Documentation: Help with Git and Github

|  |
| --- |
| *Table of Contents: The TOC is auto-created by using "Headings". To update, click below, then click the "Update table of contents" button.*  [\* Glossary](#_w3ugnw40xzqq)  [Important: Recommended Order of Installation](#_39s2n3st2jk)  [Setup Instructions: Install "Github for Desktop (PC or Mac)" or "Git"](#_pa4kye9bvag1)  [How to Create a Git Repository in an Existing Directory](#_v0zwiygblko)  [1. Initialize a Repository in an Existing Directory](#_2tlj0551ac26)  [2. Clone an Existing Repository](#_teimis6eekds)  [How to Work with a Git Repository](#_3nbaxpk24hbf)  [Git repositories need a README.md and .gitignore file](#_1aydt7x2lfhp)  [How to Add files to version-control](#_l2vclfkiedgf)  [How to Commit files to version-control](#_q8hf6yvtqro2)  [How to Pull files from the Cloud repository](#_2jk5jno40oyp)  [How to Push files from local repository to the Cloud repository](#_hlr4adch1bze)  [How to Remove files from version-control](#_dlgvdleb9wi0)  [Remote Repositories](#_94cftkvqbayw)  [How to Clone a Repo and Change the Remote pointer](#_5wk4j5enq3lc)  [GitHub Repository Management](#_f3rrfjrv5i9f)  [How to Fork a Repository](#_385dj98mqv70)  [How to Issue a Pull Request](#_r3o4aklcttmd)  [Overview: How to Contribute Code](#_abza1pd1t5wq)  [More Great Help Documentation](#_6ysy3t5xsv4s) |

## \* [Glossary](https://git-scm.com/docs/gitglossary)

* **git** - Git was designed in order to interface with remote repository. Each repository is resolved to a URL, and given a name. By convention, GitHub uses the name "origin" for the repository on its server.
* **add** - Moves changes from the working directory to the staging area. This gives you the opportunity to prepare a snapshot before committing it to the official history.
* **branch** - This command is your general-purpose branch administration tool. It lets you create isolated development environments within a single repository. A branch in Git is simply a lightweight movable pointer to a commit. The default branch name in Git is master. As you start making commits, you're given a master branch that points to the last commit you made. Every time you commit, it moves forward automatically. The "master" branch in Git is not a special branch. It is exactly like any other branch. The only reason nearly every repository has one is that the git init command creates it by default and most people don't bother to change it
* **checkout** - In addition to checking out old commits and old file revisions, git checkout is also the means to navigate existing branches. Combined with the basic Git commands, it's a way to work on a particular line of development.
* **clean** - Removes untracked files from the working directory. This is the logical counterpart to git reset, which (typically) only operates on tracked files.
* **clone** - Creates a copy of an existing Git repository. Cloning is the most common way for developers to obtain a working copy of a central repository. You can do this at the github website using the "clone or download" button. Download as a zip file to your laptop, or use the following notation in a shell to download programmatically:
  + **$ git@github.com:Islandora-Collaboration-Group/icg\_information.git**
* **commit** - Takes the staged snapshot and commits it to the project history. Combined with git add, this defines the basic workflow for all Git users.
  + As a noun: A single point in the Git history; the entire history of a project is represented as a set of interrelated commits. The word "commit" is often used by Git in the same places other revision control systems use the words "revision" or "version". Also used as a short hand for [commit object](https://git-scm.com/docs/gitglossary#def_commit_object).
  + As a verb: The action of storing a new snapshot of the project's state in the Git history, by creating a new commit representing the current state of the [index](https://git-scm.com/docs/gitglossary#def_index) and advancing [HEAD](https://git-scm.com/docs/gitglossary#def_HEAD) to point at the new commit.
* **fetch** - Fetching downloads a branch from another repository, along with all of its associated commits and files. But, it doesn't try to integrate anything into your local repository. This gives you a chance to inspect changes before merging them with your project.
* **fork** - Users who don't have push permissions for a repository can "fork" it (create their own copy), push commits to that copy, and open a merge request from their fork back to the main project. This model allows the owner to be in CHAPTER 4: Git on the Server 148 full control of what goes into the repository and when, while allowing contributions from untrusted users.
* **init** - Initializes a new Git repository. If you want to place a project under revision control, this is the first command you need to learn.
* **issue pull request** - The github website offers a "New pull request" button for initiating a pull request. A pull request sends your code edits to the original repository owner for review; they can either accept or reject your edits into their repository.
* **log** - Lets you explore the previous revisions of a project. It provides several formatting options for displaying committed snapshots.
* **master** - The default development [branch](https://git-scm.com/docs/gitglossary#def_branch). Whenever you create a Git [repository](https://git-scm.com/docs/gitglossary#def_repository), a branch named "master" is created, and becomes the active branch. In most cases, this contains the local development, though that is purely by convention and is not required.
* **merge** - A powerful way to integrate changes from divergent branches. After forking the project history with git branch, git merge lets you put it back together again.
  + As a verb: To bring the contents of another [branch](https://git-scm.com/docs/gitglossary#def_branch) (possibly from an external [repository](https://git-scm.com/docs/gitglossary#def_repository)) into the current branch. In the case where the merged-in branch is from a different repository, this is done by first [fetching](https://git-scm.com/docs/gitglossary#def_fetch) the remote branch and then merging the result into the current branch. This combination of fetch and merge operations is called a [pull](https://git-scm.com/docs/gitglossary#def_pull). Merging is performed by an automatic process that identifies changes made since the branches diverged, and then applies all those changes together. In cases where changes conflict, manual intervention may be required to complete the merge.
  + As a noun: unless it is a [fast-forward](https://git-scm.com/docs/gitglossary#def_fast_forward), a successful merge results in the creation of a new [commit](https://git-scm.com/docs/gitglossary#def_commit) representing the result of the merge, and having as [parents](https://git-scm.com/docs/gitglossary#def_parent) the tips of the merged [branches](https://git-scm.com/docs/gitglossary#def_branch). This commit is referred to as a "merge commit", or sometimes just a "merge".
* **origin** - The default upstream [repository](https://git-scm.com/docs/gitglossary#def_repository). Most projects have at least one upstream project which they track. By default origin is used for that purpose. New upstream updates will be fetched into [remote-tracking branches](https://git-scm.com/docs/gitglossary#def_remote_tracking_branch) named origin/name-of-upstream-branch, which you can see using git branch -r.
* **pull** - Pulling is the automated version of git fetch. It downloads a branch from a remote repository, then immediately merges it into the current branch. This is the Git equivalent of svn update.
* **push** - Pushing is the opposite of fetching (with a few caveats). It lets you move a local branch to another repository, which serves as a convenient way to publish contributions. This is like svn commit, but it sends a series of commits instead of a single changeset.
* **rebase** - Rebasing lets you move branches around, which helps you avoid unnecessary merge commits. The resulting linear history is often much easier to understand and explore.
* **repository ("repo")** - A collection of [refs](https://git-scm.com/docs/gitglossary#def_ref) together with an [object database](https://git-scm.com/docs/gitglossary#def_object_database) containing all objects which are [reachable](https://git-scm.com/docs/gitglossary#def_reachable) from the refs, possibly accompanied by meta data from one or more [porcelains](https://git-scm.com/docs/gitglossary#def_porcelain). A repository can share an object database with other repositories via [alternates mechanism](https://git-scm.com/docs/gitglossary#def_alternate_object_database).
* **remote** - A convenient tool for administering remote connections. Instead of passing the full URL to the fetch, pull, and push commands, it lets you use a more meaningful shortcut.
* **remote repository** - A [repository](https://git-scm.com/docs/gitglossary#def_repository) which is used to track the same project but resides somewhere else. To communicate with remotes, see [fetch](https://git-scm.com/docs/gitglossary#def_fetch) or [push](https://git-scm.com/docs/gitglossary#def_push).
* **resolve** - The action of fixing up manually what a failed automatic [merge](https://git-scm.com/docs/gitglossary#def_merge) left behind.
* **revision** - Synonym for [commit](https://git-scm.com/docs/gitglossary#def_commit) (the noun).
* **status** - Displays the state of the working directory and the staged snapshot. You'll want to run this in conjunction with git add and git commit to see exactly what's being included in the next snapshot.

## Important: Recommended Order of Installation

1. Install git
2. Install virtualbox
3. Install vagrant
4. Install ansible

## Setup Instructions: Install "Github for Desktop (PC or Mac)" or "Git"

Choose either approach:

1. [Install Git](https://git-scm.com/download/) - Git installs without the WYSIWYG client and assumes a greater knowledge of typing git commands into a shell based environment. See [Git Documentation](https://git-scm.com/doc).
2. [Install Github for Desktop](https://desktop.github.com/) - GitHub Desktop is a WYSIWYG client to contribute to projects on GitHub. See [GitHub Desktop Documentation](https://help.github.com/desktop/).
3. TODO - What happens if you install both versions? Do they conflict?

## How to Create a Git Repository in an Existing Directory

### 1. Initialize 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:

*To initialize the new repository within a local working environment:*

* **$ git init**

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 [Git Internals](https://git-scm.com/book/en/v2/ch00/_git_internals) for more information about exactly what files are contained in the .git directory you just created.)

At this point, you have a Git repository with tracked files and an initial commit.

### 2. Clone an Existing Repository

If you want to get a copy of an existing Git repository – for example, a project you'd like to contribute to – the command you need is git clone. If you're familiar with other VCS systems such as Subversion, you'll notice that the command is "clone" and not "checkout". This is an important distinction – instead of getting just a working copy, Git receives a full copy of nearly all data that the server has. Every version of every file for the history of the project is pulled down by default when you run git clone. In fact, if your server disk gets corrupted, you can often use nearly any of the clones on any client to set the server back to the state it was in when it was cloned (you may lose some server-side hooks and such, but all the versioned data would be there – see [Getting Git on a Server](https://git-scm.com/book/en/v2/ch00/_git_on_the_server) for more details).

You clone a repository with **git clone [url]**. For example, if you want to clone the Git linkable library called libgit2, you can do so like this:

* **$ git clone** [**https://github.com/libgit2/libgit2**](https://github.com/libgit2/libgit2)

That creates a directory named "libgit2", initializes a .git directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version. If you go into the new libgit2 directory, you'll see the project files in there, ready to be worked on or used. If you want to clone the repository into a directory named something other than "libgit2", you can specify that as the next command-line option:

* **$ git clone** [**https://github.com/libgit2/libgit2**](https://github.com/libgit2/libgit2) **mylibgit**

That command does the same thing as the previous one, but the target directory is called **mylibgit**.

Git has a number of different transfer protocols you can use. The previous example uses the https:// protocol, but you may also see git:// or user@server:path/to/repo.git, which uses the SSH transfer protocol. [Getting Git on a Server](https://git-scm.com/book/en/v2/ch00/_git_on_the_server) will introduce all of the available options the server can set up to access your Git repository and the pros and cons of each.

Download a copy of the repository; you can do this at the github website using the "clone or download" button. Download as a zip file to your laptop, or use the following notation in a shell to download programmatically:

* **$ git@github.com:Islandora-Collaboration-Group/icg\_information.git**

Using SSH, you can clone like this:

* **$ git clone username@host.com:relative/path/to/repo.git**

## How to Work with a Git Repository

### Git repositories need a README.md and .gitignore file

1. **README.md -** Each project should have at least one README file in markdown. This is parsed by GitHub, and rendered as HTML documentation.
2. **.gitignore -** This file contain any number of globbing patterns or file/directory paths relative to the repository path itself. It ensures that certain files cannot be versioned by Git (i.e. .cfg files with passwords, or backup files)

### How to Add files to version-control

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 git add commands that specify the files you want to track, followed by a git commit:

To add all files:

* **$ git add \*.\***

Or to only add a certain type of files (i.e. .txt):

* **$ git add \*.txt**

Or to add one specific file:

* **$ git add README.md**

Or to add multiple (in this case, two) specific files:

* **$ git add README.md .gitignore**

### How to Commit files to version-control

Git saves the state of a given branch for tracked files by "commits". Should one add a file for tracking or modify a file further after a commit, another commit must be issued. For the purposes of this outline, consider commits to only save the state of the local repository.

To commit all files with a descriptive message (the -m is the message flag):

* **$ git commit -m 'initial project version'**

### How to Pull files from the Cloud repository

Pull all files that are newer than those in your local repository:

* **$ git pull**

### How to Push files from local repository to the Cloud repository

Git synchronizes commits on local branches to branches in remote repositories using a "push" operation. The "-u" argument ensures that the branch "master" in the remote repository "origin" is tracked by the local branch "master". This, in turn, permits one to more easily synchronize modifications made to the branch "master" on origin, to the local branch, which in this case is called "master".

* **$ git push -u origin master**

### How to Remove files from version-control

One may remove files from tracking using the following invocation:

* **$ git rm README.md**

## Remote Repositories

Git has been developed primarily under the assumption that contributors would be interfacing primarily with other repositories using either over the SSH of HTTPS protocols. Within most Git projects, at least one remote repository is specified, as this repository is located upon a server running the [Git daemon service](https://git-scm.com/book/en/v2/Git-on-the-Server-Git-Daemon). GitHub offers its server resources freely to user members and organizations with the understanding that hosted projects should adhere to the principles of the Open Source software movement, and by promoting similar, related services which integrate with GitHub's own approach to Git daemon hosting (e. g. [Travis CI](https://travis-ci.org/) for continuous integration). Further, GitHub offers a number of higher-level repository management operations which are not offered by Git itself, and these must be taken into consideration for any organization investing within the architecture comprising those services offered by GitHub.

List the remote connections you have to other repositories:

* **$ git remote -v**

## How to Clone a Repo and Change the Remote pointer

Scenario: You want to take someone else's repo and put it on a github account of which you are a member.

1. On github.com, go to your account and click on "New repository" and name it: "twitter".
2. On your local machine, open a shell command and cd to where you want to put the repo:
   1. **$ git clone** [**https://github.com/sferik/twitter.git**](https://github.com/sferik/twitter.git)
3. cd into the new directory:
   1. **$ cd twitter**
4. List the remote connections you have to other repositories:
   1. **$ git remote -v**
5. This will show something like:
   1. origin <https://github.com/sferik/twitter.git> (fetch)
   2. origin <https://github.com/sferik/twitter.git> (push)
6. Add an additional source (beyond your standard associated github account) from which your local repository will be able to pull files. Here, we are calling it "upstream" because that makes sense, but we also could have called it "pizza" ("upstream" is not a reserved word).
   1. **git remote add upstream https://git@github.com/sferik/twitter.git**
7. Remove the remote origin path (that means, remove the original path that was associated with this repo)
   1. **git remote rm origin**
8. Assign a new remote origin path to which your local repo will now be able to push files:
   1. **$ git remote add origin** [**git@github.com**](mailto:git@github.com)**:Islandora-Collaboration-Group/drush-sitespinner.git**
9. See what the status is:
   1. **$ git status**
10. See how many branches exist:
    1. **$ git branch --all**
11. Now, the big moment: Push your local repo that you cloned up to your own github repository that you created in step #1. Below, you are pushing your master branch to the URL associated with origin.
    1. **$ git push origin master**
12. You have now cloned someone's repo to your your laptop and then pushed it to a desired github account where you and your team can collaborate on it.

## GitHub Repository Management

The two most common repository management operations undertaken are the "fork" and the "pull request". Both implement a certain structured, automated approach to tasks which could certainly be executed using Git alone.

### How to Fork a Repository

[Forking a GitHub Git repository](https://help.github.com/articles/fork-a-repo) essentially performs a "git clone" operation, but also provides GitHub's services with metadata regarding the nature of the clone. This permits one to more easily structure and visualize the relationships between cloned Git repositories, as well as to foster collaborative relationships between users who fork open source projects. Further, it also provides GitHub with metadata which permits it to more easily integrate changes introduced by individuals forking a project.

Forking a repository means that you are downloading a copy of someone else's repository and your fork is a distinct, entirely separate, repository that shows the original as the ancestor. You can issue a "pull request" as a means of offering commits you've made to your own repository back to the original repository; of course, the owner of the original repository must accept that pull request in order to accept your edits.

The github website offers a "fork" button

### How to Issue a Pull Request

A ["Pull Request"](https://help.github.com/articles/using-pull-requests) is a request issued by a party which has performed a "fork" on a project on GitHub with the intention of contributing to the original source code base (hence, it can be misleading to refer to this as a "true" forking of a project; often times the intention is to resolve bugs or refactor functionality where the outlying party may not be able to directly "push" to the repository holding the original project). A "Pull Request" indicates to the repository administrator(s) that users who have performed a "fork" of their project wish to contribute their own modifications to the source code base of the original project. The repository administrator(s) of the original project may decline the request, and more often than not, questions regarding quality assurance (i. e. testing) are raised before any "pull requests" made by third parties are accepted, and their contributions integrated. Promoting the "forking" of projects and the issuing of "pull requests" is the objective of GitHub. Ideally, individuals discover open source projects (of which they were unaware), extend and improve upon the application or service, and contribute to the original project's source code base.

The github website offers a "New pull request" button for initiating a pull request. A pull request sends your code edits to the original repository owner for review; they can either accept or reject your edits into their repository.

## Overview: How to Contribute Code

1. If you do not have your own github account, sign up for one.
2. Visit the home page for this repository and click "Fork" at the upper-right. This will create a repository within your github space that is a clone of this original.
3. Clone your fork down to your development environment. Or if you have already cloned Islandora-Collaboration-Group/icg\_csv\_import, you can change the remote [instructions](https://help.github.com/articles/changing-a-remote-s-url/).
4. Work on your local code, make changes, commit your changes, push back up to your github remote etc.
5. You can keep your fork updated from the ICG master [using this technique](http://stackoverflow.com/questions/7244321/how-do-i-update-a-github-forked-repository)
6. When you believe your work is ready for prime time, on your github repository, click the "Create Pull Request" button. Select Islandora-Collaboration-Group/icg\_csv\_import:master branch as the target. Describe the work you have done and what problem it solves.
7. Other "watchers" on this repository will be notified of your pull request. They can merge it into their own forks and test it.
8. Testers might have feedback for you and comment on the pull request. You might want to make changes and either withdraw your pull request or update it.
9. Once we are satisfied that your work should be committed to the master branch, the pull request will be merged and closed.

## More Great Help Documentation

1. <https://progit2.s3.amazonaws.com/en/2016-03-22-f3531/progit-en.1084.pdf>
2. <https://git-scm.com/>
3. <https://desktop.github.com/>
4. [Got 15 minutes and want to learn Git?](https://try.github.io/levels/1/challenges/1)
5. [Atlassian - Git Tutorials](https://www.atlassian.com/git/tutorials/syncing)
6. [Atlassian - Getting Git Right](https://www.atlassian.com/git/)
7. [Atlassian - Git Glossary](https://www.atlassian.com/git/glossary/)