Assignment 9. Change management

Useful pointers

- Michael Johnson, Diff, Patch, and Friends, Linux Journal 28 (1996-08)
- Linus Torvalds, Jun Hamano *et al.*, Git - local branching on the cheap
- Scott Chacon, Pro Git (2009)
- Jacob Gube, Top 10 Git Tutorials for Beginners (2011)
- Sitaram Chamarty, The missing `gitk` documentation (2015)
- David MacKenzie, Paul Eggert, and Richard Stallman, Comparing and merging files, version 3.3 (2013-03-23)

You're helping to develop an operating system and command set that as part of its acceptance test is supposed to be used by a large government agency. The agency has lots of requirements, some sensible and some persnickety, and one of these requirements is that applications must use characters properly from the Unicode character set. In particular, applications must use the Unicode character "`" (grave accent, U+0060) only as a spacing accent character.

Unfortunately, one of your applications, GNU Diffutils, regularly uses "`" as a quoting character in diagnostics. For example, the command "`diff . -`" outputs the diagnostic "`diff: cannot compare `-' to a directory`", and this misuse of "`" violates your customer's requirements. You need to change Diffutils so that it outputs "`diff: cannot compare '-' to a directory`" instead, using an apostrophe (U+0027) for both opening and closing quote. You don't want to use fancier quoting characters such as "‘" and "’" (left and right single quotation marks, U+2018 and U+2019) because they are not ASCII characters and another customer requirement is that the programs must work in ASCII-only environments.

The good news is that the Diffutils maintainers have run into a similar problem, and have a patch called "maint: quote 'like this' or "like this", not `like this'" that does what you want. The bad news is that your customer has specified Diffutils version 3.0, and the patch that you want is not in version 3.0. Also, your customer is conservative and wants a minimal patch so that it's easy to audit, whereas the Diffutils maintainers' patch also affects commentary and documentation which your customer doesn't need or want changed.

Laboratory: Managing a backported change

As usual, keep a log in the file `lab9.txt` of what you do in the lab so that you can reproduce the results later. This should not merely be a transcript of what you typed: it should be more like a true lab notebook, in which you briefly note down what you did and what happened.

1. Get a copy of the Diffutils repository, in Git format, from the file `~eggert/src/gnu/diffutils` on the SEASnet GNU/Linux servers, or from its main Savannah repository.
2. Get a log of changes to Diffutils' master branch using the "`git log`" command, and put it into the file `git-log.txt`.
3. Generate a list of tags used for Diffutils using the "`git tag`" command, and put it into the file `git-tags.txt`
4. Find the commit entitled "maint: quote 'like this' or "like this", not `like this'", and generate a patch for that commit, putting it into the file `quote-patch.txt`.
5. Check out version 3.0 of Diffutils from your repository.
6. Use the `patch` command to apply `quote-patch.txt` to version 3.0. In some cases it will not be able to figure out what file to patch; skip past those by typing RETURN. Record any problems you had in applying the patch.
7. Use the `git status` command to get an overview of what happened.

8. Learn how to use the Emacs functions `vc-diff` (`C-x v =`) and `vc-revert` (`C-x v u`). When you're in the `*vc-diff*` buffer generated by `vc-diff`, use `describe-mode` (`C-h m`) to find out the Emacs functions that you can use there, and in particular learn how to use the `diff-apply-hunk` (`C-c C-a`) and `diff-goto-source` (`C-c C-c`) functions.
9. Use Emacs to revert all the changes to files other than `.c` files, since you want only changes to `.c` files. Also, **and don't forget this part**, undo all the changes to `.c` files other than changes to character string constants, as the character-string changes are the only changes that you want; this may require editing some files by hand.
10. Use Emacs to examine the files `src/*.c.rej` carefully, and copy rejected patches into the corresponding `.c` files as needed.
11. Remove all untracked files that `git status` warns you about, since you don't plan on adding any files in your patch.
12. When you're done with the above, `git status` should report a half-dozen modified files, and `git diff` should output a patch that is three or four hundred lines long. Put that patch into a file `quote-3.0-patch.txt`.
13. Build the resulting modified version of Diffutils, using the commands described in the file `README-hacking`, skipping the part about CVS; CVS is obsolescent. (If you are building on `lnxsrv07` or `lnxsrv09` or any other host that is using version 2.16 or later of the GNU C Library, you will need to apply an additional patch after running `./bootstrap` and before running `./configure`, because glibc 2.16 removed the obsolete and dangerous `gets` function declared by a Diffutils header.) Verify that Diffutils does the right thing with the "`diff . -`" scenario, as well as with "`diff --help`".
14. Do a sanity test using the modified version of Diffutils that you just built, by using the just-built `diff` to compare the source code of Diffutils 3.0 to the source code of your modified version. Put the former source code into a directory `diffutils-3.0` and the latter source code into a directory `diffutils-3.0-patch`, and run your implementation of `diff` with the command "*`D`*`/diff -pru diffutils-3.0 diffutils-3.0-patch >quote-3.0-test.txt`", where the *`D`* is the directory containing your `diff` implementation.
15. Use `diff` to compare the contents of `quote-3.0-test.txt` and `quote-3.0-patch.txt`. Are the files identical? If not, are the differences innocuous?

Homework: Verifying and publishing a backported change

You're happy with the code that you wrote in your lab, but now you'd like to publish this patch, in a form similar to that presented in the original patch, so that others can use it.

1. Maintain a file `hw9.txt` that logs the actions you do in solving the homework. This is like your lab notebook `lab9.txt`, except it's for the homework instead of the lab.
2. Check out version 3.0 of Diffutils from your repository, into a new branch named "`quote`".
3. Install your change into this new branch, by running the `patch` command with your patch `quote-3.0-patch.txt` as input.
4. Learn how to use the Emacs function `add-change-log-entry-other-window` (`C-x 4 a`).
5. Use this Emacs function to compose an appropriate ChangeLog entry for your patch, by adapting the change log from the original patch.
6. Commit your changes to the new branch, using the ChangeLog entry as the commit message.
7. Use the command "`git format-patch`" to generate a file `formatted-patch.txt`. This patch should work without having to fix things by hand afterwards.
8. Your teaching assistant will assign you a partner, who will also generate a patch. Verify that your partner's patch works, by checking out version 3.0 again into a new temporary branch `partner`, applying the patch with the command "`git am`", and building the resulting system, checking that it works with "`make check`".
9. Verify that your ChangeLog entry works, by running the command "`make distdir`" and inspecting the resulting `diffutils*/ChangeLog` file.
10. There is a copy of the GNU Emacs git repository's master branch on SEASnet in the directory `~eggert/src/gnu/emacs`. Run the command `gitk` on it, and find the newest merge that is not newer than

2015-01-25. Take a screenshot `gitk-screenshot.png` of your view of the mergepoint, and in an ASCII text file `gitk-description.txt` briefly describe the roles of subwindows that you see in the screenshot.

Submit

Submit a compressed tarball `hw9.tgz` containing the following files.

- The files `lab9.txt`, `git-log.txt`, `git-tags.txt`, `quote-patch.txt`, and `quote-3.0-patch.txt` as described in the lab.
- The files `hw9.txt` `formatted-patch.txt`, `gitk-screenshot.png`, and `gitk-description.txt` as described in the homework.

All `.txt` files should be ASCII text files, with no carriage returns. You can create the tarball with the command:

```
tar czf hw9.tgz lab9.txt git-log.txt git-tags.txt \
  quote-patch.txt quote-3.0-patch.txt hw9.txt formatted-patch.txt \
  gitk-screenshot.png gitk-description.txt
```

$Id: assign9.html,v 1.28 2017/01/25 00:24:31 eggert Exp $