# Git Training - Advanced

Created: 30 January, 2019 – Modified: 30 January, 2019

Table of Contents

[Git Training - Advanced 1](#_Toc536706305)

[Overview 1](#_Toc536706306)

[Removing a Tracked File 1](#_Toc536706307)

[Revert to a Previous Commit 1](#_Toc536706308)

[Finding the SHA Key 2](#_Toc536706309)

[Revert to a Set of Files in a Previous Commit 2](#_Toc536706310)

[FTP Management 2](#_Toc536706311)

[Installing 2](#_Toc536706312)

[Setup 3](#_Toc536706313)

[Initializing 3](#_Toc536706314)

[Updating Files 3](#_Toc536706315)

[Workflow 4](#_Toc536706316)

[An Easier Way 4](#_Toc536706317)

[Ignore Files and Folders 5](#_Toc536706318)

[For the Repository on the Local Machine Only 5](#_Toc536706319)

[For the Project 5](#_Toc536706320)

[References 5](#_Toc536706321)

## Overview

This document will cover how to untrack a file, revert to a previous commit, and revert to a set of files in a previous commit, set up Git to push to a server, and specify files for Git to ignore.

## Removing a Tracked File

At some point in a project there may be file that no longer needs to be tracked and needs to be removed from the repository. These files could have been staged accidently, are no longer used yet aren't going to be deleted from the working directory, etc. To stage a file for removal from the repository:

$ git rm --cached <filePath> //flags the file for removal from the remote server

$ git commit -m 'removing file'

$ git push

## Revert to a Previous Commit

It is fairly common to make a commit, test the changes, and find out they broke the program. During development, this is no big deal but on a production server, it can be detrimental. In the second case, the good version needs to be restored quickly. There are two ways to return to a previous commit. Each is dependent upon the situation. If the situation is that commits were made to the local repository but not yet pushed to the remote server:

$ git reset <last good SHA> //this rewinds the repository history to the specified commit

However, if the changes have been pushed to the remote server:

$ git revert <SHA> //this will create a new commit that will revert all changes back to that point with no history delete

$ git push

## Finding the SHA Key

In order to reset or revert to a previous commit, the SHA key of that commit needs to be known. To find that key:

$ git log //opens the commit history in the default text editor

//or

$ git log -1 //prints out the last commit information to the terminal; the number can be changed to indicate how many of the previous commits to print out

## Revert to a Set of Files in a Previous Commit

Sometime only a set of files need to reverted to from a previous commit. In order to do that:

git checkout <SHA> -- path/to/file1 path/to/file2 //retrieve the files from the specified SHA

This replaces the file in the current working tree with the file from the specified previous commit. Any number of files can be listed at the end of this command.

## FTP Management

A Git extension called Git-FTP allows uploading files to a remote server easy and efficient. It uses Git logic and keywords for managing uploads. The workflow of a project should be:

1. Making changes
2. Committing those changes
3. Pushing the commit to the repository server
4. Pushing the commit to the development or production server

Step 3 can be skipped if only one developer is working on that branch but it is good practice to include step 3 so that there is verification that the branch has not been modified by another developer. Following these steps will prevent developers from overwriting one another's changes and insure the integrity of the code.

### Installing

Installing Git-FTP is straightforward on a Linux based system.

git clone https://github.com/git-ftp/git-ftp.git

cd git-ftp

git checkout "$(git tag | grep '^[0-9]\*\.[0-9]\*\.[0-9]\*$' | tail -1)" //changes to the latest stable version

sudo make install //installs Git-Ftp

### Setup

The FTP credentials will need to be written to the configuration file to allow files to be uploaded.

$ git config git-ftp.user baseline //remember the username

$ git config git-ftp.url <ftp://184.171.249.2/public_html/> //sets the upload destination

$ git config git-ftp.password secr3t //sets the password

This will upload the changed files to the public\_html/ folder when changes are pushed to the remote server.

### Initializing

Once Git-FTP has been set up, the project needs to be pushed to the remote server. This can be accomplished one of two ways.

#### No Files on the Server

If there are not files on the server and this is a brand new project being uploaded:

$ git ftp init

This will upload all the tracked files from the most recent commit.

#### Existing File on the Server

If there are already files on the server:

$ git ftp catchup

This does not upload any files except the Git files used to track the state of the files on the remote server. Effectively what it is saying is the state of the files on the remote server match the state of the files on the local machine. Thus only files that change from this point on will be uploaded

### Updating Files

From this point forward, only files that have changed in the repository since the last push will be uploaded. So, for example, if the last upload occurred two commits ago, all the files that have changed since two commits ago will be uploaded. To upload the changed files:

$ git ftp push

### Workflow

Using Git, there can be several different versions of the same site on the remote server at the same time. This works well with branching for testing the changes being made on that branch. Most development should be accomplished on the local machine, but when it is time for that to be reviewed, it will need to be uploaded to a remote server. Following the procedures above with a slight modification will allow the testing variations of the same site.

For example, let us assume Ben and Joshua are working on FullServiceReview. Ben is attempting to fix a bug and Joshua is attempting to add a new feature. Using the same test server would potentially cause development conflicts among other problems. It would be better if each used a different test server or have separate folders on the same test server. This could be done the traditional way through an FTP client or cPanel duplication. Or it could be easily accomplished using Git-FTP. The only variation that would need to be made is changing which folder the files are uploaded to.

Ben would edit his Git-FTP configuration by telling it to upload his files to a subfolder. Best practices would be to give it the same name as the branch.

$ git config git-ftp.url <ftp://184.171.249.2/public_html/bug#137/> //sets the upload destination to subfolder 'bug#137'

$ git ftp init //uploads all the tracked files to that folder

Now all the changes Ben makes will be uploaded to that folder which could also be browsed as a sperate version of the website.

Joshua would edit his Git-FTP configuration by telling it to upload his files to a subfolder.

$ git config git-ftp.url <ftp://184.171.249.2/public_html/menu-fade/> //sets the upload destination to subfolder 'menu-fade'

$ git ftp init //uploads all the tracked files to that folder

Now all the changes Joshua makes will be uploaded to that folder which could also be browsed as a sperate version of the website.

When Ben and Joshua are done with their update, the would then merge the master branch into their development branch, upload and test the system again, merge the development branch back into the master branch, and delete the development folder on the remote server. Then they would push these changes to the remote repository, change their git-ftp.url to the production server, and upload the changes.

### An Easier Way

Not all of these steps need to be accomplished or managed through the terminal. Git stores the settings in the root of the repository folder '.git/config'. The entry looks like this:

[git-ftp]

user = baseline #development server username

password = secr3t #development server password

url = ftp://184.171.249.2/public\_html/ #development server

#url = ftp://184.171.249.2/public\_html/bug#137 #development server bug#137 branch

#url = ftp://fullservicereview.com/public\_html/ #production server

## Ignore Files and Folders

Files and folders can be excluded from being seen by Git. This is useful when viewing tracked and untracked files as it will exclude all ignored file from that list. Files and folders can be added in sever ways:

* file.php – exclude a specific file
* assets\_backup/ - exclude an entire folder
* \*.orig – exclude all the files with the 'orig' extension'

### For the Repository on the Local Machine Only

* Edit the file '.git/info/exclude'

### For the Project

* Create and edit a '.gitignore' file in the root of the repository

## References

* <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>
* <https://github.blog/2015-06-08-how-to-undo-almost-anything-with-git/>
* <https://gist.github.com/Nilpo/8ed5e44be00d6cf21f22>
* <https://stackoverflow.com/questions/215718/reset-or-revert-a-specific-file-to-a-specific-revision-using-git>
* <https://stackoverflow.com/questions/4114095/how-to-revert-a-git-repository-to-a-previous-commit>
* <https://github.com/git-ftp/git-ftp>