Source Control Evaluation

The votes are in! Here are the top 5 needs we want fulfilled through our source control system, as selected by each course section:

|  |  |  |
| --- | --- | --- |
| MWF 10:00 am | MWF 1:30 pm | TH 10:00 am |
| 1. Save my stuff (I control when to save) 2. Sharing assets with a team    1. Security    2. Concurrent edits 3. Versioning    1. Log    2. Revert | 1. Keep all versions    1. never lose any code    2. Non-destructive edits    3. Choose when I push and pull 2. Remote access    1. IP protection, permissions    2. Concurrent edits 3. Must be easy to use    1. Resolving Conflicts    2. Useful error messages (or just don’t break!)    3. I18N, Accessibility 4. Cross platform | 1. Versioning, Save my work    1. No destructive edits 2. Team edits, Offline versioning    1. I can see the history without being connected    2. I can create a new version without being connected    3. Concurrent edits: Allows multiple people to edit a file at a time 3. Remote    1. Access from anywhere, internet connected    2. I can give/revoke access to users 4. GUI |
| Assumptions   * Our apps will always do what they say they will do (they will not lie and say “save successful” when in fact saving failed). * We can always save locally (no versioning) * We can save any type of file | | |

Your assignment is to evaluate Git and the GUI interface you selected (i.e. TortoiseGit) in groups against the features above selected from all sections.

Form your groups on the LMS under BIT120 Git Evaluation Groups. They may be the same or different from the groups you used in the in-class source control requirements activity. Groups must have 3-4 people, and there will be a peer review after the assignment for you to rate the effort and contributions of each group member.

As this may be your first time formally assessing software against requirements, I’ll guide you through with a series of questions and tasks.

# Prerequisites:

1. You have git and optionally a git gui installed on your system
2. You understand how to create a repository on GitHub, clone to your local machine, add files, commit, push, and pull from your GitHub repo.
3. If you’re missing any of these prerequisites, you can brush up using the resources below. You \_don’t\_ need to read/review any of these as part of the assignment; they’re resources to help if you need it.
   1. Download git: <https://git-scm.com/downloads>
   2. Git book: (through command prompt) <https://git-scm.com/book/en/v2>
   3. Git on Pluralsight: <https://app.pluralsight.com/library/courses/mastering-git/table-of-contents>
   4. Tortoise: <https://tortoisegit.org/docs/tortoisegit/>
   5. GitHub Desktop: [https://help.GitHub.com/desktop/guides/contributing-to-projects/](https://help.github.com/desktop/guides/contributing-to-projects/)

To differentiate team members, I’ll refer to them by:

1. Group-member-who-created-the-first-repo
2. Team Member #2
3. Team Member #3
4. Team Member #4 (tasks may be completed by Team Member #2 if you only have three team members)

**For all questions and steps below, explain how you verified the functionality. Include screen shots of the git log, the compare tool, conflict resolution tool, or IDE/other tools to illustrate the what happened and document git’s ability to meet the need in question.**

# Priority 1: Versioning, Save my stuff.

All class sections identified saving and versioning as the most important needs. We need to verify git meets these needs both locally and remotely. You’ll probably learn some features and capabilities about git along the way, but remember we’re evaluating git as a tool, not just trying to get it to work.

## Single user, local machine:

1. How do you create a local repository for your own use? (No GitHub or network required or allowed)
2. Let’s measure what must happen to avoid “losing our stuff”
   1. What steps does it take to get a file protected and versioning started?
   2. Even with versioning enabled and a file tracked by git, there are still ways for you to lose your current work. How? Need more hints? See the tiny text: Does git help me if I haven’t saved from within the application like Notepad++? What about after I’ve saved within Notepad++? What must still happen before the newest changes are versioned in git?
3. Demonstrate that git has non-destructive versioning (reverting changes from version “3” creates version “4” rather than deleting version “3”)
4. What data, when deleted, causes all the history and versions to be lost, despite using git? (Think destructively. More hints in tiny text: Where does git store your data? Try deleting a subfolder within a git-tracked folder. Can you revert? Now try deleting the parent directory. Can you revert?)
5. What steps can you take without network or internet connectivity to protect your git repo from a hard drive failure or a catastrophic event like the one you uncovered in the test above?
6. Evaluate how git responds to moving a tracked file out of the top level git directory (like, for example, from c:\mygitrepo\file.txt to c:\users\me\desktop\file.txt. Is such a file still protected?

## Remote (using GitHub)

1. Group-member-who-created-the-first-repo, create a private GitHub repo and add as collaborators all group members **and the instructor** (GitHub username: mattwarnerneumont). Use this repository throughout this section.

DONE

1. Describe what it takes for all team members to each create a file and have all other team members receive its most current version. It’s probably a good idea to make a few edits and commits here just to prove everyone can use and benefit from this feature over multiple commits, not just once.
   * Each member must open where they saved the repository that they cloned,
   * Then they have to create a file in that folder (This creates a new file)
   * Once one person makes a file they must push it, this will bring it to the repository
   * Others can pull from the repository and they will see the file that was created
   * Once they pull they can edit the files
   * Once they edit the files they will have to commit their changes
   * After commiting the change they must push it to the repository so that everyone can see the changes they made
2. Let’s evaluate catastrophic loss conditions again. Impending disaster awaits!
   1. Team Member #2: Star the repo on GitHub.
   2. Team Member #3: Fork the repo on GitHub.
   3. Team Member #4: Export the repo (don’t clone) from GitHub.
   4. Group-member-who-created-the-first-repo, you’ve become disgruntled and decide to show your team how mad you are by completely deleting the repo on GitHub ([I am not making this up](http://www.businessinsider.com/npm-left-pad-controversy-explained-2016-3)). Go for it, and when GitHub warns you that the deletion is permanent, continue with reckless abandon!
   5. Other group members: Each determine your level of data loss.
   6. Other group members: If possible, what does it take for one of you to get the repo back online as a private repo with the same collaborators (excluding the user who deleted the repo)?
   7. Differentiate “star”ing, forking, and exporting in GitHub.
3. Summarize what must happen to cause *irreversible and total repository loss* when using GitHub and a team of 4 developers who each have the most recent pull from the repo?

# Priority 2: Team Collaboration

## Using GitHub

1. The previously angry group member (looking at you Group-member-who-created-the-first-repo) is trying one last attempt to destroy the code. Look over your teammate’s shoulder to discover the URL to the private repository they’ve excluded you from and try to access it. What is the result?
2. Having given up trying to destroy a distributed repository (which is one reason [why Bitcoin is hard to hack](https://bitnodes.earn.com/)) you decide to rejoin the team and humbly ask to be added as collaborator on the new private repo. Other team members, oblige the request! What happens now, penitent team member, when you hit the new repo URL?
3. Here is the team collaboration scenario: we are going to work on two important but separate features on a production application. Demonstrate that you can have different and separate development efforts on your team without the efforts directly breaking or impacting other team members.
   1. Create two new branches in git named feature1 and feature2.
   2. Show that *each* team member can commit to *each* feature branch without affecting the master branch. To avoid evaluating branching/merging *and* conflict resolution, name all the files you work on in each branch respectively with an f<featurenumber> prefix, like f1whatever.txt or f2whatever.txt. They can be in the same folder.
   3. Make one change to the master branch and show this change does not appear on either of the feature branches.
   4. Note: (nothing to answer or do here, just FYI) This is how production teams work on new features while retaining the ability to quickly make a fix to production code and release it without also releasing incomplete/development features.
   5. Merge current master branch into the feature1 branch (this is different from merging the feature1 branch back into master). This pulls the most recent change from master into feature1 so that branch has the benefit of the new code.
   6. Feature2 is ready to ship. Merge the feature2 branch into master (not master into feature2)
   7. Make another change on the master branch. Show that this change does not affect the feature1 branch.
   8. Feature1 is ready to ship. Merge the feature1 branch into master. Show that all the changes from feature1, feature2 and the edits made to master are all present in the most recent version.
4. Describe when you should and should not use branching and merging in a team project.
5. When might you use branching and merging on an individual project?

# Achievement Unlocked: Top 2 Done!

1. We’ve just pounded git with a fair amount of testing, poking, and prodding, but we’ve only targeted the first two priority needs. Review *all* the priorities from each class section and create two lists below: one for the needs we ended up evaluating just through testing the other capabilities, and one for those that remain untested or undertested.
2. Describe how git meets your needs for the top two priorities we evaluated.
3. What functionality did you discover about git that does not meet your expectations?
4. Make a go/no go decision for git as your source control system based on your needs, and briefly justify your decision.
5. How did you facilitate team collaboration on \_this\_ document? Would git and github be a good solution? Why or why not? If you used a different collaboration/save/versioning solution, what was it and why?