Linux Developer Notes

Jeremiah Mahler
<jmmahler@gmail.com>

June 8, 2015

Contents

This is a collection of notes accumulated while working on Linux. Topics include, among other things, building/configuring the kernel, using Git, and using Mutt.

This information should be useful for any Linux developer but it also applies to other projects whose core development model is based around Git. Git[1] and Subsurface[2] are two examples.

1 Kernel

1.1 Retrieving The Kernel Sources

Use `git clone` to retrieve a particular branch.

linux-stable (Greg Kroah-Hartman):

```
git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

linux-next:

```
git://git.kernel.org/pub/scm/linux/kernel/git/next/linux-next.git
```

linux (Linus Torvalds):

```
git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

1.2 Following Upstream

When a project is cloned it is automatically setup to track the origin. Then a `git pull` will automatically pull updates from that origin. Local changes are kept local an updates can be pulled from upstream.

But suppose you have a mirror of the upstream project on your own server. If you clone from your project then it will not be setup to pull updates from the origin.

To git around this situation the `git remote` can be used along with `git pull -u`. Consider the following example.

First you clone from your server.

```
git clone git@github.com:jmahler/git
```

Then add the upstream remote. To view your remotes try `remote -v`.

```
git remote add upstream git://git.kernel.org/pub/scm/git/git
```

Finally, you need to configure your local branch to track upstream and not origin (your server). Here the `pu` branch is used.

```
git branch -u upstream/pu pu
```

There are several ways to use `branch -u`, this is one of the more explicit variations.

[1] *Git*. [Online; accessed 11-January-2015]. URL: `http://git.kernel.org`.

[2] Dirk Hohndel. *Subsurface*. [Online; accessed 11-January-2015]. URL: `http://git.hohndel.org/?p=subsurface.git`.

1.3 Configuring a Kernel

There are many ways to configure a kernel. When starting from scratch, with no existing configuration, a good starting place is to use the config provided with the Linux distribution.

```
cp /boot/config-`uname -r` .config
```

Often the kernel sources being used are newer than the kernel provided by the distribution. There may be new features that need to be added to the configuration. To update the configuration `oldconfig` can be used. It will prompt for each new feature and ask whether it should be enabled in the configuration.

```
make oldconfig
```

A downside of using the config file from a distribution is that it includes everything needed to support lots of different systems. Most of this is not necessary for one specific system. And because of this extra code it will take a long time to build a kernel.

One way to reduce compile time is to reduce the configuration to just what is needed for a specific system. The `localmodconfig` command[3] can be used to accomplish this.

```
make localmodconfig
```

`localmodconfig` works by looking at the modules which are currently loaded on your system (`lsmod`). This means that hot pluggable devices, such as usb, should have been plugged in once so that the modules were loaded. Note, modules are not unloaded after a device is un-plugged.

`localmodconfig` will only remove config options, it will not add them. This makes a distribution provided config, which has lots of config options for lots of modules, a good starting point.

Invariably a module will be missed from the configuration and have to be added later. While `localmodconfig` will not add the module, it will display a message. Then `menuconfig` can be used to manually add the module and update the configuration.

1.4 Building and Installing a Kernel

Two different ways to build and install a kernel will be discussed here. The first is the manual method which only use the scripts that come with the Linux kernel. The second will build an entire .deb package.

To build using the manual method simply type `make`. The `-j` option can be used to take advantage of multiple processors.

```
make -j2
```

Once this is complete they are installed.

```
sudo make modules_install install
```

To build a .deb the `make-kpkg` command is used.

```
make-kpkg -j2 --rootcmd fakeroot --initrd kernel_image
```

The resulting .deb will be in the parent directory and can be installed using `dpkg`.

```
sudo dpkg -i ../linux-image-3.15.0-rc5_3.15.0-rc5-10.00.Custom_amd64.deb
```

These two different methods each have their benefits. The manual method does less work during a rebuild. `make-kpkg` seems to do everything from a full clean every time. The manual method will install the files but they have to be manually removed. Usually it is easy to find what to remove in `/boot` and `/lib/modules`

[3] *kconfig: localmodconfig for v2.6.32.* [Online; accessed 11-January-2015]. URL: `http://lwn.net/Articles/352241/`.

but it is not as clean as a .deb which is easy to install and remove without leaving any files behind. Also, if the files are manually removed, `update-grub` must be run as well.

The manual method is better suited for developers where rebuild time is important. For distributing a kernel the .deb method is better because of the clean install/remove process.

1.5 Building Modules

The modules can be built independently from the kernel by using the `M=` option.

```
make M=scripts
make M=drivers/staging/comedi/drivers
```

Note, using `M=` does not override the kernel `.config`, if they are not enabled they won't be built.

To add extra flags use the `EXTRA_CFLAGS` variable.

```
make EXTRA_CFLAGS="-DDEBUG" M=drivers/staging/comedi/drivers
```

1.6 Rebooting to the Last Built Kernel

A typical 'git bisect' work flow is as follows:

```
make -j2
make modules_install install
(user takes note of the kernel version)
sudo shutdown -r now
(user selects kernel version to boot from Grub menu)
(bootup ...)
(perform tests)
git bisect good/bad
(build ...)
```

However, expecting the user to take note of the version and reboot correctly takes time and can cause errors. With just a few commands these steps can be automated.

Grub provides the `grub-reboot` command which allows the default kernel used during the next boot to be set [4]. Either numbered offsets or text names can be used.

```
sudo grub-reboot 1>4
sudo grub-reboot 1>Debian GNU/Linux, with Linux 4.1.0-rc6-next-20150604+
```

The Linux kernel Makefile provides the `kernelrelease` target which will print the name of the currently built kernel.

```
make kernelrelease
4.1.0-rc6-next-20150604+
```

And these can be combined in to a shell script to set which kernel will be used during the next reboot as shown in Listing 1.

[4]'grub-reboot' requires that 'GRUB_DEFAULT' is set to saved.

```
#!/bin/sh

KERNELRELEASE=$(make kernelrelease) || exit 1

MENUENTRY='1>'$(grep $KERNELRELEASE /boot/grub/grub.cfg |
                                  grep menuentry |
                                  head -n 1 |
                                  awk -F \' '{print $2}')

sudo grub-reboot "$MENUENTRY" || exit 1
```

Listing 1: "Script to set next kernel to boot."

2 Patches

2.1 Typical Work Flow

```
(make some changes)
git commit -F log.txt --signoff
git format-patch --cover-letter --reroll-count=2 -1
(edit the cover letter)
git send-email --to foo@bar.com --cc foo2@bar.com v2-0*
```

2.2 Log Messages

The width of a log message should be limited so that it fits on a 80 character terminal without wrapping. A good rule of thumb is to use 72 characters. This works for git log messages, which add 4 spaces. And it also works on mailing lists, where patch comments can be several levels deep.

2.3 Tags

```
Signed-off-by:
Reviewed-by:
Tested-by:
Suggested-by:
Reported-by:
Acked-by:
Cc:
Link: http://www.kernel.org
```

2.4 Formatting Patches

Before commits can be sent by email they need to be formatted. The `git format-patch` command is used to do this. It will produce a patch in mbox format which can be sent with `git send-email`.

```
git format-patch -1     # last change
git format-patch -2     # last 2 changes

git format-patch --cover-letter -1      # add a cover letter
git format-patch --reroll-count=2 -1    # version 2

git format-patch --signoff -1           # add a Signed-off-by
* Note: --signoff can also be added during a commit

git format-patch --signoff -1           # add a Signed-off-by
```

2.5 Cover Letter

It is helpful to add a cover letter than gives an introduction to the patch series. It should also describe what has been done in previous versions and possibly links to previous conversations. Also, ordering the changes in descending order helps to show what has most recently changed.

```
Changes in v3:
  - Removed #ifdef in header file.

Changes in v2:
  - Changed strnncmp to use strncmp instead of memcmp.
```

2.6 Check Patch

Linux includes a tool called `checkpatch.pl` for checking patches for any style errors and other sorts of problems. Always run this tool and fix any problems before submitting a patch[5].[6]

```
./scripts/checkpatch.pl 0001-my-first-patch.patch
```

2.7 Retrieving Patches

Often when going through the log history it is necessary to look at a single patch. There are many way to specify revisions (`gitrevisions(5)`) but a short way to request the first patch (-1) relative to a given sha1.

```
git format-patch -1 6630f11b
```

2.8 Emailing Patches

Before emailing a patch it is necessary to determine who to send it to. The mailing list for the project is always good. But it should also be sent to people who have worked on the code.

Linux includes the `get_maintainer.pl` script to find all the relevant people.

```
./scripts/get_maintainer.pl 0001-my-first-patch.patch
```

Alternatively, it can be used directly on a file.

```
./scripts/get_maintainer.pl --file drivers/staging/comedi/drivers/ssv_dnp.c
```

The preferred way to send a patch is using `git send-email`. It will also parse the patch and recommended additional email addresses to send the patch to. Often those with `Signed-off-by`, `Cc`, or other tags.

[5] Greg Kroah-Hartman. *YouTube: Write and Submit your first Linux kernel Patch.* [Online; accessed 11-January-2015]. 2010. URL: `https://www.youtube.com/watch?v=LLBrBBImJt4`.

[6] *linux/Documentation/CodingStyle.* [Online; accessed 26-January-2015]. URL: `https://www.kernel.org/doc/Documentation/CodingStyle`.

```
git send-email --to jmmahler@gmail.com --cc linux-kernel@vger.kernel.org \
        0001-my-first-patch.patch
```

Aliases can be used to avoid having to type the full email addresses every time (Section 3.1).

```
git send-email --to jmm --cc linux 0001-my-first-patch.patch
```

2.9 Reply To Revisions

Often a patch will be submitted. Then others will comment on it. And then revised patches will be submitted. It is important to keep this discussion as linear and coherent as possible. Imagine if they joined the conversation at revision 8, could they easily lookup the previous discussion?

One way to do this is to have each patch revision start a new subject, but provide links in the body to the previous discussions on the mailing list.

```
Version 3 of the patch series to cleanup duplicate name_compare()
functions (previously was 'add strnncmp() function' [1]).

  [1]: http://marc.info/?l=git&m=140299051431479&w=2
```

Another more advanced way is to make the new patch revision a reply to the previous by using the "Message-id". The first step is to find the message id in the email headers.

```
...
Subject: [PATCH 1/2] fixed commit
Date: Sun, 22 Jun 2014 08:01:44 -0700
Message-Id: <1403449304-551-1-git-send-email-jmmahler@gmail.com>
X-Mailer: git-send-email 2.0.0
...
```

Next, the `--in-reply-to` option is used with `send-email`. Be sure to quote the id or else the shell might mis-interpret the meaning.

```
git send-email \
  --in-reply-to="<1403449304-551-1-git-send-email-jmmahler@gmail.com>" \
  --to=jmm 0002-added-.gitignore.patch
```

Now the discussion of an entire patch series can be done in one thread with no breaks.

2.10 Reviewing Patches

Giving clear and concise feedback about a small patch is not too difficult. However, if the patch is very large this can be difficult. The most common way is to point out lots of issues about a single patch. Another way is to have one reply address only one issue.[7]

2.11 Applying a Mailed Patch

A patch received in an email can be applied by using the `git am` command. It accepts the message in mbox format which contains not only the diff but also the log message to be used for git.

```
git am patch.mbox
```

[7] Raghu Vatsavayi. *[net-next,v3] Add support of Cavium Liquidio ethernet adapters.* [Online; accessed 26-January-2015]. 2014. URL: `https://patchwork.ozlabs.org/patch/422749/`.

See Section 3.2 for a description of how to do this in Mutt.

3 Mutt

3.1 Email Aliases

Having to type a full email address in to `git send-email` can get tedious. Luckily, aliases can be setup to make this easier. The configuration is done in Mutt but `send-email` uses the same configuration.

First, configuration variables are added to Git.

```
git config --global sendemail.aliasfiletype mutt
git config --global sendemail.aliasesfile $HOME/.muttrc
```

Then the aliases should be added to the `~/.muttrc`.

```
alias jmm Jeremiah Mahler <jmmahler@gmail.com>
alias git git@vger.kernel.org
```

Now send-email can be used with an alias.

```
git send-email --to jmm --cc git  00*.patch
```

3.2 Applying Patches

On big projects, such as the Linux kernel, patches are communicated among the developers by email. Applying patches from these emails needs to be quick and easy. In Mutt this can be accomplished using a macro which allow a key to be bound to a sequence of commands.

The following commands were added to a users `~/.muttrc`. The first command disables the default key binding for `p`. The second creates a macro to pipe the currently selected message to `git am`. This will apply the patch to the project in the users current working directory.

```
# ~/.muttrc
bind index,pager p noop
macro index,pager p "<pipe-entry>git am<enter>" "git am <email patch>"
```

3.3 Editing Patches

In some cases `git format-patch` and `git send-email` do not do what is desired. For example, attaching a file is difficult with `send-email`. Luckily, Mutt can be used to edit a formatted patch by using the `-H` option.

```
git format-patch -1
mutt -H 0001-mypatch.patch
```

NOTE: Attaching files to emails is generally frowned upon in the Linux Kernel mailing list. But it might be useful in other situations.

3.4 Quick Reading

There are a huge number of mail messages on LKML every day. A common use pattern is to mark a thread as read (Ctrl-R) and then go to the next unread thread (Tab). This sequence can be shortened in just the Tab key by using a macro.

```
# ~/.muttrc
bind index <Tab> noop
macro index <Tab> "<read-thread><next-new-then-unread>"
```

TIP: To figure out which key binds to which command type `?`.

4 Bisecting a Problem

Often a problem will arise with a new kernel which wasn't present in a previous kernel. A good way to find the problem is by using `git bisect`.[8]

```
$ git bisect start
$ git bisect bad                  # (current revision is bad)
$ git bisect good next-20141028   # (tag of good revision)
```

Then `git` will checkout the next version to try. You then build the kernel as usual, test it out, and then mark it as good or bad.

```
$ git bisect good
```

or

```
$ git bisect bad
```

And this process is repeated until no more revisions are left to try and the patch that caused the problem is found.

5 Counting Commits

To count the number of commits (patches) an author has submitted `git shortlog` can be used.

```
$ git shortlog -s --author "Jeremiah Mahler"
30 Jeremiah Mahler
```

To get a listing of all the authors in descending order use `-n`.

```
$ git shortlog -s -n
```

[8] *Debugging with Git.* [Online; accessed 26-January-2015]. URL: `http://git-scm.com/book/en/v2/Git-Tools-Debugging-with-Git`.

References

Debugging with Git. [Online; accessed 26-January-2015]. URL: http://git-scm.com/book/en/v2/Git-Tools-Debugging-with-Git.

Git. [Online; accessed 11-January-2015]. URL: http://git.kernel.org.

Gruber, John. *On Top (top posting).* [Online; accessed 26-January-2015]. URL: http://daringfireball.net/2007/07/on_top.

Hamano, Junio C. *Junio on code churn patches.* [Online; accessed 26-January-2015]. URL: http://thread.gmane.org/gmane.comp.version-control.git/245133/focus=245144.

Hohndel, Dirk. *Subsurface.* [Online; accessed 11-January-2015]. URL: http://git.hohndel.org/?p=subsurface.git.

kconfig: localmodconfig for v2.6.32. [Online; accessed 11-January-2015]. URL: http://lwn.net/Articles/352241/.

Kroah-Hartman, Greg. *YouTube: Write and Submit your first Linux kernel Patch.* [Online; accessed 11-January-2015]. 2010. URL: https://www.youtube.com/watch?v=LLBrBBImJt4.

linux/Documentation/CodingStyle. [Online; accessed 26-January-2015]. URL: https://www.kernel.org/doc/Documentation/CodingStyle.

Torvalds, Linus. *Linus on Trivial Patches.* [Online; accessed 26-January-2015]. 2004. URL: https://lkml.org/lkml/2004/12/20/255.

Tso, Theodore. *Theodore Tso on code churn (wankery).* [Online; accessed 26-January-2015]. URL: https://lkml.org/lkml/2014/6/10/819.

Vatsavayi, Raghu. *[net-next,v3] Add support of Cavium Liquidio ethernet adapters.* [Online; accessed 26-January-2015]. 2014. URL: https://patchwork.ozlabs.org/patch/422749/.