Guide to Generating Patches of the Kernel

by Brian Fraser
Last update: April 10, 2015

This document guides the user through:

1. Generating a patch of your changes using Git.
2. Generating a patch of your changes without Git.
3. Applying a patch to your code with or without Git.

TO DO: Add section on how to read a diff's contents.

Table of Contents

1. Generating a Patch with Git 2
   1.1 Committing Changes Locally 2
   1.2 Actually Generating a Patch in Git 4
   1.3 Applying a Git Patch 7
2. Generating a Patch without Git 8
   2.1 Applying a non-Git Patch 9
3. Further Resources 11

Formatting

1. Commands starting with `$` are Linux console commands on the host OS:
   ```
   $ echo Hello world!
   ```
2. Almost all commands are case sensitive.

It is assumed that the user has downloaded the Linux kernel source code, compiled it, and is able to boot the kernel. See "Custom Kernel Guide" on course website (see Further Resources for link). This guide assumes you have the modified Linux source code in a folder `linux-stable/`.

Revision History

- Updated diff instructions to `make mrproper` on both original and modified kernel folders.
- Added directions on setting up user name and email with Git.

TODO:

- Add explicit steps to generating 2nd branch, switching to it, and then at end removing branch.
- Add info on what to do to check which files are in a diff, how to read the file names in it, and how to correct extra files (example: .bak files from an editor).
- Mention that only kernel code in kernel folder (assignment doc?); user mode code elsewhere.

1. Generating a Patch with Git

This section generates the patch using Git but only works if you used Git to checkout ("clone") the code. If not (you extracted a copy of the code from an archive file), see Section 2.

The process is:

1. Commit your changes to your local Git repository
2. Generate a patch
3. Test your patch

It is advisable that you create a ZIP file of the files you modified during your work so that if anything goes terribly wrong you don't lose any of your work.

1.1 Committing Changes Locally

1. Change to the folder of your modified kernel source code (assumed to be `linux-stable/`)
   ```
   $ cd linux-stable
   ```
2. List the file changes that have been made in this branch:
   ```
   $ git status
   ```

Sample output (while working in a branch named "my-patch-test"):

```
$ git status
On branch my-patch-test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

      modified:   Makefile
      modified:   arch/x86/syscalls/syscall_64.tbl

Untracked files:
  (use "git add <file>..." to include in what will be committed)

      cs300/
      launchqemu
      mount_cs300

no changes added to commit (use "git add" and/or "git commit -a")
```

   - Note that I have a couple extra files here (`launchquem`, and `mount_cs300`) which I don't want to add.

3. Select which files to commit locally using the `git add` command for each changed or new file:
   ```
   $ git add Makefile
   $ git add arch/x86/syscalls/syscall_64.tbl
   $ git add cs300/
   ```
   - Git is configured to ignore certain files, such as those created while building the Linux kernel. Therefore, we can add the `cs300/` folder even if it contains build products.
   - If you added an incorrect file, you can undo-adding using:
     ```
     $ git reset da/file/to/undo-adding.on
     ```
   - If you want to revert a file (discard all your changes and checkout the latest version in the

Git repository):
```
$ git checkout da/file/to/revert.xyz
```

- WARNING: This will destroy any changes in your local file!

4. Check what is now ready to be committed:
```
$ git status
```

Sample output:
```
$ git status
On branch my-patch-test
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

      modified:   Makefile
      modified:   arch/x86/syscalls/syscall_64.tbl
      new file:   cs300/Makefile
      new file:   cs300/cs300_test.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)

      launchqemu
      mount_cs300
```

- Note that my extra files (`launchquem`, and `mount_cs300`) are still "untracked" and won't be committed.

5. Setup your user name and email address for Git:
```
$ git config --global user.email "you@example.com"
$ git config --global user.name "Your Name"
```

6. Commit your changes to the local repository (this does not affect the "origin" repository; just your local computer!):
```
$ git commit -m "Your descriptive message goes here."
```

For example:
```
$ git commit -m "Simple 'hello-world' Sys-Call."
[my-patch-test 06efd01] Simple 'hello-world' Sys-Call.
 4 files changed, 19 insertions(+), 1 deletion(-)
 create mode 100644 cs300/Makefile
 create mode 100644 cs300/cs300_test.c
```

7. View that your commit has gone through:
```
$ git log -1
```

For example:
```
$ git log -1
commit 06efd0148547d8540a13fa17148877ea0af8b8cc
Author: Brian Fraser <my_address@some_domain.com>
Date:   Mon Apr 6 11:32:04 2015 -0700

    Simple 'hello-world' Sys-Call.
```

- You can view any number of recent commits, such as
```
$ git log -3
```

1.2 Actually Generating a Patch in Git

1. First ensure that all changes which you want to submit are checked in using:
   `$ git status`
   - See previous section for more.
2. List all commits which have been checked in on the current branch (vs the `master` branch):
   `$ git log master..`

For example:

```
$ git log master..
commit 06efd0148547d8540a13fa17148877ea0af8b8cc
Author: Brian Fraser <my_address@some_domain.com>
Date:   Mon Apr 6 11:32:04 2015 -0700

    Simple 'hello-world' Sys-Call.
```

   - This assumes that you created the current branch off the master.

3. Generate the patch by specifying how many commits to include:
   - If you have only one commit on this branch (shown in the previous step), use:
     `$ git format-patch -1 --stdout > patch.diff`
     - For example:
       `$ git format-patch -1 --stdout > patch.diff`
   - If you had multiple commits, either use a different number (like -3), or use the first and last commit IDs to define the range of commits (inclusive) to add to the patch.
     `$ git format-patch <1st-commit-ID>..<last-commit-ID> --stdout > patch.diff`
   - *You can also try to generate patches for all commits on a branch. The following commands may be useful but seemed error prone for me on my system:*
     `$ git format-patch master --stdout > patch.diff`
     `$ git format-patch HEAD~~ --stdout > patch.diff`
4. Verify the patch by viewing `patch.diff`:
   - Ensure that each "From:" line is you (vs some other developer). If you have patches from other developers showing up, it likely means that there was some error in the command and you should either try again.
   - Ensure that each of the files you created/modified is listed.
   - Ensure that no files other than those you modified/created are listed.
   - Your entire patch should be only a few Kb in size.
5. Example `patch.diff`:

```
From 06efd0148547d8540a13fa17148877ea0af8b8cc Mon Sep 17 00:00:00 2001
From: Brian Fraser <my_address@some_domain.com>
Date: Mon, 6 Apr 2015 11:32:04 -0700
Subject: [PATCH] Simple 'hello-world' Sys-Call.

---
 Makefile                        |  2 +-
 arch/x86/syscalls/syscall_64.tbl |  2 ++
 cs300/Makefile                  |  1 +
 cs300/cs300_test.c              | 15 +++++++++++++++
 4 files changed, 19 insertions(+), 1 deletion(-)
```

```
 create mode 100644 cs300/Makefile
 create mode 100644 cs300/cs300_test.c

diff --git a/Makefile b/Makefile
index 1100ff3..76b3e4e 100644
--- a/Makefile
+++ b/Makefile
@@ -884,7 +884,7 @@ export mod_sign_cmd



 ifeq ($(KBUILD_EXTMOD),)
-core-y        += kernel/ mm/ fs/ ipc/ security/ crypto/ block/
+core-y        += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ cs300/


 vmlinux-dirs  := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
                      $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
diff --git a/arch/x86/syscalls/syscall_64.tbl b/arch/x86/syscalls/syscall_64.tbl
index 8d656fb..01f9a9a 100644
--- a/arch/x86/syscalls/syscall_64.tbl
+++ b/arch/x86/syscalls/syscall_64.tbl
@@ -329,6 +329,8 @@
 320    common  kexec_file_load             sys_kexec_file_load
 321    common  bpf                 sys_bpf
 322    64      execveat            stub_execveat
+330    common  cs300_test          sys_cs300_test
+


 #
 # x32-specific system call numbers start at 512 to avoid cache impact
diff --git a/cs300/Makefile b/cs300/Makefile
new file mode 100644
index 0000000..e1e14ad
--- /dev/null
+++ b/cs300/Makefile
@@ -0,0 +1 @@
+obj-y := cs300_test.o
diff --git a/cs300/cs300_test.c b/cs300/cs300_test.c
new file mode 100644
index 0000000..d0c9b36
--- /dev/null
+++ b/cs300/cs300_test.c
@@ -0,0 +1,15 @@
+#include <linux/kernel.h>
+
+// Implement a HelloWorld system call
+// Argument is passed from user's call in user space.
+asmlinkage long sys_cs300_test(int argument)
+{
+      long result = 0;
+
+      printk("Hello World!\n");
+      printk("--syscall argument %d\n", argument);
+
+      result = argument + 1;
+      printk("--returning %d + 1 = %ld\n", argument, result);
+      return result;
+}
--
2.1.0
```

1.3

1.4 Applying a Git Patch

1. Switch to your Linux kernel source folder:
   `$ cd linux-stable/`
2. For testing your patch, you may want to create a new branch and apply the patch to that branch:

- List the branches you have in Git:
  `$ git branch`
- Switch back to the `master` branch:
  `$ git checkout master`
- Create a new branch for testing the patch and switch to it:
  ```
  $ git branch patch-test-1
  $ git checkout patch-test-1
  ```
- Now apply the patch to this new branch to test it out (next few steps).
- You can remove a branch and all its modified files:
  `$ git branch -D patch-test-1`

3. View a summary on what will change when the patch is applied (but not apply it yet):
   `$ git apply --stat path/to/patch.diff`
4. Check if the patch *can* be applied cleanly (but not apply it yet):
   `$ git apply --check path/to/patch.diff`
   - No output means no problems detected.
5. Apply the patch using Git's `am` command (designed for applying patches from a mailbox):
   `$ git am < path/to/patch.diff`
   - It's OK if it gives warnings about whitespace after EOF.
   - Note that if you are applying the patch on the same machine as you did the development that unchecked-in files will be present in all branches (they are not deleted when you switch between branches), so your build may still work even though your patch is incomplete.

2. Generating a Patch without Git

It is advisable that you create a ZIP file of the files you modified during your work so that if anything goes terribly wrong you don't lose any of your work.

1. Create a folder which contains the exact source code you started with (base version) before editing the kernel. If you extracted the provided ZIP file of the code, you'll need a fresh copy of this extracted to another folder.
   - Clean the newly copied Linux source code:
     ```
     $ make clean
     $ make mrproper
     ```
   - If you are out of space, try extracting it to a USB memory stick and using that. It will be slower but should still work.
2. Switch to your Linux kernel source folder where you have done your development (i.e., the folder containing your changes):
   ```
   $ cd linux-stable/
   ```
3. Do a full clean on your Linux source folder. This will remove your configuration file and all build products:
   ```
   $ make clean
   $ make mrproper
   ```
4. Switch to the parent directory of your Linux kernel source folder:
   ```
   $ cd ..
   ```
5. Create the diff:
   ```
   $ diff -rupN ../path/to/original/copy/ linux-stable/ > patch.diff
   ```
   - For example, on my system if I'm in the parent folder of my Linux kernel code (which is in a folder named linux-stable-v3.19.1/), and have a copy of the original code at the path ../orig/ so I use:
     ```
     $ diff -rupN ../orig/ linux-stable-v3.19.1/ > patch.diff
     ```
   - General form:
     ```
     $ diff -rupN original-copy/ modified-copy/ > patch.diff
     ```
6. **Check that your `.diff` file contains changes on only those files which you expected.**
   - **If the diff include build products it will be huge! Your final `.diff` file should be no more than a few Kb.**
     - **Run both of the following commands on both your original kernel source folder, and the folder containing your modified kernel code:**
       ```
       $ make clean
       $ make mrproper
       ```
   - **If your diff contains incorrect files or changes, re-run `make mrproper` and double check that you are using the correct base version of your kernel source files.**
7. Example `patch.diff` for kernel source modified by following the syscall guide (note that the first folder in your paths will likely be different; this is not a problem based on how patches are applied):

```
diff -rupN ../orig/arch/x86/syscalls/syscall_64.tbl linux-stable-
v3.19.1/arch/x86/syscalls/syscall_64.tbl
--- ../orig/arch/x86/syscalls/syscall_64.tbl 2015-03-22 00:16:20.000000000 -0700
+++ linux-stable-v3.19.1/arch/x86/syscalls/syscall_64.tbl  2015-04-06 20:54:54.609996492 -0700
```

```
@@ -329,6 +329,8 @@
 320    common  kexec_file_load              sys_kexec_file_load
 321    common  bpf                  sys_bpf
 322    64      execveat             stub_execveat
+330    common  cs300_test           sys_cs300_test
+


 #
 # x32-specific system call numbers start at 512 to avoid cache impact
diff -rupN ../orig/cs300/cs300_test.c linux-stable-v3.19.1/cs300/cs300_test.c
--- ..//cs300/cs300_test.c    1969-12-31 16:00:00.000000000 -0800
+++ linux-stable-v3.19.1/cs300/cs300_test.c 2015-04-06 20:54:54.613996492 -0700
@@ -0,0 +1,15 @@
+#include <linux/kernel.h>
+
+// Implement a HelloWorld system call
+// Argument is passed from user's call in user space.
+asmlinkage long sys_cs300_test(int argument)
+{
+       long result = 0;
+
+       printk("Hello World!\n");
+       printk("--syscall argument %d\n", argument);
+
+       result = argument + 1;
+       printk("--returning %d + 1 = %ld\n", argument, result);
+       return result;
+}
diff -rupN ../orig/cs300/Makefile linux-stable-v3.19.1/cs300/Makefile
--- ../orig/cs300/Makefile    1969-12-31 16:00:00.000000000 -0800
+++ linux-stable-v3.19.1/cs300/Makefile     2015-04-06 20:54:54.613996492 -0700
@@ -0,0 +1 @@
+obj-y := cs300_test.o
diff -rupN ../orig/Makefile linux-stable-v3.19.1/Makefile
--- ../orig/Makefile  2015-03-22 00:18:20.000000000 -0700
+++ linux-stable-v3.19.1/Makefile    2015-04-06 20:54:54.641996493 -0700
@@ -879,7 +879,7 @@ export mod_sign_cmd



 ifeq ($(KBUILD_EXTMOD),)
-core-y        += kernel/ mm/ fs/ ipc/ security/ crypto/ block/
+core-y        += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ cs300/

 vmlinux-dirs  := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
                      $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
```

2.1 Applying a non-Git Patch

1. Change into your kernel source folder:
   `$ cd linux-stable/`
2. Apply the patch:
   `$ patch -p1 < /path/to/diff/file.diff`
   - The `-p1` strips off one level of paths from in the `.diff` file before trying to apply the changes to the current folder. This is why having different base paths for your modified Linux kernel source code (and for the original copy) do not affect the ability to apply a patch on another computer.

For example:

```
$ patch -p1 < ../patch_via_diff.diff
patching file arch/x86/syscalls/syscall_64.tbl
patching file cs300/cs300_test.c
patching file cs300/Makefile
patching file Makefile
```