**07 – Version Control & A Branching Workflow**

**Activities**

COMP190 – Tools and Techniques for Software Development

Dickinson College

Fall 2020

Prof. Grant Braught

In this set of activities, you will begin the process of learning about version control and how it helps in managing software (and other digital projects). We’ll then learn specifically about the version control software git and its companion web service GitHub. While we will be using git and GitHub, note that all of the things that we do also apply to other version control systems (SVN, Mercurial) and other web-based repository hosting services (e.g. GitLab, Bitbucket). The specific terminology and commands may be different, but the fundamental concepts are the same. These activities will walk you through a standard workflow for collaborative teams that use git and GitHub for software development. To practice with git and GitHub, we will imagine that we are working as a team on a piece of Java software for a calculator. It is going to be a relatively simple program with some obvious issues. The point of keeping the software simple, is to allow you to focus on the process of using git and GitHub before we eventually move on to use it in a more complicated project.

**Open Source Basics:**

Watch the video *Open Source Basics* video from Intel Software. This video introduces some of the ideas of how open source gets produced today. In fact, most proprietary software development happens in a similar manner, just with the source code and contributions all being private within a company.

* <https://www.youtube.com/watch?v=Tyd0FO0tko8&feature=youtu.be> (5:41)

1. After watching the video and personally thinking about the content, answer each of the following questions in a few sentences of your own words. Your answers should not simply paraphrase the video, but should show an understanding of the concepts involved.

a. What defines the rules of collaboration in an open source project?

b. What is a branch?

c. What is a maintainer and a contributor? How are they related?

d. What is upstreaming and why is it beneficial?

e. What is a fork and why might it happen?

**Version Control:**

The above video concluded by saying that open source software development with lots of contributors and branches is complicated. This means we need ways to structure the collaboration and interaction between maintainers and contributors of projects. This applies equally well to closed source projects, which in practice use the same tools as open source projects. These tools are call version control systems. To learn more about version control systems and what they do watch the *What is Version Control?* video by Brad Schiff of LearnWebCode:

* <https://www.youtube.com/watch?v=9GKpbI1siow> (9:35)

2. After watching the video and personally thinking about the content, answer each of the following questions in a few sentences of your own words. Your answers should not simply paraphrase the video, but should show an understanding of the concepts involved.

a. What is history with respect to version control and why is it beneficial?

b. How does version control support collaboration?

c. What are feature branches and why are they useful?

**Preliminaries:**

Now that we have some of the main concepts and ideas down, let’s put it into practice. But before you can actually get started working with a project using git and GitHub, you’ll need to take care of some preliminaries. You’ll need to have a GitHub account and you will need to have git installed on your LinixLite distribution.

*Getting a GitHub Account:*

3. Visit GitHub.com and either “Sign up” for an account or if you already have an account confirm that you can “Sign in”. Once you have an account and can sign in give your GitHub username as your answer to this question.

*Installing git:*

4. Use apt-get or the synaptic package manager to install git on your LinuxLite virtual machine. What output is generated when you run the command git --version in the terminal?

5. A little configuration will make your git installation easier to use. Enter each of the following commands in the terminal, substituting in the appropriate values for you.

git config --global user.name <GitHubUserName>

git config --global user.email you@email.com

git config --global push.default simple

git config --global credential.helper cache

Give the output of the command git config --global --list as your answer to this question.

**Getting the Project:**

Working on an existing project on GitHub is a two-step process. First you *fork* the project repository into your GitHub account. Second, you *clone* the copy from your GitHub account to your local machine.

*Forking*

Any significant project on which you will work is likely to maintain a repository on a git compatible hosting site (e.g. GitHub) that contains all of the code for the project. The project's repository will be read-only except for the project maintainers. So, to work on the code for the project you will create a copy of the repository in your own GitHub account. The process of making this copy is called *forking.*

6. Fork the Calculator Project to your GitHub by:

a. Log into GitHub.

b. Go to <https://github.com/dickinson-comp190/Calculator>

c. Click the “Fork” button in the upper right:



d. If a pop-up dialog appears, select your GitHub account from those shown.

e. Return to your GitHub account and verify that you now have a repository that is a fork of the Calculator project by visiting the main page for the repository.

f. Give the full URL of your Calculator project from the browser as your answer to this question. If the URL does not contain both your GitHub username and the word “Calculator” then you are not on the correct page.

7. When you fork a Main Project repository on GitHub, what is the more technical name for the repository that you forked? (Hint: See “Some More Detail & Terminology in the video.)

*Cloning*

The fork of the upstream repository is in your GitHub account and thus you will be able to both read and write to it. It will provide cloud storage for all of the work you do on the project and will also be the means for contributing code back to the main project through pull requests. To work on the code however, you will need to *clone* this repository onto your local machine.

8. Clone your GitHub repository for the Calculator project to your LinuxLite virtual machine by:

a. In GitHub:

i. Ensure that you are on the Calculator repository in your GitHub account. The repository name in the upper left should be prefixed with your GitHub username.

ii. Click the green “Code” button:



iii. Check that the URL listed starts with https and contains your GitHub username.

iv. Click the clipboard icon to copy the repository URL.



b. In a terminal on your LinuxLite virtual machine:

i. Navigate to the directory where you would like to store this project (e.g. Documents).

ii. Use the command:

git clone <OriginRepoURL>

iii. The git clone command should have created a local copy of the repository. This will consist of a directory named Calculator that is populated with the files for the project. Verify that you now have a local copy of the repository.

iv. Give as your answer to this question a list of all of the files in the Calculator directory, including any hidden files (Hint: use a flag with ls to show all files).

8. What is the more technical name for the copy of the repository that is now in your GitHub account? (Hint: See “Some More Detail & Terminology in the video.)

9. Open the Calculator.java file in a text editor and scan through the code. It is polluted with errors that will cause the program to produce incorrect results. Describe one of the errors that you notice. Do not fix any of these errors yet… we’ll get to that. Note: If you want more of an Integrated Development Environment (IDE) for working with code you can install Atom or Eclipse using the package manager.

**Working on the Project:**

Once you have cloned the origin to your local machine you are ready to begin working on the project.

*The Issue Tracker*

Most projects will use an issue tracker to keep track of bugs that need to be fixed and new features that are to be added. You can find the issue tracker for our Calculator project by going to the upstream repo (<https://github.com/dickinson-comp190/Calculator>) and clicking on the “Issues” tab at the top:



10. Scan through the issues that have the tags “bug” and “good first issue” and find the one that corresponds to the bug you found in question 9. What is the title and issue number of that bug?

**Creating a Branch and Checking it out:**

Now that you’ve found something in the project that needs to be fixed we can practice a common “branching workflow” that is often used with git and GitHub. The term *branch* simply means a copy of the project. When working on a project you will usually want to start from the most up to date branch. In the Calculator project this branch is called the *main branch*. In other projects or on other sites this branch may have different names (e.g. trunk, release, live, or master*[[1]](#footnote-1)*).

In practice you will never edit code on the main branch. Instead, each time you want to work on the code (add a feature, fix a bug, etc) you will create a new *feature branch* and edit the code there. The feature branch behaves like it is a full copy of the main branch, but git is smart enough to do this efficiently without making a full copy of all of the files. By editing code on your feature branch, you will be able to leave the main branch in a state that is consistent with the upstream repository. This will be very helpful in integrating changes made by others into your local and origin repositories. It also has the advantage of making it easy to discard changes you have made in your feature branch if you decide they are not working out.

11. The command git branch -a will list all of the branches that exist in your local repository. What output is generated by this command?

12. An \* indicates the currently active branch. That is, the copy of the files in that branch are what appear in your working directory and are what will be changed if you make edits. What branch is currently active?

13. The first step in starting to work on fixing a bug or adding a feature will be to create a new feature branch. The command below will create a new branch:

git branch <NewBranchName>

Use this command to create a new branch with a name that is related to the issue you identified in question 10. Give the full command that you used.

14. Give the output that you see now when you list all of the branches?

15. Does creating a new branch make it the active branch? How could you tell?

16. To change to another branch (i.e. make it the active branch) the command to use is:

git checkout <BranchName>

Use this command to checkout the branch that you created in question 13.

17. Check that your new branch is now the active branch. How did you do this?

**Editing the Code:**

Now that you have a feature branch created any edits that you make to the code will be done using the “copies” of the files in that branch rather those in the main branch.

18. Before beginning an edit it is a good idea to be sure that everything is up to date and that you are on the right branch. The command below will tell you what branch you are on and if you have made any changes that have not been committed (more on what that means soon):

git status

What output is generated by this command?

18. Using an editor of your choice, fix the bug described in the issue that you identified in question 10. Be sure to save the changes to your file. Copy and paste just the line that you fixed as your answer to this question.

19. What does git status now report about the file that you changed?

**Staging and Committing Your Changes:**

When you edit and save a file you are changing its contents and saving those changes in the working directory. However, this does not change the files that are stored in the feature branch in the local git repository. To place the changes into the repository you will need *stage* them by *adding* them to the staging area and then *commit* them to the repository. In general, a commit should represent the completion of a unit of work (e.g. a bug fix, or a new feature being added). Thus, you should save often and then add and commit when you complete a task.

Given that you have completed the bug fix for the issue that you took on above, it is now time to add the file(s) containing the changes to the staging area and then commit them to the local repository.

20. The git status command you used in question 19 showed that the Calculator.java file had been modified but was not yet staged for commit. To stage a file that has been changed for commit use the command:

git add <FileToStage>

What does git status report after the file has been staged?

21. If the changes you have made to fix a bug or add a feature are spread across many files you would want to add all of them to the staging area. Sometimes in that process you mistakenly add a file that should not have been added. What command could you use to remove the Calculator.java file from the stage? Hint: Read the last output from git status.

22. Now that the changes have been staged we need to commit them to the feature branch in the local repository. To commit your staged changes use the command:

git commit -m '<A descriptive commit message>'

The commit message should be something that describes the unit of work that was completed (e.g. the bug that was fixed or the feature that was added). This is the message that other developers see when they look at the project history. Commit your changes and give the full command you used, including the commit message?

23. What does git status report after you have committed the changes?

24. To see a history of the changes that have been made to the repository you can use the command:

git log

Who made the commit just before yours? What was changed?

25. Let’s now experiment with the branches a little to gain a little better understanding of how they work.

a. Use cat to display the Calculator.java file. Is the change you made there?

b. Checkout the main branch. Give the full command that you used.

c. Use cat to display the Calculator.java file. Is the change you made there? Briefly explain why or why not?

**Pushing a Branch to Origin:**

26. What branches are there in the origin repository? To find this out, go to Calculator project in your GitHub. Confirm that your username appears in the URL. Open the “<> Code” tab (if it is not already open):



Then click on the branches drop down:



You have committed the changes you made to the feature branch in your local repository. But they are not in your origin repository on GitHub or in the upstream repository either. To get your changes into the origin repository and then hopefully integrated into the upstream, you need to *push* your feature branch to the origin.

27. To push a branch to the origin the command is:

git push origin <BranchName>

Use this command to push your feature branch to the origin repository in your GitHub account. Check the branches drop down in GitHub now. What is listed there now?

**Making a Pull Request to the Upstream:**

Now that the feature branch containing your bug fix is on GitHub, you can request that the maintainers of the upstream integrate it into the upstream repository. The mechanism for doing this on GitHub is called a *Pull Request*, which is often abbreviated as *PR*. Other repository hosting platforms also have this functionality, but they may use a different name. For example, GitLab calls this a Merge Request.

28. A pull request (PR) is made in the browser using the GitHub user interface as follows:

a. Click the branch drop down:



and choose the feature branch that contains the changes you want to propose for integration into the upstream.

b. You will see a message similar to:



Click on “Pull request.” Note that if you go to make your pull request shortly after pushing the feature branch then there will be a shortcut for this. You may see a banner like:



Clicking on the green “Compare & pull request” button will have the same effect.

c. Examine your pull request to ensure that it is configured correctly.



The “base repository” should be set to the main branch of the upstream where you want your changes to go. The “head repository” should be set to the feature branch that contains your proposed changes.

d. Scroll down to the bottom of the page. There you will see two panels that compare the base and head repositories. Do you see your changes? Do the changes you made appear in the left panel or the right panel?

e. Scroll back up. Write a commit message in the comment block explaining what you have done. In general, this will include more information than your commit message. Note that your commit message is displayed lower on the page, so there is no need to repeat it.

f. Click the “Create pull request” button:



When you pull request has been submitted you will see a page with information something like the following:



There is nothing more to turn in for this question. As the maintainer of the upstream Calculator repository I will see your PR and will count that as your submission for this question.

**Issue, Branch, Edit, Add, Commit, Push, PR Repeat:**

While you started by finding a bug in the code earlier, often the process of bug fixing begins by finding an issue in the Issue tracker. Then finding the corresponding code that needs to be fixed.

29. Using the issue tracker, find another issue that is tagged “bug” and “good first issue”. What is the number and title of the issue you selected?

30. Repeat the process of creating a branch, fixing the issue, adding and committing your changes, pushing the branch and creating a PR for at least one more issue in the Calculator project. The PR you create will be your answer for this question.

**Review & Reflection:**

31. Create a list of all of the git commands that were used in this lab and include one or two sentences describing what each does.

1. Historically the main branch had been called the master branch. GitHub makes it possible to change this default and will formally move to main being the default for any new projects beginning October 1, 2020. (See: <https://github.com/github/renaming>) Because of this history you are likely to find many projects where the main branch is still called the master branch. [↑](#footnote-ref-1)