CPE-2600 FALL 2024

Systems Programming

Lab 8 Git Some More

Topical Concepts

GitHub README and Markdown

"README" files have a long history in software development. Since 'C' programs have been historically distributed in source form, the README (or README.txt or readme.txt) files are included with the distribution and contain instructions for building the program, operator instructions, and known limitations and bugs (errata). README files are typically plain text files easily readable on any system. GitHub and other providers of git remote service embrace this tradition and if a README file is present in a repository, the web-based repository viewer will display the README file.

In order to allow for a richer view suitable for a web-viewer, GitHub also supports README files that are not just simple text files. To bridge human-readable text files with word-processor-like formatting, GitHub has adopted a flavor of Markdown to embed formatting into simple text files. Thus, for a GitHub repository, it is common to use README.md for the README file, and for GitHub to automatically render that file. There are many "flavors" of Markdown, so GitHub may or may not render everything the same as other Markdown renderers, but it seems to be pretty universal. There are a variety of guides available from GitHub and others to help out. One such guide is here: [Basic writing and formatting syntax - GitHub Docs](#).

Source Version Control – Next Steps

Periodic Commits

Ideally, as you continue to develop, you will commit changes to the git repository periodically. You can view a history of those commits with the command **`git log`**. Example:

```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git log
commit 8cbfcad61b0cb2382fe4a312432b710a681e4e4e (HEAD -> main, origin/main, origin/HEAD)
Author: Dr. Rothe <rothede@msoe.edu>
Date:   Sun Oct 15 20:51:44 2023 -0500

    Added help message to main

commit 8b60f47c84ac0150eb3b39c3270c5c0efd2954e2
Author: Dr. Rothe <rothede@msoe.edu>
Date:   Sun Oct 15 20:06:34 2023 -0500

    Changed length of name in vect typedef

commit fbe2732c9bdd3993815c763cbbfecf5cf329f1b9
Author: Dr. Rothe <rothede@msoe.edu>
Date:   Sun Oct 15 20:05:20 2023 -0500

    Changed num vectors

commit 59f6558555ba4ef4d3eff461e0ad074934ba7ba6
Author: Dr. Rothe <rothede@msoe.edu>
Date:   Sun Oct 15 20:03:56 2023 -0500

    Added an operation
```

We can see each commit's comments along with a timestamp and a hash of the repository at the time of the commit.

We can also, change our "view" to any of these versions of the repository. Now, some care must be exercised.

Temporary View - git checkout

The command **`git checkout <commit hash>`** can be issued to revert the working directory to the state of a previous commit.

```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git log --oneline
8cbfcad (HEAD -> main, origin/main, origin/HEAD) Added help message to main
8b60f47 Changed length of name in vect typedef
fbe2732 Changed num vectors
59f6558 Added an operation
be5824d Added printbyname method to vectarray
1a0568b Added .gitignore and added help message to main
9487b2e Initial Version
332db3e add deadline
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git checkout fbe2
Note: switching to 'fbe2'.

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 switching back to a branch.

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

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at fbe2732 Changed num vectors
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

In this example, note a few things. Firstly, you do not have to type the entire hash – it just needs to be long enough to be unique. Secondly, git is warning you of this 'detached HEAD' state. Take this warning seriously – we are just here to look around. In any case, your working directory contains the files in the state they were in when the select commit took place. Any edits committed after that are not seen.

So given these cautions, why are we doing this. Again, maybe just to look around – that is see the entire project at a previous state. Perhaps you added a feature in future commits that is not working properly, so you want to go back to a previous working state. Despite the caution not to make changes, this is your full working directory – you can still build and run or debug the program.

You can return to the most current commit with **`git checkout main`** for this example.

Tagging Commits

That brings us to one of our main topics, tagging commits. You have already seen that a commit requires a message describing the nature of changes made to a project for that commit. Occasionally you would like to mark a commit with a message where perhaps to were no changes at all. This sort of need would usually be associated with a milestone or product release of some sort. Git calls this a "tag" and really is not much different than a regular commit, but is handled distinctly to allow tags to be searched separate from regular commits, since you may have hundreds of commits over the span of development of a single milestone or release.

There are two types of tags. An "annotated tag" is somewhat analogous to a commit and stores a timestamp, message, and information about the tagger. This is the type of tag you would use for a release. A "lightweight tag" is really just an alias to an existing commit and not as versatile. Any future discussion or usage of tags will be referring to annotated tags.

Creating a tag is much like a commit, just a different command. For example:

`git tag -a v1.0 -m "Major Release"`

will create an annotated tag (-a) named v1.0 with message "Major Release."

One slight complication is that tags are not pushed to remote with git push like normal commits. You have to explicitly list the tag(s) you wish to push or --tags to commit all. Example:

`git push --tags`

Tags can be used with the 'checkout' command to set the working directory to the exact set of source that was present when tagged. Now, the purpose of checkout should be clear.

```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git checkout v1.0
Note: switching to 'v1.0'.

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 switching back to a branch.

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

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 8cbfcad Added help message to main
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

Note the same 'detached HEAD' state and the same cautions as before.

Since you have a record of all commits, it is easy enough apply tags later. Simply add the hash for the commit you wish to tag.

`git tag -a v0.1 -m "Minor Release" 59f6`

In this example, commit with a hash that starts with 59f6 would be tagged instead of HEAD.

Branching and Merging

This will be discussed in a future Lab.

The Exercise

Overview

For your current lab exercise, Vector Calculator Updates, you should have been committing update to your repository periodically. When you are finished with various updates, you will be tagging the new "release" as well as tagging your previous "release." In addition, you will be adding a README.md file to your repository.

Specific Instructions

1. Add a README.md file to your repository. You should already know how to add new files to the repository. In the README.md file, you should provide basic instructions typical of a README file. Items to include:
    a. A brief description of the program and how it works.
    b. How to build the program
    c. How to run the program – available command line options
    d. A list of commands supported by the program
    e. A short description of how your program uses dynamic memory (ok, not typical – think lab report…)

    Much of this information might already be embedded in your program – no problem if you wish to copy-and-paste to save some typing.

    Your README.md must make use of basic Markdown features. Specifically, headings and bulleted or numbered lists. You may wish to experiment with other Markdown features that may be used in future assignments.

    Note, VS Code will happily render Markdown, so it is easy to write your README.md in VS Code and expect it to look good in GitHub. Alternatively, take a stab in your text editor, commit the file and push to remote. You can further edit within the GitHub website to clean up any malfunctioning Markdown.

2. When your updates are complete and ready to demo, tag your repo **v2.0**. Make sure the README.md is part of the repo at this time. Be sure to push all commits and tags to remote.

3. After demoing the update, use **git checkout** to revert to your previous version that was demoed prior to adding dynamic memory and file i/o. Build and verify proper operation. Tag this commit with **v1.0**. Push all commits and tags. Also be sure to return HEAD to the latest commit at some point.

4. This is due by the date in Canvas.

Deliverable

Provide a lab report showing screen shots of the exercises in the lab. A demo checkoff of your repository will be completed in lab. I will expect to find a useful README.md as described above and at least two tags that will correspond to the initial Vector Calculator assignment and the Vector Calculator Updates.