Getting Started with Git

Got Git? No? Get Git!

- Official Download Site: http://git-scm.com/download

Git is included with XCode 4.x

XCode 4.x installs a copy of git in /usr/bin/git

Learn About Git

- Learn the concepts.
    - Scott Chacon (of GitHub) Video Intro http://www.youtube.com/watch?v=ZDR433b0HJY
    - This is an excellent tutorial on the basic concepts behind Git
    - Git for Computer Scientists gives another good view on the underlying concepts
    - Git For Ages 4 And Up: video tutorial (mp4, YouTube)
- Learn the basics.
    - Everyday GIT With 20 Commands Or So
    - Free Online Pro Git book
        - There is even a free epub version for your iPad/iPhone! https://github.s3.amazonaws.com/media/progit.epub
- Get familiar with the reference materials available online
    - Git Community Book
    - Official Git Documentation Site
    - Official Git User Manual
    - Git Reference
- Git Cheat Sheets
    - http://cheat.errtheblog.com/s/git/
- Linus Torvalds talks about git
  Have a Good Laugh here as Linus Torvalds Evangelizes git. *NOTE: This isn't really learning, but it is way more fun and gives you insight into the motivation behind git.*

Get Started with Project Wonder!

The best way to learn a new concept is to actually use it, and what better way than to do something practical like clone the Project Wonder git source code repository!

- Downloading Wonder Wiki Page

Use Git with a Subversion Project

OK, so you want to use Git but you are working on a team project that is hosted in a subversion repository ... and it is making you depressed 😞. Well, you can still use Git to manage your local SVN working copy and be almost happy again! (Either way, you won't be 100% happy unless the repository is a git one.)

There are a couple ways of doing this. Two of them are:

- Follow these Instructions to manually set up the integration.
- Use SourceTree which can clone your SVN repository locally as a full Git repository including all historical SVN commits and their metadata.

Git Goodies for WebObjects Developers

This is a list of tools that other WebObjects developers have found to be helpful. It's not a comprehensive list, and if you find things not listed here please add them!

EGit

EGit is a plugin for Eclipse 3.6. It's not as good as Tower or GitX, but you can use it for most tasks (add, commit, push, pull).

- EGit/Git For Eclipse Users
- User Guide
- Installation
    1. In Eclipse, select *Help > Install New Sofware...*
    2. Select the "Helios" download site from the Work with: combo box
    3. Expand the "Collaboration" group
    4. Select the EGit plugin, click Next
    5. Complete the install process and agree to the license, etc.
    6. Restart Eclipse.

Git Bash Completion

This is really an absolute necessity and a **huge** productivity improvement. Basically you need the bash completion script from the source tarball and use your shell profile to include it whenever you open a shell. See the **Auto-Completion** section on this page:
Git Bash Auto-Completion

SourceTree by Atlassian

SourceTree is a free Git/Mercurial GUI for OS X. A key feature is that it can use git-svn to "Clone" a SVN repository into a standard Git repository with your full SVN commit history and maintains a link back to the SVN repository. This allows you to easily use Git locally for development but still do your final commits to SVN.

- SourceTree on AppStore
- SourceTree Home Page

GitX (L) git GUI

While most (if not all) Git GUI apps will never be able to emulate all the intricate functionality of Git, one advantage of a GUI app is more convenient and faster browsing/visualization of history and inter-relationships of branches. While development on the original open source GitX.app has waned, this fork of the original has continued to mature into a nice Git client, and is useful for common every day Git operations. However, power users (aka "cool kids") will probably only use it for history viewing while they continue to use the terminal command line for checkouts, branching, staging, rebasing and committing.
GitX (L) Home Page

Note this app replaces the original GitX by Pieter de Bie.

Open in GitX

Open In GitX Finder Droplet

Git Tower

Git Tower is a commerical app for those of you who just get panic attacks at the thought of using the Terminal.

> *I use Tower (bought it), but only for committing and fixing merge conflicts. I think Tower misses quite a bit of the flagship feature of GIT: branches. You can't see a branch tree graphically, like you can with gitX or even the command line (git log --graph --color, IIRC). But it's cool for committing. For everything else I use command line.*
> *- Miguel Arroz*

Gitolite - A Git Repository Server

Need a way to host a few dozen or hundred repositories with dead simple administration? Look no further than the free and open-source gitolite. You just need a single dedicated user account on some unix based OS. Once installed and configured, you can perform administration tasks such as adding users, adding repositories and setting fine-grained user access privileges all from your own desktop.

The best place to start is probably the ProGit chapter on gitolite, which is maintained by the author of gitolite, and is right here:
Pro Got Gitolite Chapter

Some tips/gotchas to keep in mind when setting up gitolite

- Make sure you give the gitolite user ownership of the admin public ssh key after copying the key over!
- Rename your public keyname to username.pub, where username is the name you want to use to identify yourself when configuring gitolite access control to repositories.
- During setup, gitolite creates the file ~/.ssh/authorized_keys in the gitolite user account. Ensure it has permissions of 600. If not change it!
- If other system users such as apache, chiliproject or redmine need read access to the gitolite repositories to allow integration, then you probably want to change the $REPO_MASK configuration value from 0077 to 0027 in the gitolite configuration file

Gitlab - Self hosted Git management software

From the Gitlab homepage:

GitLab is a fast, secure and stable solution to manage your projects.
It is based on Ruby on Rails and has a free and open-source license (MIT).
GitLab is the most installed git management application in the world.

Unique Git Concepts

Rebasing

Rebasing is unique to git. There is no counterpart in Subversion.

Rebasing cuts down on the spaghetti history of merging and helps to keep history nice and linear.

Conceptually, when I rebase my current branch A on another branch B, git removes all my branch A commits back to the common ancestor of A and B, stashes those commits away temporarily, moves the head of the current branch A to the tip of the other branch B, and then re-applys all my stashed commits as **new commit** patches to my branch B.

However before using it on work that is shared with others you MUST follow the these rules, otherwise you will screw everyone else on the project.

- If you don't understand rebasing, don't use it! You can learn more about it here:
  - http://progit.org/book/ch3-6.html
  - http://www.eecs.harvard.edu/~cduan/technical/git/git-5.shtml
  - http://book.git-scm.com/4_rebasing.html
  - http://blip.tv/file/4094727
- Use rebase only on **private** branches!
- Use rebase only for commits that have **not been pushed** to a remote repo

Push/Pull vs Commit/Update

It may be helpful to separate "push/pull" in your mind from "commit/update". Say that I have a bunch of changes that I want to ~~commit to Wonder~~ share with the community. I am not going to push my changes. I am going to check in my changes to my repository (either a clone or a fork) and then push from my repository or generate a pull request from my repository. My repository should be fully checked in before I do this.

Say that I do not want to check in all of the changes I am looking at. The answer seems to be that you create a branch, take the changes you want to share, commit them and then share from that branch. And, again, before one generates the pull request or does the push, the branch should be clean and all changes in it should be committed.

What if one wants to generate a pull request the origin and also push to a fork? You would do this if you were not a Wonder committer but wanted to share code. You organize your changes in your local repository/branch and, from the clean repository or branch, generate the pull request and, separately, push the changes to your fork.

Testing a Pull-Request

It may not be obvious how to do this. If one is a committer, there is a button one can push to automatically accept a pull-request. But is this really a safe thing to do? But how can you get the diffs being suggested? It seems more complicated than it needs to be to merge the branch the pull-request was made from. One can actually use a URL that specifies just the diff. If the pull-request is at https://github.com/wocommunity/wonder/pull/43, one can do:

```
curl 'https://github.com/wocommunity/wonder/pull/43.patch' | git am
```

The curl fetches the diff. Take off the "git am" to just see the diffs. The "git am" merges the diffs. Then you may build and test the pull-request. A normal push will push the code into the repository.

How Easy Are Branches?

It may be helpful to realize this. You should become comfortable with branching. In svn, branching is hard but it is not hard in git. If you have a thought, mull over that thought for a few minutes and jot something down, you have done enough to justify a branch and the branch will be not much harder than finding the post-its on your desk, and perhaps easier. Think about when you would have created a branch in svn. For that amount of work and to establish that level of separation, in git you would create a fork, or create a new repository.

Might you end up with too many branches? Perhaps. How messy is your desk? If you are a clean-desk person, you will probably not have too many branches. You will organize and combine your branches. You will rebase your branches. You may delete your branches. After all, you had the thought. You can write the code again. If it is too distracting to keep the branch relevant, tuck the thought into the back of your head, delete the branch, let it go and move on. If you are a messy-desk person, you will probably have lots of branches. It would probably be a good thing to clean up, merge and rebase your branches before you try to share your changes. But that is obvious. Nobody wants to see how messy your desk is.

Keep in mind that the difficult part of what you do should be the thinking. Git makes it easy to organize and share your thoughts as branches. Your local copies may be messy or not. They are the thoughts you have in your head and lots of those thoughts will stay there and never see the light of day and that is ok. But when you have clarified your thoughts, git will make it easy to share them.

Finding the List of All Git Commands

The "git help" command gives one the list of commonly used git commands. How does one find the others, since some of them turn out to be very useful? I tried "git help -v". That was wrong. It would have been better to try "git help help", but this did tell me about the --all flag, which actually lists out all of the commands, and there are quite a few of them. So, good hunting.

Git Everyday Tasks

How to revert?

Reverting is easy. The following command removes all modifications to files in your working copy and brings them back to the HEAD.

```
git reset --hard
```

Stash to Clean and Checkout from Stash to Work

Say that you have several different things going on in your project at any one time. How can you seperate them to check something in? You know about the "git stash" command, which cleans up your project. But what if you want part of the dirtiness, but not all of it?

Suppose you have file1.java, file2.java, file3.java, and file4.java. You have a single feature implemented in file1.java and file3.java and those edits stand on their own. If you are going to check in just file1.java and file3.java, you should test them on their own, without your other changes. First, do a "git stash". A "git status" will show you have no staged changes. Now do:

```
git checkout stash@{0} -- file1.java
git checkout stash@{0} -- file3.java
```

Now you have your two files and not the others and you can build and test them in a clear project before you check them in!
One odd thing I noticed is this. The "git diff" you were doing before the stash will show your changes. Now you have to do "git diff HEAD". Not completely sure why, but there it is.