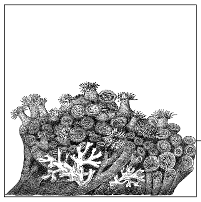

A
Helpful Utilities

Retrieving, building, updating, and maintaining a Linux kernel source tree involves a lot of different steps, as this book shows. Being naturally lazy creatures, developers have created some programs to help with the various routine tasks. Here we describe a few of these useful tools and the basics on how to use them.

Linux kernel development differs in many ways from traditional software development. Some of the special demands on kernel programmers include:

- Constantly applying your changes to the moving target of a fast-based kernel development release schedule
- Resolving any merge conflicts between changes you have made and changes made by other people
- Exporting your changes in a format that lets others incorporate and work with it easily

patch and diff

This section is based on an article originally published in *Linux Journal*.

One of the most common methods of doing kernel work is to use the *patch* and *diff* programs. To use these tools, two different directory trees: a "clean" one and a "working" one must be used. The clean tree is a released kernel version, while the working one is based on the same version but contains your modifications. Then you can use *patch* and *diff* to extract your changes and port them forward to a new kernel release.

For an example, create two directories containing the latest kernel version as described in Chapter 3:

```
$ tar -zxf linux-2.6.19.tar.gz
$ mv linux-2.6.19 linux-2.6.19-dirty
$ tar -zxf linux-2.6.19.tar.gz
$ ls
linux-2.6.19/
linux-2.6.19-dirty/
```

Now make all of the different changes you wish to do in the *-dirty* directory and leave the clean, original kernel directory alone. After finishing making changes, you should create a patch to send it to other people:

```
$ diff -Naur -X linux-2.6.19/Documentation/dontdiff linux-2.6.19/ \
linux-2.6.19-dirty/ > my_patch
```

This will create a file called *my_patch* that contains the difference between your work and a clean 2.6.19 kernel tree. This patch then can be sent to other people via email.

New Kernel Versions

If a new kernel version is released, and you wish to port your changes to the new version, you need to try to apply your generated patch onto a clean kernel version. This can be done in the following steps:

1. Generate your original patch, as in the previous example.
2. Using the official patch from *kernel.org*, move the old kernel version forward one release:

   ```
   $ cd linux-2.6.19
   $ patch -p1 < ../patch-2.6.20
   $ cd ..
   $ mv linux-2.6.19 linux-2.6.20
   ```

3. Move your working directory forward one release by removing your patch, then apply the new update:

   ```
   $ cd linux-2.6.19-dirty
   $ patch -p1 -R < ../my_patch
   $ patch -p1 < ../patch-2.6.20
   $ cd ..
   $ mv linux-2.4.19-dirty linux-2.6.20-dirty
   ```

4. Try to apply your patch on top of the new update:

   ```
   $ cd linux-2.6.20-dirty
   $ patch -p1 < ../my_patch
   ```

   If your patch does not apply cleanly, resolve all of the conflicts that are created (the *patch* command will tell you about these conflicts, leaving behind *.rej* and *.orig* files for you to compare and fix up manually using your favorite editor). This merge process can be the most difficult part if you have made changes to portions of the source tree that have been changed by other people.

If you use this development process, I highly recommend getting the excellent *patchutils* set of programs (found at *http://cyberelk.net/tim/patchutils*). These programs enable you to manipulate text patches easily in all sorts of useful ways, and have saved kernel developers many hours of tedious work.

Managing Your Patches with quilt

Kernel development using *patch* and *diff* generally works quite well. But after a while, most people grow tired of it and look for a different way to work that does not involve so much tedious patching and merging. Luckily, a few kernel developers came up with a program called *quilt* that handles the process of manipulating a number of patches made against an external source tree much easier.

The idea for *quilt* came from a set of scripts written by Andrew Morton that he used to first maintain the memory management subsystem and then later the entire development kernel tree. His scripts were tied very tightly to his workflow, but the ideas behind them were very powerful. Andreas Gruenbacher took those ideas and created the *quilt* tool.

The basic idea behind *quilt* is that you work with a pristine source tree and add a bunch of patches on top of it. You can push and pop different patches off of the source tree, and maintain this list of patches in a simple manner.

1. To get started, create a kernel source tree like always:

   ```
   $ tar -zxf linux-2.6.19.tar.gz
   $ ls
   linux-2.6.19/
   ```

2. And go into that directory:

   ```
   $ cd linux-2.6.19
   ```

3. To get started, create a directory called *patches* that will hold all of our kernel patches:

   ```
   $ mkdir patches
   ```

4. Then tell *quilt* to create a new patch called *patch1*:

   ```
   $ quilt new patch1
   Patch patches/patch1 is now on top
   ```

5. *quilt* needs to be told about all of the different files that will be modifed by this new patch. To do this, use the *add* command:

   ```
   $ quilt add Makefile
   File Makefile added to patch patches/patch1
   ```

6. Edit the file *Makefile*, modify the `EXTRAVERSION` line, and save the change. After you finish, tell *quilt* to refresh the patch:

   ```
   $ quilt refresh
   Refreshed patch patches/patch1
   ```

The file *patches/patch1* will contain a patch with the changes that you have just made:

```
$ cat patches/patch1
Index: linux-2.6.19/Makefile
===================================================================
--- linux-2.6.19.orig/Makefile
+++ linux-2.6.19/Makefile
@@ -1,7 +1,7 @@
 VERSION = 2
```

```
 PATCHLEVEL = 6
 SUBLEVEL = 19
-EXTRAVERSION =
+EXTRAVERSION = -dirty
 NAME=Crazed Snow-Weasel

 # *DOCUMENTATION*
```

You can continue on, working with this single patch, or create a new one to go on top of this patch. As an example, if three different patches had been created, `patch1`, `patch2`, and `patch3`, they will be applied one on top of one another.

To see the list of patches that are currently applied:

```
$ quilt series -v
+ patches/patch1
+ patches/patch2
= patches/patch3
```

This output shows that all three patches are applied, and that the current one is `patch3`.

If a new kernel version is released, and you wish to port your changes to the new version, quilt can handle this easily with the following steps:

1. Pop off all of the patches that are currently on the tree:

    ```
    $ quilt pop -a
    Removing patch patches/patch3
    Restoring drivers/usb/Makefile
    Removing patch patches/patch2
    Restoring drivers/Makefile
    Removing patch patches/patch1
    Restoring Makefile
    No patches applied
    ```

2. Using the official patch from *kernel.org*, move the old kernel version forward one release:

    ```
    $ patch -p1 < ../patch-2.6.20
    $ cd ..
    $ mv linux-2.6.19 linux-2.6.20
    ```

3. Now have quilt push all of the patches back on top of the new tree:

    ```
    $ quilt push
    Applying patch patches/patch1
    patching file Makefile
    Hunk #1 FAILED at 1.
    1 out of 1 hunk FAILED -- rejects in file Makefile
    Patch patches/patch1 does not apply (enforce with -f)
    ```

4. As the first patch doesn't apply cleanly, force the patch to be applied and then fix it up:

    ```
    $ quilt push -f
    Applying patch patches/patch1
    patching file Makefile
    Hunk #1 FAILED at 1.
    1 out of 1 hunk FAILED -- saving rejects to file Makefile.rej
    ```

```
Applied patch patches/patch1 (forced; needs refresh)
$ vim Makefile.rej Makefile
```

5. After the patch is applied by hand, refresh the patch:

```
$ quilt refresh
Refreshed patch patches/patch1
```

6. And continue pushing the other patches:

```
$ quilt push
Applying patch patches/patch2
patching file drivers/Makefile
Now at patch patches/patch2
$ quilt push
Applying patch patches/patch3
patching file drivers/usb/Makefile
Now at patch patches/patch3
```

quilt also has options that will automatically email out all of the patches in the series to a group of people or a mailing list, delete specific patches in the middle of the series, go up or down the series of patches until a specific patch is found, and many more powerful options.

If you want to do any kind of kernel development, *quilt* is strongly recommended, even for tracking a few patches, instead of using the more difficult *diff* and *patch* method. It is much simpler and will save you much time and effort.

On a personal note, I cannot recommend this tool enough, as I use it everyday to manage hundreds of patches in different development trees. It is also used by numerous Linux distributions to maintain their kernel packages and has an involved and responsive development community.

git

git is a source code control tool that was originally written by Linus Torvalds when the Linux kernel was looking for a new source code control system. It is a distributed system, which differs from traditional source code control systems such as CVS in that it is not required to be connected to a server in order to make a commit to the repository.

git is one of the most powerful, flexible, and fast source code control systems currently available, and has an active development team working behind it. The main web page for *git* can be found at *http://git.or.cz/*. It is recommended that any new user of *git* go through the published tutorials in order to become familiar with how *git* works, and how to use it properly.

The Linux kernel is developed using *git*, and the latest *git* kernel tree can be found at *http://www.kernel.org/git/*, along with a large list of other kernel developer's *git* repositories.

It is not necessary to use *git* in order to do Linux kernel development, but it is very handy in helping to track down kernel bugs. If you report a bug to the Linux kernel developers, they might ask you to use *git bisect* in order to find the exact change that caused the bug to happen. If so, follow the directions in the *git* documentation for how to use this.

ketchup

ketchup is a very handy tool used to update or switch between different versions of the Linux kernel source tree. It has the ability to:

- Find the latest version of the kernel, download it, and uncompress it.
- Update a currently installed version of the kernel source tree to any other version, by patching the tree to the proper version.
- Handle the different development and stable branches of the kernel tree, including the *-mm* and *-stable* trees.
- Download any patches or tarballs needed to do the update, if they are not present on the machine already.
- Check the GPG signatures of the tarball and patches to verify that it has downloaded a correct file.

ketchup can be found at *http://www.selenic.com/ketchup/* and has lots of additional documentation in the wiki at *http://www.selenic.com/ketchup/wiki/*.

Here is a set of steps that show how simple it is to use *ketchup* to download a specific kernel version, and then have it switch the directory to another kernel version with only a minimal number of commands.

To have *ketchup* download the 2.6.16.24 version of the kernel source tree into a directory, and rename the directory to be the same as the kernel version, enter:

```
$ mkdir foo
$ cd foo
$ ketchup -r 2.6.16.24
None -> 2.6.16.24
Unpacking linux-2.6.17.tar.bz2
Applying patch-2.6.17.bz2 -R
Applying patch-2.6.16.24.bz2
Current directory renamed to /home/gregkh/linux/linux-2.6.16.24
```

Now, to upgrade this kernel to contain the latest stable kernel version, just enter:

```
$ ketchup -r 2.6
2.6.16.24 -> 2.6.17.11
Applying patch-2.6.16.24.bz2 -R
Applying patch-2.6.17.bz2
Downloading patch-2.6.17.11.bz2
--22:21:14--  http://www.kernel.org/pub/linux/kernel/v2.6/patch-2.6.17.11.
bz2
           => `/home/greg/.ketchup/patch-2.6.17.11.bz2.partial'
Resolving www.kernel.org... 204.152.191.37, 204.152.191.5
Connecting to www.kernel.org|204.152.191.37|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 36,809 (36K) [application/x-bzip2]
100%[====================================>] 36,809        93.32K/s
22:21:14 (92.87 KB/s) - `/home/greg/.ketchup/patch-2.6.17.11.bz2.partial'
saved [36809/36809]
Downloading patch-2.6.17.11.bz2.sign
```

```
--22:21:14--  http://www.kernel.org/pub/linux/kernel/v2.6/patch-2.6.17.11.
bz2.sign
           => `/home/greg/.ketchup/patch-2.6.17.11.bz2.sign.partial'
Resolving www.kernel.org... 204.152.191.37, 204.152.191.5
Connecting to www.kernel.org|204.152.191.37|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 248 [application/pgp-signature]
100%[==================================>] 248           --.--K/s
22:21:14 (21.50 MB/s) - `/home/greg/.ketchup/patch-2.6.17.11.bz2.sign.
partial' saved [248/248]
Verifying signature...
gpg: Signature made Wed Aug 23 15:01:04 2006 PDT using DSA key ID 517D0F0E
gpg: Good signature from "Linux Kernel Archives Verification Key >
ftpadmin@kernel.org<"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the
owner.
Primary key fingerprint: C75D C40A 11D7 AF88 9981  ED5B C86B A06A 517D 0F0E
Applying patch-2.6.17.11.bz2
Current directory renamed to /home/greg/linux/tmp/x/linux-2.6.17.11
```

This shows that *ketchup* automatically determined that the newest stable version was 2.6.17.11 and downloaded the needed patch files in order to get to that version.

It is highly recommended that you use *ketchup* if you want to download any Linux kernel source trees. It takes all of the work in finding where on the server the correct patch file is, and automatically applies the patch in the proper format, after checking that the downloaded file is properly signed. Combine *ketchup* with *quilt* and you have a very powerful setup that contains everything that you need in order to deal effectively with kernel sources as a Linux kernel developer.