**[Initializing a Repository in an Existing Directory](http://git-scm.com/book/en/Git-Basics-Getting-a-Git-Repository" \l "Initializing-a-Repository-in-an-Existing-Directory)**

If you’re starting to track an existing project in Git, you need to go to the project’s directory and type

$ gitinit

This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet. (See *Chapter 9* for more information about exactly what files are contained in the .git directory you just created.)

If you want to start version-controlling existing files (as opposed to an empty directory), you should probably begin tracking those files and do an initial commit. You can accomplish that with a few gitaddcommands that specify the files you want to track, followed by a commit:

$ git add \*.c

$ git add README

$ git commit -m 'initial project version'

=>If you want to check your settings, you can use the git config --list command to list all the settings .

=>For example, you can get the manpage help for the config command by running

$ git help config

=>git add command (it’s a multipurpose command — you use it to begin tracking new files, to stage files, and to do other things like marking merge-conflicted files as resolved)

//shortcut for adding in all the changes to the files in the current directory and below

git add .

=>$ git diffcommand result tells you the changes you’ve made that you haven’t yet staged.

[That command compares what is in your working directory with what is in your

staging area.]

=>git diff –staged command compares your staged changes to your last commit

[To exit VIM editor use :q or :q! or :wq command]

=>If you want to skip the staging area, Git provides a simple shortcut

$ git commit -a -m 'added new benchmarks'

Git automatically stage every file that is already tracked before doing the commit, letting you skip the git add part

=> If you type gitk on the command line in your project, you should see visual git log tool.

=> The gitrm command remove a file from Git and also removes the file from your working directory.

=>$ git log

One of the more helpful options is -p, which shows the diff introduced in each commit. You can also use -2, which limits the output to only the last two entries

$ git log -p -2

Other options:

git log --pretty=oneline --max-count=2

git log --pretty=oneline --since='5 minutes ago'

git log --pretty=oneline --until='5 minutes ago'

git log --pretty=oneline --author=<your name>

git log --pretty=oneline --all

I like the following log format for most of my work

git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short

=>Common Aliases

Add the following to the .gitconfig file in your $HOME directory

[alias]

hist = log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short

With these aliases defined in the .gitconfig file you can type $ **githist**which allow you to avoid the really long log command

=>Return the latest version in the master branch

$ git checkout master

=>Tagging

$ git tag v1

Now you can refer to the current version of the program as v1.

Tagging Previous Versions

$ git checkout v1^ [or provide hash of any other commit]

=> Revert changes in the working directory

$ git checkout <file>

=> Revert changes that have been staged

$ git reset HEAD <file>

The reset command (by default) doesn’t change the working directory. We can use the checkout command to remove the unwanted change from the working directory.

=> Revert changes that have been committed to a local repository

$ git reset --hard v1 or $ git reset --hard <hash>

The --hard parameter indicates that the working directory should be updated to be consistent with the new branch head.

How to amend an existing commit.Let’s amend the previous commit to include the email change.

$ git add <file>

$ git commit --amend -m "Add an author/email comment"

You can achieve the same effect by resetting the branch back one commit and then recommitting the new changes.

Move a file within a repository

$ gitmv <file><dir>

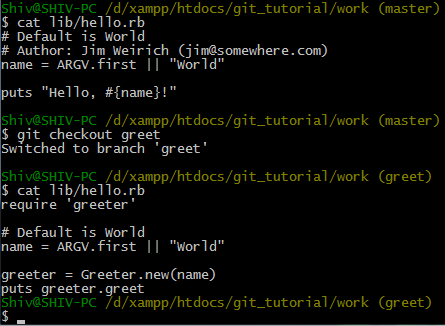
$git commit -m "Moved hello.txt to lib"

Branching

Let we have a file called lib/hello.rb and then we created new branch ‘greet’

$ git checkout -b greet

then we have edited lib/hello.rb, we can noticed that files are different in two branches



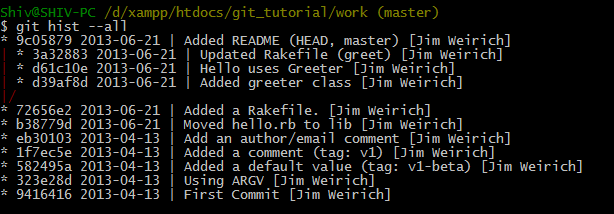
# Merging

We have created new file ‘README’ in Mater Branch. Let’s go back to the greet branch and merge master onto greet.

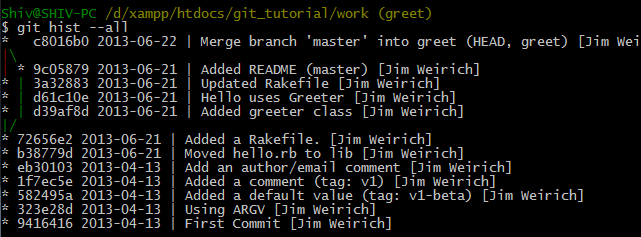
$ git checkout greet

$ git merge master

Before Merger ($ git hist --all)



After Merger ($ githist --all)



Important: By merging master into your greet branch periodically, you can pick up any changes to master and keep your changes in greet compatible with changes in the mainline.

## [Git for Windows tip: Use P4Merge as mergetool](http://danlimerick.wordpress.com/2011/06/19/git-for-window-tip-use-p4merge-as-mergetool/)

1. Download P4Merge: Visual Merge Tool from <http://www.perforce.com/perforce/downloads/component.html>
2. In the installer for P4Merge you can choose which components you wish to install, you only need the Visual Merge Tool (P4Merge).
3. Install p4merge and then set it as your merge tool for git by running the following two config commands:
   1. gitconfig --global merge.tool p4merge
   2. gitconfig --global mergetool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"
4. P4Merge can be used for doing diffs and merges. If using P4Merge for diffs then call:

gitdifftool

If the file you want to compare is already staged then use the –cached switch after that command.

1. When Git tells you that there has been conflict, to resolve it type:

gitmergetool

And this is what a 3-way merge looks like.

(1) remote version (the master branch) (2) base version (3) local version (local branch)

Below is the final file. Click on desired colored icon and save the changes.

# Cloning Repositories

=>Go to the working directory and make a clone of your hello repository.

$ git clone hello cloned\_hello

=>What is origin?

$ git remote show origin

Remote repositories typically live on a separate machine, possibly a centralized server. As we can see here, however, they can just as well point to a repository on the same machine. There is nothing particularly special about the name “origin”.

=>Fetching Changes from remote (hello) local (cloned\_hello)

$ git fetch origin

the “git fetch” command will fetch new commits from the remote repository, but it will not merge these commits into the local branches.

=>$ git merge origin/master

Even though “git fetch” does not merge the changes, we can still manually merge the changes from the remote repository.

=>**git pull is equivalent to a git fetch followed by a git merge.**

# Add a local branch(greet) that tracks a remote branch(greet)

The branches starting with ‘remotes’ or ‘origin’ are branches from the original repo.

$ git branch --track greet origin/greet

Branch ‘greet’ will be set up to track remote greet from origin (origin/greet)

# Creating Bare Repositories for sharing [Server Side]

Bare repositories (without working directories [hello]) are usually used for sharing, usually shared on some sort of network server.

$ git clone --bare hello hello.git

The convention is that repositories ending in ‘.git’ like git://github.com/paulboone/ticgit.git are bare repositories.

# Pulling Shared Changes[Client Side]

Let’s pull down the changes just pushed to the shared repo. ’shd’ is the short name of the repository (hello.git) receiving the changes we are pulling.

$ git remote add shd ../hello.git

$ git branch --track shd master

$ git pull shd master [Note: **git pull is equivalent to a git fetch followed by a git merge]**

# How to push any change in working directory to the bare repository

=>Made required changes in the working directory (hello).

=>Use bare repository with local name ‘shared’.

$ git remote add shared ../hello.git

=>Now push any change in working directory (hello) to the shared repo.

$ git push shared master

‘shared’ is the local name of the bare repository receiving the changes we are pushing. We had to explicitly name the branch ‘master’ that was receiving the push.

# Hosting your Git Repositories

(From the work directory [i.e ../hello])

$ git daemon --verbose --export-all --base-path=.

Now, in a separate terminal window, go to your work directory

$ git clone git://localhost/hello.git network\_hello

You should see a copy of hello project.

## Working with Remotes

If you’ve cloned your repository, you should at least see origin — that is the default name Git gives to the

server you cloned from:

$ git clone git://github.com/schacon/ticgit.git

=>To see which remote servers you have configured, you can run the git remote -v command.

=>To add a new remote Git repository as a shortname you can reference easily, run git remote add [shortname] [url]

e.g. $ git remote add pb git://github.com/paulboone/ticgit.git

You can run git fetch pbto fetch all the information that Paul has

=>It’s important to note that the fetch command pulls the data to your local repository — it doesn’t automatically merge it with any of your work or modify what you’re currently working on. You have to merge it manually into your work when you’re ready.

$ git merge pb/master

=> If you have a branch set up to track a remote branch, you can use the git pull command to automatically fetch and then merge a remote branch into your current branch; and by default, the git clone command automatically sets up your local master branch to track the remote master branch on the server you cloned from (assuming the remote has a master branch).

$ git pull pb master [or simply git pull from master branch]

**Pushing**

$ git push (remote) (branch)

e.g. $ git push origin serverfix

Which means, “Take my serverfix local branch and push it to update the remote’s serverfix branch.”

You can use this format to push a local branch into a remote branch that is named differently. If you didn’t want it to be called serverfix on the remote, you could instead run git push origin serverfix:awesomebranch to push your local serverfix branch to the awesomebranch branch on the remote project.

=>It’s important to note that when you do a fetch that brings down new remote branches, you don’t automatically have local, editable copies of them. In other words, in this case, you don’t have a new serverfix branch—you only have an origin/serverfix pointer that you can’t modify.

To merge this work into your current working branch, you can run git merge origin/serverfix. If you want your own serverfix branch that you can work on, you can base it off your remote branch:

$ git checkout -b serverfix origin/serverfix

This gives you a local branch that you can work on that starts where origin/serverfix is.

**Tracking Branches**

Checking out a local branch from a remote branch automatically creates what is called a tracking branch.

If you’re on a tracking branch and type git push, Git automatically knows which server and branch to push to. Also, running git pull while on one of these branches fetches all the remote references and then automatically merges in the corresponding remote branch.

When you clone a repository, it generally automatically creates a master branch that tracks origin/master. That’s why git push and git pull work out of the box with no other arguments. However, you can set up other tracking branches if you wish—ones that don’t track branches on origin and don’t track the master branch. The simple case is the example you just saw, running git checkout -b [branch] [remotename]/[branch]. If you have Git version 1.6.2 or later, you can also use the --track shorthand:

$ git checkout --track origin/serverfix

To set up a local branch with a different name than the remote branch, you can easily use the first version with a different local branch name:

$ git checkout -b sf origin/serverfix

Now, your local branch sf will automatically push to and pull from origin/serverfix.

**Deleting Remote Branches**

You can delete a remote branch using the rather obtuse syntax git push [remotename] :[branch]. If you want to delete your serverfix branch from the server, you run the following:

$ git push origin :serverfix

Rebasing