|  |
| --- |
| Catalyst it services |
| Git |
| Step-by-Step |
|  |
| **Katherine Root**  **Holly Zelhart**  **Version 1.3** |
| **10/25/2013** |

Contents

[Git Terminology 2](#_Toc370460025)

[Git Workflow Diagram 4](#_Toc370460026)

[Git Workflow 7](#_Toc370460027)

[How to Create a Working Branch from the Master Branch 8](#_Toc370460028)

[How to Put the Changes in your Workspace into the Remote Repository 9](#_Toc370460029)

[How to Clean Up Your Workspace (three options): 10](#_Toc370460030)

[How to Revert your Working Branch 11](#_Toc370460031)

[How to make your Working Branch current: 12](#_Toc370460032)

[Save Modified Files without Pushing Them into the Remote Repository 13](#_Toc370460033)

[How to Push Changes from your Workspace into the Remote Repository 15](#_Toc370460034)

[How to Add Your Changes to the Master Branch 17](#_Toc370460035)

[How to Add a New or Modified File Directly into your Remote Repository: 18](#_Toc370460036)

[How to Delete a Branch 19](#_Toc370460037)

[If you are not sure that you want to delete a branch from the remote repository: 20](#_Toc370460038)

[Troubleshooting 22](#_Toc370460039)

[Quick Reference 24](#_Toc370460040)

[Git Commands 24](#_Toc370460041)

[Git Workflow Diagram 25](#_Toc370460042)

[Works Cited 26](#_Toc370460043)

# Git Terminology

**add –** Changes that were made to the current branch are put into the index (Akiri Solutions, 2012).

**checkout –** A branch is put into workspace so that it can be viewed and/or edited. (Ex: if you checkout Sprint3, when you go to Eclipse you will see and can edit the Sprint3 branch version of the project. If you then checkout Sprint1\_BBP-29\_Login branch, when you return to Eclipse and refresh the project you will now see and be able to edit the Sprint1\_BBP-29\_Login branch version of the project.

**commit –** Adds the changes made to the branch in your workspace to the corresponding branch in your local repository.

**fetch** - Fetching is what you do when you want to see what everybody else has been working on. Fetching has no effect on your local development work. This makes fetching a safe way to review commits before integrating them with your local repository. A git fetch lets you see how the central history has progressed, but it doesn’t force you to actually merge the changes into your repository (Atlassian).

**Git -** The purpose of Git is to manage a project, or a set of files, as they change over time. Git stores this information in a data structure called a repository (Duan, 2013).

**index (a.k.a. “staging area”)** - Where you place files you want committed to the git repository. Before you “commit” files to the git repository, you need to first place the files in the git “index” (Akiri Solutions, 2012).

**Local Repository -** A repository can be thought of as container where a collection of files and their complete history are stored. The Local Repository is the container that is on your computer. It is likely that you are the only one who can view or access the branches in your Local Repository.

**Master Branch –** The Master Branch is the project’s central origin. All developers working on the project make their Working Branches off of this Master Branch. When work is completed on the segment of work, the Working Branch is then merged back into the Master Branch.

**merge –** Join two or more development histories together. If you changed the same part of the same file differently in the two branches you’re merging together, Git won’t be able to merge them cleanly, and a “merge conflict” will occur (Linux, 2013).

**pull –** Gathers the changes from a branch in the remote repository and incorporates them into the corresponding branch in your local repository and your workspace. In other words, a git pull synchronizes your local repository with upstream changes. A git pull does the same thing that a git fetch followed by git merge does in a single command. A git pull can be thought of as Git's version of an SVN update.

**push** – Publishes the changes made to a local branch up to that branch in the remote repository (Atlassian).

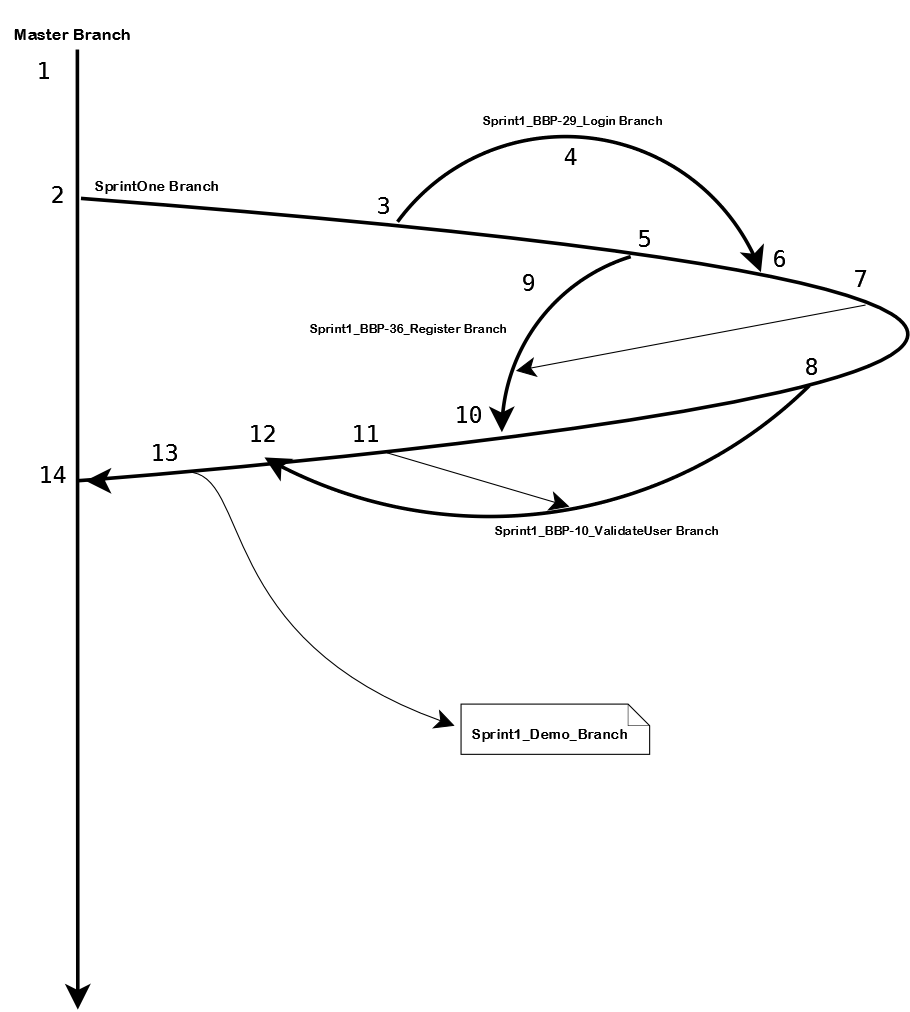
**Remote Repository –** A repository can be thought of as container where a collection of files and their complete history are stored. The Remote Repository is the container that is most likely somewhere other than your computer, perhaps on a server in a different room or building. Your branches that are in your Remote Repository can be viewed and accessed by everyone.

**staging area -** (see “index”)

**status** (command: “git status”) - Shows the differences between the working directory, the index, and the most recent commit (Akiri Solutions, 2012).

**Working Branch** – Before beginning work on a user story, a developer will make a copy of the Master Branch. This copy is called the “Working Branch”. All work for the user story is done to this copy. After completing the user story, the developer will merge the Working Branch back into the Master Branch and then delete the Working Branch.

**Workspace** - Where you do your work: You add, remove, and modify files in your workspace (Akiri Solutions, 2012).

Git Workflow Diagram

1. The Master Branch is created.
2. A branch called “SprintOne” is created off of the Master Branch.
3. A Working Branch called Sprint1\_BBP-29\_Login[[1]](#footnote-1) is created off of the SprintOne Branch. This is so the work necessary to complete user story BBP-29 can be done away from the SprintOne Branch.
4. The work for BBP-29 is completed. This means that files might be created, updated, and/or deleted on the Sprint1\_BBP-29\_Login branch. This may take hours, days, or weeks.
5. In the meantime, a second Working Branch called Sprint1\_BBP-36\_Register is created off of the SprintOne Branch for the purpose of accomplishing user story BBP-36.
6. Because the work on BBP-29 is completed, Working Branch Sprint1\_BBP-29\_Login needs to be pushed back into the SprintOne branch. This means that all of the work that was done on the Sprint1\_BBP-29\_Login branch will be added to the SprintOne branch. (This action can be thought of as doing an SVN commit).
7. In order to make the Sprint1\_BBP-36\_Register branch current with the changes that were made to the SprintOne branch in step 6, the branch will pull from the SprintOne branch. (This action can be thought of as updating-to-head in SVN).
8. A third Working Branch called Sprint1\_BBP-10\_ValidateUser is created off of the SprintOne Branch for the purpose of accomplishing user story BBP-10.
9. In the meantime, the work for BBP-36 is completed.
10. Because the work on BBP-36 is completed, Working Branch Sprint1\_BBP-36\_ Register needs to be pushed back into the SprintOne branch. This means that all of the work that was done on the Sprint1\_BBP-36\_ Register branch will be added to the SprintOne branch. (Again, this action can be thought of as doing an SVN commit).
11. In order to make the Sprint1\_BBP-10\_ValidateUser branch current with the changes that were made to the SprintOne branch in step 10, the branch will pull from the SprintOne branch. (This action can be thought of as updating-to-head in SVN).
12. After the work on BBP-36 is completed, Working Branch Sprint1\_BBP-10\_ValidateUser needs to be pushed back into the SprintOne branch. This means that all of the work that was done on the Sprint1\_BBP-10\_ValidateUser branch will be added to the SprintOne branch. (Again, this action can be thought of as doing an SVN commit).
13. At a prearranged time before the demo (for instance, 12:00 noon the day before), a branch called Sprint1\_Demo\_Branch will be created off of the SprintOne Branch. This is the branch that has all of the code and tests that have been prepared for the demo, and will be kept isolated from the workflow. It will not be updated with any changes after it is created. The demo branch is used to “freeze” the code while allowing production to continue. This step could be done at any point between step 2 and step 14, as long as at that point the SprintOne branch reflects the project as it is to be demoed.
14. Work on Sprint One is completed and all working branches have been pushed back into the Sprint One branch. Now the Sprint One branch is pushed back into the Master Branch.

At this point, steps 2 through 14 can be repeated for the next Sprint. For example, a new branch called SprintTwo will be created off of the Master Branch. One or more Working Branches will be created off of SprintTwo and as work is completed on them they will be pushed back into the SprintTwo branch. The Working Branches will periodically pull changes from the Sprint Two branch. A Demo Branch will be created and used to demo the work completed on the sprint. Sprint Two is pushed back into the Master Branch, and the cycle can begin again.

Git Workflow**:**

Each time a new user story is started (or a meaningful segment of changes are going to be made to the existing project), these three steps will be performed:

1. **Make your own Working Branch off of the Sprint Branch.** The Sprint Branch is the branch that everyone else is branching off of, for example “SprintTwo”. (See “***How to Make a Working Branch from a Master Branch***”)
2. **Merge changes that others have “pushed” to the Sprint branch with your Working Branch.** While you are making your changes to the code in your Working branch, others will make changes to the code in their own Working Branches, and will “push” these changes to the Sprint Branch. You will want to periodically “pull” these changes that the others have “pushed” to the Master branch with your Working branch so that the Working Branch in your workspace is as up to date as possible. \*Remember, you can only merge changes when you have a “clean” workspace (i.e. you have no changes to your workspace that have not been “added” or “committed” and “pushed”). See “***How to Make your Workspace Clean***”.
3. **Merge your changed code with the Sprint Branch.** You have completed a user story and the new/changed code is ready to be added to the Master Branch. (See “***How to Add Your Changes to the Master Branch****”*.

# How to Create a Working Branch from the Master Branch

The Master Branch is the branch that everyone else is branching off of, for example “SprintTwo”. You need to make your own copy of this Master Branch that you will work on in your own Workspace. (Later, after the work has been completed and tested, you will merge this Working Branch back into the Master Branch).

1. Checkout the Master Branch that you want to branch off of. You will checkout the Master Branch that is in your Local Repository and put it into your Workspace:  
     
   *git checkout <Master Branch>*ex: *git checkout SprintTwo*
2. Make sure that your Master Branch in the Local Repository is completely up to date by pulling the most up to date copy down from your Remote Repository into your Local Repository, and then down into the checked out copy in your Workspace:  
     
   *git pull origin <Master Branch>*ex: *git pull origin sprintTwo*
3. Make an exact copy of this Master Branch in the Local Repository, and then check it out into your Workspace. This copy will be your Working Branch. The naming convention for branches for completing user stories is sprint#\_userStory\_shortDescription (ex:  *sprint2\_BBP-19\_NewEndpoint* ). For branches created for writing unit tests after the userStory branch has been completed and remerged (i.e. writing unit tests retroactively) is sprint#\_unitTest\_class (ex: sprint2\_unitTest\_userEndpoint). If the unit tests will be for more than one class, separate the classes with a hyphen (ex: sprint2\_unitTest\_userEndpoint-bookEndpoint) :  
     
   *git checkout –b <*Working Branch*>*ex: *git checkout –b sprint2\_BBP-19\_NewEndpoint*
4. Push the new branch into the Remote Repository so that everybody can see it:  
     
   *git push origin <*Working Branch*>*  
   ex: git push origin *sprint2\_BBP-19\_NewEndpoint*
5. Make sure that the New Local Branch made it up to the Remote Repository by listing all the branches that exist off of the Master Branch:  
     
   *git branch*  –*a*Your new branch should appear in the list in green, and you should also see it in the list of remote branches in red font.  
   Red = Branches in the Remote Repository  
   White = Branches in the Local Repository  
   Green = The local branch that is currently checked out into your Workspace

# How to Put the Changes in your Workspace into the Remote Repository

After you have made changes to code in your Working Branch, you should put these changes into your Working branch in your Remote Repository so that everybody can see and use them, and so you have a backup if something goes wrong with your environment. This step can be done as often as you like without a risk of affecting the Master Branch with your changes.  
  
The following instructions assume that you have your Working Branch checked out into your workspace. They also assume that you are the only person who makes modifications to your Working Branch.

1. View the files in your workspace that need to be staged. They will be listed in red:  
     
   *git status*
2. Stage all modifications to the project (including deletions of and classes or files) in the Index:  
     
   *git add* – –*all*
3. View the files that now need to be committed. They will be listed in green:  
     
   *git status*
4. Push all staged changes into the Local Repository:  
     
   *git commit –m “a commit comment”*ex: *git commit –m “Fixed the bug that made the totals be doubled”*
5. Verify that there is nothing to commit and that your working directory is clean:  
     
   *git status*
6. Push changes up to your Working Branch in your Remote Repository:  
     
   *git push origin <Working Branch>*  
   ex: git push origin *sprint2\_BBP-19\_NewEndpoint*

# How to Clean Up Your Workspace (three options):

1. You have made changes to code in your Working Branch and you want these changes to be put into your Working Branch in the Remote Repository. (See “***How to Put the Changes in your Workspace into the Remote Repository***”).
2. You have made changes to code in your Working Branch in your workspace, but you want these changes to be destroyed (i.e. you want to revert your Working Branch back to the way it was right before the last time you “pulled” your Working Branch in your Local Repository or in your Remote Repository.) (See “***How to Revert your Working Branch***”).
3. You have experimental or broken code that you don’t want to push up to the Working Branch in the Remote Repository yet, and you do not want these changes destroyed. Before you merge changes that others have pushed to the Master branch with your Working branch, you must first “clean up your workspace” of this changed code that you want to save but not yet push. (See “***Save Modified Files without Pushing Them into the Remote Repository***”).

# How to Revert your Working Branch

If you want to destroy the changes that you made to your Working Branch in your Workspace and start with a fresh copy, you can revert your Working Branch in your Workspace to the copy of your Working Branch in your Local Repository or your Remote Repository, whichever is the closest copy (Closest copy means that if your Working Branch isn’t available to check out in the Local Repository, your Working Branch in the Remote Repository will be checked out instead.)

1. Check out a different branch into your Workspace, and in the process destroy the changes that you made to your Working Branch in your Workspace. The “different branch” can be any branch, as long as it is not your Working Branch:  
     
   *git checkout –f <Different Branch>*ex: *git checkout –f Master*
2. Check your Working Branch back out:  
     
   *git checkout <Working Branch>*  
   ex: *git checkout* *sprint2\_BBP-19\_NewEndpoint*
3. Make sure that your Workspace is now “clean”:  
     
   *git status*  
     
   \*There should be no files listed in red or green. If there are, then your Workspace is not clean, and you will not be able to merge. See “***How to Make Your Workspace Clean***”.

How to make your Working Branch current:Throughout the sprint, others will be pushing their Working Branches back into the Master Branch in the Remote Repository. You should frequently incorporate the latest changes that were made to the Master with your Working Branch. (In SVN, this is “Updating to Head”). Remember, in order to merge the Master Branch with your Working Branch, your workspace must be clean (i.e. you have no changes that need to be committed).

From your Working Branch, make sure that your workspace is clean:  
  
*git status*If there are files listed, your workspace is not clean. See “***How to Clean Up Your Workspace***”.

If your workspace is already clean:

1. Checkout the Master Branch:  
     
   *git checkout <Master Branch>*ex: *git checkout sprintTwo*
2. Make sure that you have the updated versions of the Master Branch (and the Working Branch if other people are making changes to your Working Branch in the Remote Repository – this is probably not the case on a bench project here at Catalyst. Each person tends to have their own Working Branch). Update the Master Branch in the Local Repository with the changes that have been made to the Master Branch in the Remote Repository:  
     
   *git pull origin <Master Branch>*ex: *git pull origin sprintTwo*
3. Checkout the Working Branch  
     
   *git checkout <Working Branch>*ex: *git checkout sprint2\_BBP-19\_NewEndpoint*
4. Merge the changes in the Master Branch with your Working Branch:  
     
   *git merge <Master Branch>*ex: *git merge sprintTwo*If you didn’t have conflicts, you can proceed to step 5. If you did have conflicts, you must resolve them before proceeding.
5. Push your updated Working Branch up to the Remote Repository:  
     
   *git push origin <Working Branch>  
   ex: git push origin sprint2\_BBP-19\_NewEndpoint*

## Save Modified Files without Pushing Them into the Remote Repository

You want to merge changes that others have pushed to the Master branch with your Working branch, but you have experimental or broken code in your Workspace that you don’t want to push up to the Working Branch in the Remote Repository yet. In order to merge the Master Branch with your Working Branch, you must first “clean up” your Workspace of this changed code that you want to save but not yet push.

1. Checkout your Working Branch from your Local Repository:  
     
   *git checkout <Working Branch>*ex: *git checkout* *sprint2\_BBP-19\_NewEndpoint*
2. Request a list of the files in your Workspace that you have modified but not added or committed to the Local Repository, or pushed to the Remote Repository:  
     
   *git status*These are the files that need to be “staged” before you can do a “merge”.
3. Stage all of modified files that are in your Workspace:  
     
   *git add* – –*all*This puts all of the modified files that are in your Workspace into the “staging area” (a.k.a. “Index”), and leaves your Workspace “clean”.
4. Make sure that the modified files are successfully staged. They will be listed in green:

*git status*

1. Now you can check out the Master Branch without interfering with your changed files:  
     
   *git checkout <Master Branch>*ex: *git checkout sprintTwo*
2. Update this checked out copy of Master Branch with all of the changes that are in the Remote Repository:  
     
   *git pull origin <Master Branch>*ex: *git pull origin sprintTwo*
3. Check out the Working Branch again so that it is your checked out Branch.  
     
   *git checkout <Working Branch>*ex: *git checkout* *sprint2\_BBP-19\_NewEndpoint*
4. Now merge the changed Master Branch that is in the Remote Repository with your Working Branch in your Local Repository.  
     
   *git merge <Master Branch>*ex: *git merge sprintTwo*
5. Now un-stage those changes that you staged in step 3 and add them back into your Working Branch in your Workspace:  
     
   *git reset head*

# How to Push Changes from your Workspace into the Remote Repository

You want to merge the changes that others have made to the Master branch with your Working Branch, but you have to first make your workspace clean by pushing your modified Working Branch up to the Remote Repository.

1. From your Working Branch, request a list of the files in your Workspace that you have modified but not added or committed to the Local Repository, or pushed to the Remote Repository:  
     
   *git status*These are the files that need to be “staged” before you can do a “merge”.
2. Add the file(s) to your index:  
   *git add* – –*all*
3. *git status*You should see your file in green. This means it’s been added to the index.
4. Commit the file to your local repository:  
   *git commit –m “some commit message”*
5. *git status*You will see your file that needs to be pushed. Your branch will be ahead by x number of commits.
6. Push the file into the remote repository:  
   *git push origin sprint2\_BBP-19\_NewEndpoint*
7. *git status*  
   You should see no red or green files, and see that there is nothing to commit, and the working directory is clean.

Now you can merge the updated Master Branch with your Working Branch without interfering with your changed files:

1. Checkout the Master Branch:  
     
   *git checkout <Master Branch>*ex: *git checkout sprintTwo*
2. Update this checked out copy of Master Branch with all of the changes that are in the Remote Repository:  
     
   *git pull origin <Master Branch>*ex: *git pull origin sprintTwo*
3. Check out your Working Branch again:  
     
   *git checkout <Working Branch>*ex: *git checkout* *sprint2\_BBP-19\_NewEndpoint*
4. Now merge the changed Master Branch that is in the Remote Repository with your Working Branch in your Local Repository.  
     
   *git merge <Master Branch>*ex: *git merge sprintTwo*
5. Test that the code works as expected.
6. Push the merged copy of your Working Branch in your Workspace up to your Working Branch in the Remote Repository:  
     
   git push origin *sprint2\_BBP-19\_NewEndpoint*

If work is completed on this Working Branch, it can now be pushed back into the Master Branch (see ***How to Add Your Changes to the Master Branch***).

# How to Add Your Changes to the Master Branch

You have completed a user story and your Working Branch is now ready to be merged back into the Master Branch in the Remote Repository.

1. Make sure that your Workspace is “clean” (i.e. there has to be “no changes to add/commit” and the Workspace has to be “up to date”):  
     
   *git status*If your workspace is not clean, see “***How to Make your Workspace Clean***”.
2. Check out the Master Branch into your Workspace:  
     
   *git checkout <Master Branch>*ex: *git checkout sprintTwo*
3. Make sure that the Master Branch in your Local Repository and in your Workspace are updated with all of the changes that were made to the Master Branch in the Remote Repository:  
     
   *git pull origin <Master Branch>  
   ex: git pull origin sprintTwo*
4. Add the changes that were made to the Working Branch in your Local Repository to the Master Branch in your Local Repository:  
     
   *git merge <Working Branch>  
   ex: git merge sprint2\_BBP-19\_NewEndpoint*
5. Verify that this merged code in the Master Branch functions properly and as expected, that the unit test still pass, etc.
6. Push this now updated Master Branch in your Local Repository up to the Remote Repository:  
     
   *git push origin <Master Branch>*ex: *git push origin sprintTwo*
7. Verify that the Master Branch in the Remote Repository reflects the changes that you added, and you have the code is functioning as it should (functional and unit tests work, war file is deployable, anything else that proves that the Master Branch absorbed the changes properly).
8. Now that the Master Branch has all of the changes that you made with your Working Branch, you should delete your working branch so that the remote repository is kept tidy. (See “How to Delete a Branch”).

How to Add a New or Modified File Directly into your Remote Repository:You can do this if you’ve added a file or made minor and inconsequential changes to a file, and the changes will not affect the functioning of the project in any meaningful way (for example, you’ve added or modified a Word file in the documentation folder.

1. Get the latest copy of the branch from the remote:  
     
   *git pull origin <Branch Name>*ex: *git pull origin sprintTwo*
2. *git status*You should see your file in red.
3. Add the file to your index:  
     
   *git add* – –*all*
4. *git status*You should see your file in green. This means it has been added to the index.
5. Commit the file to your local repository:  
     
   *git commit –m “some commit message”*
6. *git status*You will see that your branch is ahead by x number of commits. There will be nothing to commit. The working directory will be clean.
7. Push the file into the remote repository:  
     
   *git push origin sprintTwo*
8. *git status*  
   You should see no red or green files, and see that there is nothing to commit, and the working directory is clean.

How to Delete a BranchThe Remote Repository should be kept tidy. This means that after you have completed work on a your Working Branch, have merged the branch back into the Master Branch in the Remote Repository, and have tested that the merge didn’t break anything and all is well, you should now delete that Working Branch from both your Local Repository and the Remote Repository. All of the branches in the Remote Repository should either be the Master Branch, a SprintDemo Branch, or be branches that people are currently working on. Remember, you should delete a branch locally before you delete it remotely. Be careful. Deleting a branch both remotely and locally means that that branch is gone forever to everyone. If the Working Branch was not first merged into the Master Branch, then the work that was done on the Working Branch will be gone forever. All local branches that aren’t also in the remote repository should be deleted unless you have a good reason for them to exist (i.e. the branch is a special experiment that will be deleted at a later time.)

**If you are sure that you want to delete a branch from your local and remote repository:**

1. Make sure that your list of local and remote branches is current:  
     
   *git remote update –p*
2. View all branches. The branches in white are in your local repository. The branches in red are in your remote repository:  
     
   *git branch –a*
3. Delete the branch in the Local Repository:  
     
   *git branch –D Branch\_to\_be\_Deleted*ex: *git branch –D* *sprintTwo\_BBP-234\_ExperimentalBranchToBeDeleted*
4. Make sure that the branch is gone from your Local Repository (it should no longer be listed in white):  
     
   *git branch –a*
5. Now you can delete the branch out of the Remote Repository. Don’t forget to type the colon:  
     
   *git push origin* :*Branch\_to\_be\_Deleted*ex: git push origin *:* *sprintTwo\_BBP-234\_ExperimentalBranchToBeDeleted*This command pushes the lack of Branch in your Local Repository up to your Remote Repository, in effect deleting it from the Remote Repository. Notice that there is no space after the colon.
6. Make sure that the branch was deleted from the Remote Repository (it should no longer be listed in red:  
     
   *git branch –a*

## If you are not sure that you want to delete a branch from the remote repository:

You’re not sure if the Working Branch that you are considering deleting from the remote repository has been merged with the Master Branch in the remote repository. To make sure that it has been merged before you delete it:

1. Make sure that your list of local and remote branches is current:  
     
   *git remote update –p*
2. View all branches. The branches in white are in your local repository. The branches in red are in your remote repository:  
     
   *git branch –a*
3. Checkout the branch that may or may not have been merged:  
     
   *git checkout <Working Branch>*ex: *git checkout sprintTwo\_BBP-234\_ExperimentalBranchToMaybeBeDeleted*
4. View the list of commits that were made to the Working Branch and the list of files that were affected by each of the commits:  
     
   git log – –stat  
     
   Write down the first few letters of the two most recent commits, and write down the corresponding comments. You can also write down the dates of the commits to help you locate them later.
5. Checkout the Master Branch:  
     
   *git checkout <Master Branch>*ex: *git checkout <sprintTwo>*
6. Make sure that you have the most up-to-date version of the branch:  
     
   *git pull origin <Master Branch>*ex: *git pull origin sprintTwo*
7. View the list of commits that were made to the Master Branch and the list of files that were affected by each of the commits:  
     
   git log – –stat  
     
   The commits and comments that you wrote down in step four should be present in the list of commits to the Master Branch. If they are listed, then the Working Branch in question has been merged with the Master Branch, and it can be safely deleted from the Remote Repository (you can now follow the “**If you are sure that you want to delete a branch from your local and remote repository…”** instructions above).   
     
   If they are not listed, then this Working Branch has not been merged with the Master Branch recently, and you should consult with the person who is working on that branch as to whether they want to merge those changes into the master or abandon them. Don’t delete the Working Branch until you check or the changes could be lost forever!

# Troubleshooting

Ooops! You started making changes to the code in the Master Branch instead of your Working Branch. You don’t want to lose your changes, but you want them to be in your Working Branch instead. This might have happened if you had the Master Branch checked out and started making changes to it before you noticed your mistake. How to fix it:

1. Commit your changes to your Master Branch Local Repository  
   1. You should see your modified files in your workspace when you git status:  
        
      *git status*
   2. Add all of your modified files to the index (staging area):  
        
      *git add –all*
   3. Commit your added files to your Local Repository:  
        
      *git commit –m “commit message here”*
2. Create a Working Branch off of your Local Master Repository and check it out into your Workspace.  
   (See ***How to Create a Working Branch from the Master Branch***).
3. Merge your new Working Branch with the Master Branch in your Local Repository.  
     
   *git merge <Master Branch>*ex: *git merge sprintTwo*
4. Make sure that the work that you did on the Master Branch is now in your new Working Branch. The first commit listed will be the one that you created in step 1-c:  
     
   *git log* – –*stat*
5. *Push your Working Branch to the Remote Repository:  
     
    git push origin <Working Branch>*

ex: *git push origin sprint2\_bbp-9\_AddingEndpoint*

1. Delete the Master Branch in your Local Repository. You want to do this so that the work that did to the Master Branch in your Local Repository does not get pushed up to the Master Branch in the Remote Repository. Instead you have already pushed the work that you did up to the Working Branch in the Remote Repository in step 5.

*git branch –D <Master Branch>*

ex: *git branch –D sprintTwo*

Note: “git branch –D” only deletes the branch in your Local Repository and does not affect the one in the Remote Repository. Now when you check out sprintTwo, it will be the clean remote version.

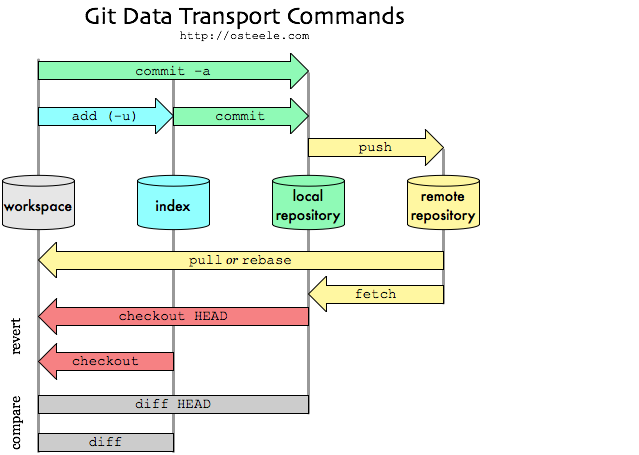
# Quick Reference

Git Commands**:**

|  |  |
| --- | --- |
| git add – –all | Stages all modifications to the project in the index |
| git branch –a | View all branches |
| git branch –D *branch\_Name* | Delete branch from local repository |
| git checkout *branch\_Name* | Checks the branch out into your Workspace |
| git commit –m “*comment*” | Pushes all staged changes into the Local Repository |
| git fetch – –all | Updates your Remote Repository with the current branches |
| git log – –stat | View the list of commits made to the current branch and the list of files that were affected by each of the commits |
| git merge *branch\_Name* | Merge the changes in *branch\_Name* with the currently checked out branch |
| git pull origin *branch\_Name* | Update the branch in the Local Repository with the changes to the branch in the Remote Repository |
| git push origin *branch\_Name* | Pushes the branch into the remote repository |
| git remote update –p | Makes your list of local and remote branches current |
| git reset head | Unstages changes and adds them back into the Working Branch in your Workspace |
| git status | Shows the differences between the working directory, the index, and the most recent commit |

# Git Workflow Diagram

(Tiebing, 2010)



# Works Cited

Akiri Solutions. (2012). *What’s The Deal With The Git Index?* Retrieved October 24, 2013, from Git Guys: http://www.gitguys.com/topics/whats-the-deal-with-the-git-index/

Atlassian. (n.d.). *Git Tutorials*. Retrieved October 24, 2013, from Remote Git Repositories: https://www.atlassian.com/git/tutorial/remote-repositories#!push

Duan, C. (2013). *Understanding Git: Repositories*. Retrieved October 17, 2013, from http://www.sbf5.com/~cduan/technical/git/git-1.shtml

Linux. (2013, February 2). *git-merge(1) Manual Page* . Retrieved October 24, 2013, from The Linux Kernel Archives: https://www.kernel.org/pub/software/scm/git/docs/git-merge.html

Tiebing. (2010, December 12). *Git Data Transport Commands*. Retrieved October 3, 2013, from Journey of Life: http://tiebing.blogspot.com/2010/12/git-data-transport-commands.html

1. The naming convention for branches that complete user stories is sprint#\_userStory\_shortDescription (ex: sprint2\_BBP-19\_NewEndpoint. [↑](#footnote-ref-1)