GIT Setup and Usage

Version 7, September 1, 2012

Original Author: Giri Alwar

NOTE: This document assumes that you have a basic understanding of Git. If not, please read the first 3 chapters of the Git book available online here: <http://git-scm.com/book> before following the instructions below.

One piece of information is worth mentioning. Any branch available in the remote repository (for instance, “master”) has 3 instances:

* “master” branch in your *local* repository: This is a local branch that is like any other branch you create in your local repository. The code can be modified in this branch, committed and pushed to remote. Note however that as part of development process outlined in this document, you will never directly develop in the “master” branch and will instead work on a new branch.
* “origin/master” in your *local* repository: This is a reference that exists in your *local* repository that points to where the latest (HEAD) of “origin/master” was in the *remote* repository when a sync was performed with the remote repository. Note that “origin/master” in the remote repository can be newer than the reference in your local repository if other developers made changes to it since your last sync.
* “origin/master” in the *remote* repository: Indicates where the latest (HEAD) for the master branch is in the remote repository. This gets updated when someone do a “push” of their local master branch.

**GIT SETUP (ONE-TIME ONLY):**

1. (For Windows Users) Install Git from here: <http://code.google.com/p/msysgit/downloads/detail?name=Git-1.7.11-preview20120710.exe&can=2&q=full+installer+official+git>

(For Mac Users) It is possible you already have Git installed on your machine. Open a shell and type in “git –version” to see if you have Git installed. If you don’t have Git installed, install it from here: <http://git-scm.com/download/mac>

1. Open Git Bash (on Windows) or its Mac equivalent.
2. Set up Git by running these two commands (provide your real name and email address in double quotes). These two commands tell Git the name and email to use to identify your commits:
   1. git config --global user.name "*Your Real Name Here*"
   2. git config --global user.email "*Your Email Address Here*"

Verify your newly created Git config settings by typing the following command:

git config --list

**PROJECT SETUP (ONE-TIME ONLY)**

1. Change directory to the directory that you wish to use for the project (create one, if it does not exist). For instance:
   1. cd /Projects/ATT/DigitalLife/Web
2. Make a clone of the project remote repository
   1. git clone <https://gitsrv.razorfishtc.com/git/attdigitallife-websrv>
   2. Enter your credentials (for Razorfish folks, this is your “main” login credentials):

Note: This will create a sub-folder under the project directory called “attdigitallife-websrv”. This sub-folder will be the working directory for your project.

1. (Optional) Setup Git to remember your credentials for the “https” repo URL temporarily or permanently.
   1. Change directory to the newly created “attdigitallife-websrv” folder
   2. Choose ONE of the following options:

* TEMPORARY: Run the following command if you want Git to remember your credentials for this project repository temporarily for an hour:

git config credential.helper 'cache --timeout=3600'

* PERMANENT: Run the following command if you want Git to store your credentials locally and use it permanently for this project repository:

git config credential.helper store

NOTE: While permanently storing your credentials may be handy, it is important to understand that Git stores them in clear text under your “home” directory in a file called “.git-credentials”. If you don’t like this, then you should “cd” to your home directory and delete this file. After this, you may want to switch to using temporary storage option shown above.

NOTE: Either option will take effect AFTER the next time you enter your credentials.

If at any point you want Git to resume prompting you each time it needs to connect to the remote repository, enter the command “git config --unset credential.helper”

**DEVELOPMENT:**

1. Assign yourself a ticket in JIRA and mark as In Progress (click on the “Start Progress” button).
2. Open Git Bash
3. Change directory to the working directory for the project, if not already there (for instance: /Projects/ATT/DigitalLife/Web/attdigitallife-websrv). You should by default be in the “master” branch (verify by typing “git status”, if not known).

TIP: If you would like Git Bash to open in your project working directory every time, do the following:

* 1. Change directory to your “home” directory by typing in “cd”.
  2. Create a file called “.bashrc” with the following 2 lines and save it:

foo() { cd /Projects/ATT/DigitalLife/Web/attdigitallife-websrv; }

foo

Next time you open Git Bash, it should open in your project working directory.

1. Update your local repository remote references and your local “master” branch code with the latest from the remote repository
   1. git pull

Note: This command is a shortcut for two commands – to update your local repository remote references (index) with the latest from the remote repository (git fetch) and to merge the latest code from the remote “master” branch (also known as “origin/master”) into your local/current “master” branch (git merge)

1. Verify that your master branch does not have any unintended changes you made locally:
   1. git diff origin/master

This should return the prompt with no files changed. That tells you that your local “master” branch is the same as “origin/master” as the starting point for your branch. If you see changes in master, then you will need to revert your master to match origin/master using “git reset --hard origin/master” command.

1. Create a branch and switch to it
   1. git checkout -b <INITIALS>\_<JIRA\_ID>

Note: Replace <INITIALS> with your initials and <JIRA\_ID> with the real JIRA ID. This command is a shortcut for two commands

* + create a branch (git branch <INITIALS>\_<JIRA\_ID>) and
  + switch to that branch (git checkout <INITIALS>\_<JIRA\_ID>)

1. Develop feature or fix defect
2. Before submitting your work for review, update your branch with latest from remote master
   1. git pull origin master
3. If conflicts exist, resolve them manually
4. Test and verify that all is good
5. Verify what has changed in the branch
   1. git status
6. Add all changes (new files, modified files and deleted files) for commit:
   1. git add –-all .
7. Verify the files you are about to commit are what you want:
   1. git diff –b <INITIALS>\_<JIRA\_ID>

If any files need to be reverted, do this:

* git checkout -- <full\_path\_to\_filename>

1. Commit all changes to your local branch
   1. git commit –m “Commit Message”
2. Push your local branch to the remote repository for review
   1. git push origin <INITIALS>\_<JIRA\_ID>

NOTE: Your local branch will be available in the remote repository to everyone as “origin/<INITIALS>\_<JIRA\_ID>”

1. Update JIRA item as follows:
   1. Click on “Stop Progress” to indicate your work is currently done.
   2. Mark the ticket as “***Resolved****”* and add comments that development is complete - mention your Git branch name as ready for review.
   3. Assign the ticket to “***Unassigned***” so that someone else can review the code. Do not assign this to QA yet. It has not been reviewed or merged into master.
2. Post to team room that your branch is ready for review so that someone can review immediately.

**REVIEWING SOMEONE ELSE’S WORK:**

1. In JIRA, assign the ticket to yourself so that no one else is reviewing the same branch.
2. If you are in the middle of doing some work yourself on a local branch, have uncommitted changes and are not ready to commit to your branch, then you will need to temporarily stash your work:
   1. git stash
3. Switch to master branch and update both your local repository index and master branch with latest:
   1. git checkout master
   2. git pull
4. Switch to the branch that you need to review
   1. git checkout <INITIALS>\_<JIRA\_ID>
5. Merge latest master code into this branch so that you are reviewing the latest
   1. git merge origin/master

NOTE: If you get conflicts, then you should stop here, undo the merge using “git merge --abort” and ask the original developer to merge master, resolve conflicts and resubmit for review. As a reviewer you should NOT manage conflicts.

If you stashed your work before this review, get back to where you were by following these 2 steps:

* git checkout <BRANCH>, where <BRANCH> is the local branch you were working on before the review. Note your local branch only contains your last commit i.e., it does NOT contain your uncommitted/WIP code that you stashed
* git stash pop. Essentially this will apply your uncommitted/WIP code changes that you stashed earlier on top of the last commit on your local branch bringing you back to where you were before the review began. Continue development after this.

1. Review code. If review is good, proceed to next step. Otherwise, stop and ask the developer to fix code and resubmit for review. If you stashed your work before this review, follow the steps outlined under NOTE in the step above to resume.
2. Merge reviewed branch to local master
   1. Switch to master
      1. git checkout master
   2. Merge branch into master
      1. git merge -m <MESSAGE> --no-ff <INITIALS>\_<JIRA\_ID>

Note: If the JIRA ticket pertains to a Quality Center (QC) ticket in the AT&T system, please add the QC ticket number in the merge message AND the branch name you are merging (for instance, “QC 1234 – Merge ga\_123 into master”).

1. Push local master to remote master
   1. git push origin master
2. Delete remote branch
   1. git push origin :<INITIALS>\_<JIRA\_ID>

Note: It is important to specify the “:” prefix here but also be careful with this as it deletes the branch from the remote repository

1. Delete local branch
   1. git branch –d <INITIALS>\_<JIRA\_ID>
2. Add comments to JIRA ticket that you have reviewed and merge it to master. Assign ticket to QA for testing.