# The `git` Utility

> Git is a free and open source distributed version control system. - Git Website

GitHub, and competitors BitBucket and GitLab, all use Git as the underlying version control system.

References:

- https://github.com/git/git
- https://git-scm.com/doc
- https://guides.github.com/introduction/flow/
- https://education.github.com/git-cheat-sheet-education.pdf
- https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf
- https://git-scm.com/book/en/v2/Getting-Started-The-Command-Line
- https://www.atlassian.com/git/tutorials
- https://www.youtube.com/watch?v=MJUJ4wbFm_A
- https://help.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh

## Installation

First see if Git is already installed (it may come pre-installed):

```
# Mac Terminal:
git --version #> git version 2.20.1
which git #> /usr/local/bin/git

# Windows Command Prompt, Anaconda Prompt, or Git Bash:
git --version #> git version 2.20.1.windows.1
where git #> C:\Program Files\Git\cmd\git.exe
```

If these commands produce a version-looking output and a filepath-looking output, respectively, then Git is already installed and you can skip down to the "Git Commands Overview" section. Otherwise, follow the OS-specific sections below to install Git.

### Installation on Mac

Mac users can install Git from https://git-scm.com/downloads, or via homebrew (recommended) with `brew install git`.

After installing, restart your terminal application, where you should now be able to execute Git commands (like `git --version`).

### Installation on Windows

Windows users can install Git from https://git-scm.com/downloads. This installation should also install a program called Git Bash, which Windows users will use as their default command-line computing application.

The Git Bash application is where you can execute Git commands (like `git --version`). Depending on how you installed it, your Command Prompt and Anaconda Prompt applications may also have access to the Git CLI, in which case you can run Git commands from within those applications as well.

## Usage (Git Commands Overview)

> NOTE: after executing some commands like `git log` and `git diff`, you can press the "Enter" key to keep reading, and type "q" to quit when you are done.

> NOTE: after executing commands like `git pull`, you may find yourself at times in an unfamiliar-looking "Vi" text editor window, which you can exit by pressing the "shift + ZZ" keys.

### Local Repositories

#### Initializing a Local Repository

Navigate into a project directory, then initialize a new repository there:

```
cd path/to/my/project # where path/to/my/project is the actual path to your project directory
git init . # initialize a new git repository, creating a hidden folder called .git in your project's root directory
```

#### Viewing Revision History

> NOTE: newly-created repositories won't have any revisions to view until you make your first commit (see "Committing Changes" below)

List the most recent revisions:

```
git log
```

Show details about the most recent revision:

```
git show # optionally specify any commit's identifier, or "SHA", to show that specific commit (e.g. `git show a5290eda34e9e0d89b90ae1cc01afe7753c294b8`)
```

#### Making Revisions

Use your text editor to add, delete, and/or modify files, then save them.

**Committing Changes**

After making and saving changes, detect and review them:

```
git status # see which files have changed since the last commit
git diff # see how those files have changed (only shows diffs for files that
existed during the last version, not for newly created files)
```

After reviewing the changes, if you are satisfied, stage and commit them:

```
git add . # this "stages" the files for commit. specify a period (`.`) to add
all changed files, or specify a single filename to add only that file (e.g.
`git add path/to/file.py`)
git commit -m "my message" # saves the changes and adds a unique reference
identifier for this particular version
```

Continue to iteratively repeat the process of reviewing and committing revisions as you incrementally develop your software.

**Reverting Changes**

One of the biggest benefits of version control is the ability to revert to previous versions. If you need to restore the state of your repository to some previous commit:

```
git reset --hard abc123def456 # where abc123def456 is the identifier, or
"SHA", of the commit you would like to revert to
```

## Remote Repositories

When you create a new empty repository on GitHub, it will display a screen that contains the `REMOTE_ADDRESS`. Otherwise, to find the `REMOTE_ADDRESS` of any existing GitHub repository, visit its homepage and click the big green "Clone or download" button. The address should resemble `https://github.com/USERNAME/REPONAME.git` (HTTPS), or `git@github.com:USERNAME/REPONAME.git` (SSH).

> NOTE: to enable SSH, see this guide on connecting to GitHub via SSH, which will prompt you to generate a public/private key pair and save the files locally in your "~/.ssh" directory. These files help establish your identity to GitHub.

### Cloning Remote Repositories

If there is a remote repository you would like to download, clone it (where `REMOTE_ADDRESS` refers to the repository's remote address):

```
git clone REMOTE_ADDRESS
```

After cloning, a default remote address named "origin" is automatically created. You can check a local repository's remote addresses at any time:

```
git remote -v
```

### Managing Remote Addresses

Add or remove a local repository's remote addresses, where `REMOTE_NAME` refers to the name of the remote address, (e.g. "origin"), and `REMOTE_ADDRESS` refers to the repository's remote address:

```
git remote add REMOTE_NAME REMOTE_ADDRESS
git remote rm REMOTE_NAME REMOTE_ADDRESS
```

If you would like to upload the contents of a local repository to a remote address but you don't already have a remote repo, follow these steps in order:

1. Create a new repo on GitHub, then note its remote address.
2. From the command-line, navigate to the root directory of your existing local repository (e.g `cd path/to/my-dir`).
3. Configure a "remote" address for your local repository: `git remote add origin REMOTE_ADDRESS`. NOTE: the overwhelming convention is to name your default GitHub remote address "origin".
4. Associate the local repo with the remote repo (one-time, first-time only): `git pull origin master --allow-unrelated-histories`. After doing so, you may be in a "Vi" text editor window, which you can exit by pressing the "shift + ZZ" keys.
5. Follow the "Syncing Local and Remote Repositories" section, below, to push your local changes up to GitHub.

### Syncing Local and Remote Repositories

Assuming you have already associated a local repository with a remote address named "origin":

```
git pull origin master # downloads recent contents from the remote repo, in
case changes have been made to the remote repo since you last pushed.
```

```
git push origin master # uploads local repo contents to remote address
```

Sometimes after pulling, you may see merge conflicts, in which case you might need to perform a "rebase" before being able to push. The rebase process can be difficult, so feel free to ask the professor for help.

After pushing successfully, you should be able to visit your remote repository on GitHub and see your code there.

## Collaboration

### Branch Operations

Branches offer separate namespaces for different versions of your code. This allows a single developer to try different approaches without affecting some known working version of the code. It also facilitates developer collaboration by allowing each developer a space to work, and by enabling Pull Request operations and reviews on GitHub.

The default branch is called "master", but developers usually do their work on a branch corresponding with some development goal (e.g. "my-new-feature"):

```
# determine which branch you're on:
git branch

# create and switch to a new branch named "my-new-feature":
git checkout -b my-new-feature

# switch between existing branches:
git checkout master
git checkout my-new-feature

# after committing some changes, push the branch up to GitHub:
git push origin my-new-feature
```

> FYI: when you switch branches, you're able to change between different versions of various files, so try making some changes and committing them and inspecting the files before and after you switch branches, to see the effects in action.