**Github as a Configuration Management System**

Samantha M. Hipple

MSCS, City University of Seattle

CS504: Software Engineering

Kendra Schraml

July 18, 2021

**Github as a Configuration Management System**

Students were asked to prepare written test cases concerning the functionality of Github as an appropriate Configuration Management System for a software engineering project. Test cases are used to ensure a consistent structure and improve repeatability and record keeping such as trackability, monitoring and reporting of any changes when managing a team software engineering project. This paper reviews the creation of a simple sample software project that is uploaded to Github by account name Hipples, in order to demonstrate how Github allows team collaboration via the following themed test cases: code check-out, code check-in, code snapshot/release point, code roll-back, and a bug fix in a previous release.

**Github Collaboration Test Case Summaries**

**1. Code Check-Out**

***Assumptions & Test Data Required***

This initial test case (tc\_github\_01) assumes that Hipples has already (1) created an IP repository, (2) uploaded a sample software program for collaboration, and (3) invited another Github user (e.g., swiseWHAT) to collaborate on the project as shown in Figure 1 below.

Graphical user interface, text

Description automatically generated

Figure 1. Screenshot of Github collaboration invitation email.

***Steps***

The steps taken for this test case include: (1) find Github invitation email, (2) view/accept the invitation, (3) clone repository into an alternate directory for user swiseWHAT, (4) open IP01 – Windows Calculator.sln to verify a successful code check-out.

***Expected vs Actual Results***

The expected result of the above steps is that user swiseWHAT is able to successfully clone the repository shared by Hipples into their own directory and open/edit the code in the Windows Calculator sample project that Hipples uploaded for collaboration. Figures 2 and 3 below provide verification of the actual results.

Text

Description automatically generated

Figure 2. Screenshot of successful Hipples repository clone to swiseWHAT user account.

Graphical user interface, text, application

Description automatically generated

Figure 3. Screenshot of successfully reaching editing phase of Hipples’ coding project via swiseWHAT’s download.

***Pass/Fail & Comments***

Pass. During tc\_github\_01, user swiseWHAT was able to successfully accept Hipples collaboration invitation, clone the software project repository to their own directory, and open/edit code that Hipples had previously uploaded.

**2. Code Check-In (attempt 1)**

***Assumptions & Test Data Required***

This second test case assumes that tc\_github\_01 was a success and requires the repository to have been cloned, edited, and updated/saved within swiseWHAT’s personal folder before continuing.

***Steps***

The steps taken for this test case are: (1) open cmd prompt, (2) set directory to swiseWHAT’s folder, (3) enter the following commands:

* *git add .*
* *git commit -m “Update 01 from swiseWHAT folder”*
* *git push origin main*

(4) sign into Github browser on Hipples account and verify updates were posted by swiseWHAT.

***Expected vs Actual Results***

The expected results for the above steps were that Hipples would find successfully updated files by swiseWHAT in the shared repository upon sign in to the admin account. Figure 4 below illustrates step three for tc\_github\_02. The actual results showed that the updates had uploaded correctly from newly cloned repository; however, they were stated to be uploaded by Hipples and not swiseWHAT as expected – which can also be seen in Figure 5 below.

Text

Description automatically generated

Figure 4. Uploading repository updates from swiseWHAT folder.

Graphical user interface, text, application, email

Description automatically generated

Figure 5. Update 01 from swiseWHAT folder was committed by Hipples, not swiseWHAT.

***Pass/Fail & Comments***

Fail. Although the data successfully uploaded, the records show it was pushed by the Hipples account and not swiseWHAT. To resolve this issue, an alternative windows user account for swiseWHAT will be created and the repository will be re-downloaded to swiseWHAT’s personal user directory instead of using a folder within Hipples Window’s user account. This should allow swiseWHAT’s updates to record as being committed by swiseWHAT’s Github account instead of the Hipples account.

**3. Code Check-In (attempt 2)**

***Assumptions & Test Data Required***

The third test case (tc\_github\_03) assumes that one has (1) successfully created a separate Window’s user account for swiseWHAT, (2) cloned Hipples Github repository to the new user’s directory, (3) updated the project’s repository in some manner and saved the changes to swiseWHAT’s new directory. Figure 6 below illustrates the creation of the new Window’s user account for swiseWHAT.

Text

Description automatically generated

Figure 6. Creation of separate Window user account for swiseWHAT Github contributions.

***Steps***

The steps for tc\_github\_03 are nearly identical to tc\_github\_02: (1) open cmd prompt, (2) set directory to the newly cloned repository, (3) *git commit -m "Update 03 from swiseWHAT account"*,and (4) sign into Github browser on Hipples account to check the update results. Figure 7 below records swiseWHAT’s portion of these steps, including the use of *git pull* when the initial push attempt failed to commit.

Text

Description automatically generated

Figure 7. Merging (pulling) Hipples Github repository with swiseWHAT’s updates.

***Expected vs Actual Results***

The expected result for this test case is for user Hipples to find an updated repository with commits from user swiseWHAT upon sign-in to the admin account for the shared repository. Figure 8 is a screenshot verifying the new repository updates were successfully committed by swiseWHAT’s Github account.

Graphical user interface, text, application, email, Teams

Description automatically generated

Figure 8. Screenshot of Update 03 being committed by swiseWHAT account in shared repository.

***Pass/Fail & Comments***

Pass – the shared repository has successfully accepted a commit from another Github user account – demonstrating how one would check-in code during a collaborative software project.

**4. Code Snapshot/Release**

***Assumptions & Test Data Required***

For this test, one may use any collaborator account.

***Steps***

1. Sign into a Github account with access to the sample repository.
2. Select the link labeled “Releases”.
3. Draft a new release, including Tag version & description.
4. Publish release.

***Expected vs Actual Results***

The expected and actual results were the same – Hipples was able to successfully release v1.0 of the shared repository as demonstrated with the figures below.

Graphical user interface, text, application, email

Description automatically generated

Figure 9. Creating a snapshot/release v1.0 of the shared sample repository.

Graphical user interface, text, application, email

Description automatically generated

Figure 10. Screenshot of the published IP01 Snapshot/Release, v1.0.

***Pass/Fail & Comments***

Pass. Official release v1.0 for the sample project was published.

**5. Commit Rollbacks and Debugging Previous Releases**

***Assumptions & Test Data Required***

The repository in question has been through multiple updates/releases. There is more than one branch (main & master, in this sample case) that is being updated by the different collaborators. This prerequisite step can be seen in Figure 11 below.

Text

Description automatically generated

Figure 11. Git checkout Head~1 command (reverting project by one commit).

***Steps***

1. Use the command prompt to checkout Head~1

2. Create a master branch to save the rolled back repository’s updates

3. Update information/documents within the master branch repository

4. Commit updates to the master branch & check Github browser for accuracy

***Expected vs Actual Results***

These steps should demonstrate the ability to work on a rolled back version of a repository and commit the changes without effecting the most current main branch’s repository. Afterwards, either collaborator should be able to review and merge the changes into the main branch if desired. The actual results of this test case matched the expected results and are recorded via screenshots in the figures below.

Text

Description automatically generated

Figure 12. Committing updates to a master branch instead of the main branch of the project.

Graphical user interface, text, application

Description automatically generated

Figure 13. Master branch is 1 commit ahead and 1 commit behind the main branch.

Graphical user interface, text, application

Description automatically generated

Figure 14. Main branch shows commit in submission-02 folder that was removed (rolled back) during the creation of the master branch shown in Figure 13.

***Pass/Fail & Comments***

Pass. This test case successfully demonstrated the ability for users to work on any current or previous version of their repositories and store the updates as separate branches as needed to better track changes/versions of software releases. Although this test case does not explicitly demonstrate the debugging of a “previous release”, the steps would be similar enough that the two cases have been combined here.

**Conclusion**

This document demonstrates basic functions of Github as a CMS as promised – however, it was not until the final step of this process that the author realized code check-in and check-out can be differentiated (or, at least, has its own set of CL terms) from simply pushing and pulling a repository. Additionally, we learned during this process that one must be accessing their Github repository from this own personal user accounts if they want to easily ensure that the updates they push, accurately reflect their account as the contributor. The test cases above can also be found in a charted/outlined version of this document as well, created in Excel. No actual code was tested/edited during the creation of this document, all test cases simply involved updating various documents within the repository.

**References**

Gienow, M. (2019, May 31). *Getting legit with git and github: Rolling back changes with revert and Reset.* The New Stack. https://thenewstack.io/getting-legit-with-git-and-github-rolling-back-changes-with-revert-and-reset/.

*Releasing projects on github*. GitHub Docs. (n.d.). https://docs.github.com/en/github/ administering-a-repository/releasing-projects-on-github.

*Reverting a commit.* GitHub Docs. (n.d.). https://docs.github.com/en/desktop/contributing-and-collaborating-using-github-desktop/managing-commits/reverting-a-commit.