A Mostly Correct and Surprisingly Accurate Introduction To Git.

Credits

Git for Ages 4 And Up

Taught by Michael Schwern.

Linus Torvalds

Creator of Linux and Git

Git

What is it

- **Distributed** Source Code / Version Control Management System.
- Git is not a backup solution.

Why learn it

Besides being "fun", Git is becoming a necessary skill.

Anyone who is interested in GitHub, or ever working on a collaborative project will benefit from knowing how Git works.

Brief Introduction

Common Commands

Do Work

- Init
- Clone
- Status
- Log
- Add
- Commit

Experimentation

- Branch
- Checkout
- Merge

Collaboration

- Pull
- Push

Init

`git init`

Creates a new Git repository in the current directory inside `.git`

Clone

`git clone url/or/path/to/git/repository`

Copy a git repository from anywhere into a directory.

Status

`git status`

Lists what branch you have checked out, and other helpful information about your files / the staging area.

Log

`git log --decorate --graph`

Prints out your commit history and displays an ASCII graph of the repository.

Diff

`git diff file1 file2`

Useful for managing conflicts with multiple files.

Add

`git add file`

Add is the way you write a file to Git. When you add a file, you are adding it to the staging area.

Staging Area / The Index

The staging area is where files "wait" until you make a commit. It is a very fundamental and important concept in Git.

Commit

- Every ID is unique (If the ID is the same, then all the previous content is also the same)
- Every commit is unique

Commit Object

SHA Hash (ID)

- Content
- Author
- Date
- Log
- Previous Commit

Reference

Head

The `HEAD` is a reference to where you are currently working in the repository.

Branch

A `branch` is a "separate" part of your repository typically used to work on something (to add in later) without disturbing your previous work.

Tag

Similar to a branch except it marks a certain point in your commit history that will not change. For Example: (v0.9, v1.6, v2.0)

Branch

`git branch branch_name`

Branches let you make changes to your files without "damaging" what you already have. In other words, branching is for when you want to modify or add to your project without messing with what you are currently working on.

Master

Master is the name of the "main" branch in your repository.

Feature

Feature is an example branch name. In this branch we will add a *feature* to our project and then `merge` it back into `master`

Checkout

`git checkout branchname` or `git checkout -b branchname`

Checkout is the way you switch to another branch to work on. In the first command, you switch to a branch you have already created, and in the second example command, you crate a branch then switch

to it.

Tagging

- lightweight tag

`git tag tag_name`

This creates a reference to the current (where the `HEAD` is) position in the repository that will not change.

- annotated tag

`git tag -a tag_name`

An annotated tag will be stored as a full object in the Git database and will contain similar information as a commit. Annotated tags can also be signed and verified with GPG.

Merge

`git merge feature`

This command will merge the `feature` branch into the `master` branch. Note: We currently have `master` checked-out.

If you are working on a specific `branch` for a long time, then it would make sense to `merge` the latest `master` branch into `feature` first. This is done to ensure everything still works with the addition of your new `feature` before you `merge` it back into `master`.

Fast-forward

A fast-forward is a common type of `merge`, where the only changes to the `branch` are made "outside" or "ahead" of the current `branch`. Another example would be if you have not worked on `master` after checking out `feature`. In other words, a fast-forward is a *linear* `merge`.

Rebase

It is basically a fancy `merge`.

Rebasing allows you to modify the appearance of your commit history. If you ever hear about "re-writing history", this is what that is generally referring to. Rebasing is a very powerful tool, but we will only look at one example.

In our example, we use a `rebase` to *linearize* our commit history so that it is much more readable. This is very useful after working on `master` at the same time as `feature`. Rebasing can also be used to reorder or combine (squash) multiple commits.

Warning: You should **never** `rebase` *after* pushing to a remote repository. However, it is encouraged to do so *before* a `push`.

Remote

Adding a Remote

```
git remote add origin url/or/path/to/git/repository
```

This command adds a `remote` repository to your Git project. `origin` is the simply the name you are giving to it.

In all the `remote` commands we will use, the first argument is the remote branch name, and the second one is the local branch name.

Note: It is common to have multiple remote-repositories. To view all remotes, use the command, `git remote -v`.

Fetch

```
git fetch remote_name
```

When you `fetch` a remote-repository, you are simply retrieving the latest commits from that it. This will not result in a new commit in your local repository.

This is a great way to try and avoid conflicts. After you are satisfied with the new changes, you simply merge the `remote_name` with one of your local branches.

Pull

```
git pull origin master
```

A `pull` is just a `fetch` and `merge` combined into one command. This is mostly used when you have not made any changes to your local `branch` . Or, if you are confident that your commits will not be affected by changes on the `remote`.

Push

```
git push origin master
```

This is how you "send" or "share" your work with a remote-repository. Typically, this is where other people will `pull` from, in order to get your latest changes.

If `origin` is a GitHub repository, then you will be prompted for your GitHub username and password. Note: You can only `push` to somewhere that you have write access to.

You can think about a `push` as being a `commit` that is added to a copy of your repository (instead of your local one).

Resources

Reference / Tutorials

- Pro Git (Best)
- Try Git
- Git Simplified)

Talks

- Git for Ages 4 And Up
- Linus Torvalds on Git

Git GUI Clients

OS X

- Git-Cola
- Git X
- GitHub for Mac
- Git K (Included with Git)
- UnGit

Linux

- Git-Cola
- Git K (Included with Git)
- UnGit

Windows

- Git-Cola
- GitHub for Windows
- Git K (Included with Git)
- UnGit