Lauren Fromm
404751250

Lab 9 Log

Since git is installed already on the linux
servers, the first thing I do is change my
path so it can be used:
$ export PATH=/usr/local/cs/bin:$PATH

Then, I need to make a directory to install git:
$ mkdir gitroot
$ cd gitroot

Then I use:
$ git clone git://git.savannah.gnu.org/diffutils.git
To get the diffutils folder. This outputs:

Cloning into 'diffutils'...
remote: Counting objects: 4944, done.
remote: Compressing objects: 100% (1345/1345), done.
remote: Total 4944 (delta 3537), reused 4944 (delta 3537)
Receiving objects: 100% (4944/4944), 1.40 MiB | 2.91 MiB/s, done.
Resolving deltas: 100% (3537/3537), done.

So I know the folder was downloaded successfully.

I go into the diffutils directory:
$ cd diffutils

And then get a log of changes into git-log.txt
$ git log > git-log.txt

I do the same thing for getting a list of tags:
$ git tag > git-tag.txt

Next, I go into the log:
$ emacs git-log.txt
and I search for the commit entitled “maint: quote
'like this' or "like this", not `like this'”. The whole
commit is:

commit 62ca21c8c1a5aa3488589dcb191a4ef04ae9ed4f
Author: Paul Eggert <eggert@cs.ucla.edu>
Date: Wed Jan 25 23:46:16 2012 -0800

    maint: quote 'like this' or "like this", not `like this'

    This is in response to a recent change in the GNU coding
standards,

```
    which now suggest quoting 'like this' or "like this", instead of
    `like this' or ``like this''.
    * HACKING, NEWS, README, README-hacking, TODO, doc/diagmeet.note:
    * doc/diffutils.texi, ms/config.bat, ms/config.site:
    * src/analyze.c, src/cmp.c, src/context.c, src/diff.c:
    * src/diff.h, src/diff3.c, src/dir.c, src/ifdef.c, src/io.c:
    * src/sdiff.c, src/side.c, src/system.h, src/util.c:
    * tests/help-version:
    Quote 'like this' or "like this" in commentary.
    * cfg.mk (old_NEWS_hash): Adjust to reflect new NEWS quoting.
    * man/help2man: Update to 1.40.4 version, with quoting fixed as
above.
    * po/en.po: Remove translation involving `, as it's no longer
needed.
    * src/cmp.c (try_help, specify_ignore_initial, usage, main):
    * src/diff.c (main, try_help, option_help_msgid, specify_value)
    (compare_files):
    * src/diff3.c (main, try_help, option_help_msgid, usage)
    (read_diff):
    * src/dir.c (compare_names):
    * src/sdiff.c (try_help, usage, check_child_status, main):
    * src/util.c (finish_output):
    * tests/help-version:
    Quote 'like this' in output.

To apply a patch, I use the commit number and the command
format-patch:
$  git format-patch -1 62ca21c8c1a5aa3488589dcb191a4ef04ae9ed4f --
stdout > quote_patch.txt

I then need to checkout version 3 of diffutils
$ git checkout v3.0

which outputs:

Note: checking out 'v3.0'.

You are in 'detached HEAD' state. You can look around, make
experimental
changes and commit them, and you can discard any commits you make in
this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you
may
do so (now or later) by using -b with the checkout command again.
Example:

  git checkout -b new_branch_name
```

```
HEAD is now at 022cd5c... version 3.0
```

Next, I need to use the patch command to apply quote_patch.txt to version
3.

```
$ patch -p1 < quote_patch.txt
```

There were a few errors that came up and I had to skip past some of them
by hitting enter. But after everything, there is an output of all the successful places the patch was used.

I then use

```
$ git status
```

and see:

```
# HEAD detached at v3.0
# 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:   NEWS
#       modified:   README
#       modified:   TODO
#       modified:   doc/diagmeet.note
#       modified:   ms/config.bat
#       modified:   ms/config.site
#       modified:   po/en.po
#       modified:   src/analyze.c
#       modified:   src/cmp.c
#       modified:   src/context.c
#       modified:   src/diff.c
#       modified:   src/diff.h
#       modified:   src/diff3.c
#       modified:   src/dir.c
#       modified:   src/ifdef.c
#       modified:   src/io.c
#       modified:   src/sdiff.c
#       modified:   src/side.c
#       modified:   src/system.h
#       modified:   src/util.c
#       modified:   tests/help-version
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
```

```
#       NEWS.orig
#       README-hacking.orig
#       README-hacking.rej
#       README.orig
#       README.rej
#       cfg.mk.orig
#       cfg.mk.rej
#       git-log.txt
#       git-tag.txt
#       ms/config.site.orig
#       quote_patch.txt
#       src/cmp.c.orig
#       src/context.c.orig
#       src/diff.c.orig
#       src/diff.c.rej
#       src/diff.h.orig
#       src/diff3.c.orig
#       src/diff3.c.rej
#       src/dir.c.orig
#       src/sdiff.c.orig
#       src/system.h.orig
#       src/util.c.orig
#       tests/help-version.orig
no changes added to commit (use "git add" and/or "git commit -a")
```

I then want to learn what vc-diff and vc-revert do in emacs.
I see that vc-diff displays a diff which compares each work file in the current VC fileset to the version from which you started editing. Vc-revert reverts the work file in the current fileset to the last revision.

To revert all the files besides the .c files, I go through each of the files and use vc-revert.
The following are the files I need to revert:

```
NEWS
README
TODO
doc/diagmeet.note
ms/config.bat
ms/config.site
po/en.po
src/diff.h
src/system.h
tests/help-version
```

For each of the file, I edit using emacs. Then I use C-x v u to revert them to their original, and then type yes.

Next I want to undo all the changes to .c files other than changes to character string constants. I do this by using vc-diff. I need

to look at:
src/analyze.c
src/cmp.c
src/context.c
src/diff.c
src/diff3.c
src/dir.c
src/ifdef.c
src/io.c
src/sdiff.ccu
src/side.c
src/util.c

I do this by using C-x v = to see the differences. I go through
each difference and see if it is a change to a character string.
If it is, I leave it, if not, I change it by changing the ' to `.
Some of the files, like analyze.c, side.c, ifdef.c, and io.c
only had changes in the comments, so those were able to be completly
reverted using C-x v u.

I next want to examine src/*.c.rej. UsingL
$ emacs src/*.c.rej
brings up two windows, one with diff3.c and the other with diff.c.
I use a different terminal window to edit these files and manually
add the change that the patch missed. These changes failed because
the line numbers didn't match, so I had to look through the files
using C-s.

I then want to remove all untracked files that git status warned.
These are:
#	NEWS.orig
#	README-hacking.orig
#	README-hacking.rej
#	README.orig
#	README.rej
#	cfg.mk.orig
#	cfg.mk.rej
#	git-log.txt
#	git-tag.txt
#	ms/config.site.orig
#	quote_patch.txt
#	src/cmp.c.orig
#	src/context.c.orig
#	src/diff.c.orig
#	src/diff.c.rej
#	src/diff.h.orig
#	src/diff3.c.orig
#	src/diff3.c.rej
#	src/dir.c.orig
#	src/sdiff.c.orig

```
#       src/system.h.orig
#       src/util.c.orig
#       tests/help-version.orig

To do this, I use:
$ git clean
which outputs:

Removing #NEWS#
Removing #TODO#
Removing NEWS.orig
Removing README-hacking.orig
Removing README-hacking.rej
Removing README.orig
Removing README.rej
Removing cfg.mk.orig
Removing cfg.mk.rej
Removing git-log.txt
Removing git-tag.txt
Removing ms/#config.site#
Removing ms/config.site.orig
Removing quote_patch.txt
Removing src/cmp.c.orig
Removing src/context.c.orig
Removing src/diff.c.orig
Removing src/diff.c.rej
Removing src/diff.h.orig
Removing src/diff3.c.orig
Removing src/diff3.c.rej
Removing src/dir.c.orig
Removing src/sdiff.c.orig
Removing src/system.h.orig
Removing src/util.c.orig
Removing tests/help-version.orig

Then, I use git status again to see the final modified files:
# HEAD detached at v3.0
# 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:   src/cmp.c
#       modified:   src/diff.c
#       modified:   src/diff3.c
#       modified:   src/dir.c
#       modified:   src/sdiff.c
#       modified:   src/util.c
#
no changes added to commit (use "git add" and/or "git commit -a")
```

I then want to use git diff to create a patch:
$ git diff > quote-3.0-patch.txt

I read the README-hacking to see how to build the version.
I am originally on lnxsrv07 but switch to lnxsrv02 since the
newer versions don't use the gets function. I then run the following:
$ ./bootstrap
$ ./configure
$ make
$ make check

These commands take a lot of time and ouput a lot, because
they're making a new executable. Finally, it finishes.

To check that everything works, I use
$ diff . -
$ diff --help

In both output uses ' instead of `, so we know that our patch worked.

To do a sanity check, I need to compare the old and new versions of
diff.

I first make the folders to put the different source codes.
$ mkdir diffutils-3.0-patch
$ mkdir diffutils-3.0

I then create another directory to get the original diffutils folder
so I can
extract just the source code.
$mkdir temp
$ git clone git://git.savannah.gnu.org/diffutils.git temp
$ cd temp

I then get version 3's source code by using:
$ git checkout v3.0
$ cp ./src/*.c ~/cs35l/lab9/gitroot/diffutils-3.0

This copies every .c file in src to the folder to compare.

I then do the same thing with my newer version.
$ cd diffutils
$ cp ./src/*.c ~/cs35l/lab9/gitroot/diffutils-3.0-patch

I run the diff command using:
$ ./diffutils/src/diff -pru diffutils-3.0 diffutils-3.0-patch >
quote-3.0-test.txt

I then want to use diff to compare this new text file and the

patch file:

```
$ ./diffutils/src/diff quote-3.0-test.txt diffutils/quote-3.0-
patch.txt
```

This outputs differences, but all of the differences are just the change
of ' to ` or just different file paths, so the differences don't matter
and are innoucous.