch series, because finding the right place has become too complicated. In the long run, a mess accumulates.

To help avoid this by keeping the big picture, the quilt graph command generates *dot* graphs showing the dependencies between patches.[5] The output of this command can be visualized using the tools from AT&T Research's Graph Visualization Project (GraphViz, `http://www.graphviz.org/`). The quilt graph command supports different kinds of graphs.

5.7 Advanced Diffing

Quilt allows us to diff and refresh patches that are applied, but are not on top of the stack (quilt diff -P *patch* and quilt refresh *patch*). This is useful in several cases, for example, when patches applied higher on the stack modify some of the files that this patch modifies. We can picture this as a shadow which the patches higher on the stack throw on the files they modify. When refreshing a patch, changes to files that are not shadowed (and thus were last modified by the patch that is being refreshed) are taken into account. The modifications that the patch contains for shadowed files will not be updated.

The quilt diff command allows us to merge multiple patches into one by optionally specifying the range of patches to include (see quilt diff -h). The combined patch will only modify each file contained in these patches once. The result of applying the combined patch is the same as applying all the patches in the specified range in sequence.

Sometimes it is convenient to use a tool other than *GNU diff* for comparing files (for example, a graphical diff replacement like *tkdiff*). Quilt will not use

[5] The quilt graph command computes dependencies based on which patches modify which files, and optionally will also check for overlapping changes in the files. While the former approach will often result in false positives, the latter approach may result in false negatives (that is, quilt graph may overlook dependencies).

tools other than *GNU diff* when updating patches (quilt refresh), but quilt diff can be passed the --diff=*utility* argument. With this argument, the specified utility is invoked for each file that is being modified with the original file and new file as arguments. For new files, the first argument will be /dev/null. For removed files, the second argument will be /dev/null.

When quilt diff is passed a list of file names, the diff will be limited to those files. With the -R parameter, the original and new files are swapped, which results in a reverse diff.

Sometimes it is useful to create a diff between an arbitrary state of the working tree and the current version. This can be used to create a diff between different versions of a patch (see Section 5.5), etc. To this end, quilt allows us to take a snapshot of the working directory (quilt snapshot). Later, a diff against this state of the working tree can be created with quilt diff --snapshot.

Currently, only a single snapshot is supported. It is stored in the .pc/.snap directory. To recover the disk space the snapshot occupies, it can be removed with quilt snapshot -d, or by removing the .pc/.snap directory manually.

5.8 Working with RPM Packages

Several Linux distributions are based on the RPM Package Manager [5]. RPM packages consist of a spec that defines how packages are built, and a number of additional files like tar archives, patches, etc. Most RPM packages contain an official software package plus a number of patches. Before these patches can be manipulated with quilt, a series file must be created that lists the patches along with their strip levels.

The quilt setup command automates this for most RPM packages. When given a spec file as its argument, it performs the %prep section of the spec file, which is supposed to extract the official software package, and apply the patches. In this run, quilt remembers the tar archives and the patches that are applied, and creates a series file. Based on that series file, quilt setup extracts the archives, and copies the patches into the patches sub-directory. Some meta-information like the archive names are stored as comments in the series file. quilt setup also accepts a series file as argument (which must contain some meta-information), and sets up the working tree from the series file in this case.

6 Customizing Quilt

Upon startup, quilt evaluates the file .quiltrc in the user's home directory, or the file specified with the –quiltrc option. This file is a regular bash script. Default options can be passed to any command by defining a QUILT_*COMMAND*_ARGS variable (for example, QUILT_DIFF_ARGS="–color=auto" causes the output of quilt diff to be syntax colored when writing to a terminal).

In addition to that, quilt recognizes the following variables:

QUILT_DIFF_OPTS Additional options that quilt shall pass to *GNU diff* when

generating patches. A useful setting for C source code is "-p", which causes *GNU diff* to show in the resulting patch which function a change is in.

QUILT_PATCH_OPTS Additional options that quilt shall pass to *GNU patch* when applying patches. (For example, recent versions of *GNU patch* support the "–reject-format=unified" option for generating reject files in unified diff style (older versions used "–unified-reject-files" for that).

QUILT_PATCHES The location of patch files (see Section 3). This setting defaults to "patches".

7 Pitfalls and Known Problems

As mentioned earlier, files must be added to patches before they can be modified. If this step is overlooked, one of the following problems will occur: If the file is included in a patch further below on the stack, the changes will appear in that patch when it is refreshed, and for that patch the quilt pop command will fail before it is refreshed. If the file is not included in any applied patch, the original file in the working tree is modified.

Patch files may modify the same file more than once. *GNU patch* has a bug that corrupts backup files in this case. A fix is available, and will be integrated in a later version of *GNU patch*. The fix has already been integrated into the SUSE version of *GNU patch*.

There are some packages that assume that it's a good idea to remove all empty files throughout a working tree, including the .pc directory. The *make clean* target in the linux kernel sources is an example. Quilt uses zero-length files in .pc to mark files added by patches, so such packages may corrupt the .pc directory. A workaround is to create a symbolic link .pc in the working tree that points to a directory outside.

It may happen that the files in the patches directory gets out of sync with the working tree (e.g., they may accidentally get updated by CVS or similar). Files in the .pc directory may also become inconsistent, particularly if files are not added before modifying them (quilt add / quilt edit). If this happens, it may be possible to repair the source tree, but often the best solution is to start over with a scratch working directory and the patches sub-directory. There is no need to keep any files from the .pc directory in this case.

8 Summary

We have shown how the script collection *quilt* solves various problems that occur when dealing with patches to software packages. Quilt is an obvious improvement over directly using the underlying tools (*GNU patch*, *GNU diff*, etc.), and offers many features not available with competing solutions. Join the club!

The quilt project homepage is `http://savannah.nongnu.org/projects/quilt/`. There is a development mailing list at `http://mail.nongnu.org/mailman/listinfo/quilt-dev`. Additional features that fit into quilt's mode of operation are always welcome, of course.

References

[1] Andrew Morton: Patch Management Scripts, `http://lwn.net/Articles/13518/` and `http://userweb.kernel.org/~akpm/stuff/patch-scripts.tar.gz`.

[2] Andreas Grünbacher et al.: Patchwork Quilt, `http://savannah.nongnu.org/projects/quilt`.

[3] IEEE Std. 1003.1-2001: Standard for Information Technology, Portable Operating System Interface (POSIX), Shell and Utilities, diff command, pp. 317. Online version available from the The Austin Common Standards Revision Group, `http://www.opengroup.org/austin/`.

[4] *GNU diff* info pages (info Diff), section *Output Formats.*

[5] Edward C. Bailey: Maximum RPM: Taking the Red Hat Package Manager to the Limit, `http://www.rpm.org/max-rpm/`.