git - the simple guide

just a simple guide for getting started with git. no deep shit ;)

Tweet

by Roger Dudler

credits to @tfnico, @fhd and Namics

deutsch, español, français, indonesian, italiano, nederlands, polski, português, pycck

ᏩᏂᏍᏛ, 日本語, 中文, 한국어 Vietnamese

please report issues on github

setup

Download git for OSX

Download git for Windows

Download git for Linux

create a new repository

create a new directory, open it and perform a

```
git init
```

to create a new git repository.

checkout a repository

create a working copy of a local repository by running the command

```
git clone /path/to/repository
```

when using a remote server, your command will be

```
git clone username@host:/path/to/repository
```

workflow

your local repository consists of three "trees" maintained by git. the first one is your `Working Directory` which holds the actual files. the second one is the `Index` which acts as a staging area and finally the `HEAD` which points to the last commit you've made.

add & commit

You can propose changes (add it to the **Index**) using

```
git add <filename>
```

```
git add *
```

This is the first step in the basic git workflow. To actually commit these changes use

```
git commit -m "Commit message"
```

Now the file is committed to the **HEAD**, but not in your remote repository yet.

pushing changes

Your changes are now in the **HEAD** of your local working copy. To send those changes to your remote repository, execute

```
git push origin master
```

Change *master* to whatever branch you want to push your changes to.

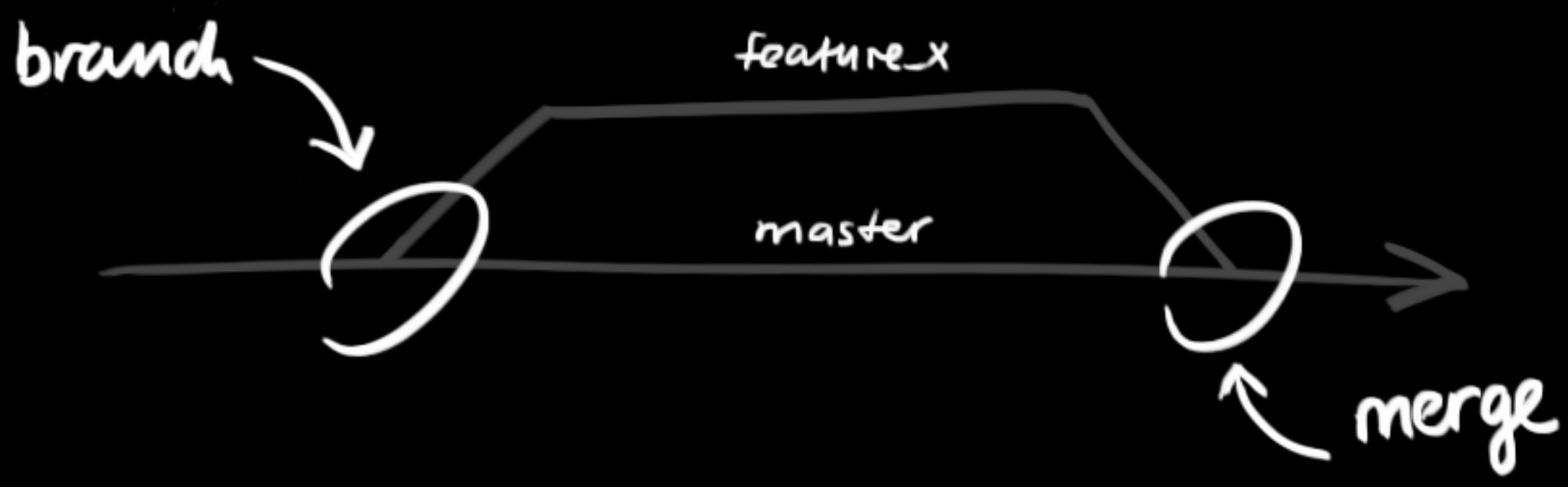
If you have not cloned an existing repository and want to connect your repository to a remote server, you need to add it with

```
git remote add origin <server>
```

Now you are able to push your changes to the selected remote server

branching

Branches are used to develop features isolated from each other. The *master* branch is the "default" branch when you create a repository. Use other branches for development and merge them back to the master branch upon completion.



create a new branch named "feature_x" and switch to it using

```
git checkout -b feature_x
```

switch back to master

```
git checkout master
```

and delete the branch again

```
git branch -d feature_x
```

a branch is *not available to others* unless you push the branch to your remote repository

```
git push origin <branch>
```

update & merge

to update your local repository to the newest commit, execute

```
git pull
```

in your working directory to *fetch* and *merge* remote changes.

to merge another branch into your active branch (e.g. master), use

```
git merge <branch>
```

in both cases git tries to auto-merge changes. Unfortunately, this is not always possible and results in *conflicts*. You are responsible to merge those *conflicts* manually by editing the files shown by git. After changing, you need to mark them as merged with

```
git add <filename>
```

before merging changes, you can also preview them by using

```
git diff <source_branch> <target_branch>
```

tagging

it's recommended to create tags for software releases. this is a known concept, which also exists in SVN. You can create a new tag named *1.0.0* by executing

```
git tag 1.0.0 1b2e1d63ff
```

the *1b2e1d63ff* stands for the first 10 characters of the commit id you want to reference with your tag. You can get the commit id by looking at the...

log

in its simplest form, you can study repository history using.. `git log`

You can add a lot of parameters to make the log look like what you want.

To see only the commits of a certain author:

```
git log --author=bob
```

To see a very compressed log where each commit is one line:

```
git log --pretty=oneline
```

Or maybe you want to see an ASCII art tree of all the branches, decorated with the names of tags and branches:

```
git log --graph --oneline --decorate --all
```

See only which files have changed:

```
git log --name-status
```

These are just a few of the possible parameters you can use. For more, see `git log --help`

replace local changes

In case you did something wrong, which for sure never happens ;), you can replace local changes using the command

```
git checkout -- <filename>
```

this replaces the changes in your working tree with the last content in HEAD. Changes already added to the index, as well as new files, will be kept.

If you instead want to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it like this

```
git fetch origin
```

```
git reset --hard origin/master
```

useful hints

built-in git GUI

```
gitk
```

use colorful git output

```
git config color.ui true
```

show log on just one line per commit

```
git config format.pretty oneline
```

use interactive adding

```
git add -i
```

links & resources

graphical clients

GitX (L) (OSX, open source)
Tower (OSX)
Source Tree (OSX & Windows, free)
GitHub for Mac (OSX, free)
GitBox (OSX, App Store)

guides

Git Community Book
Pro Git
Think like a git
GitHub Help
A Visual Git Guide

get help

Git User Mailing List
#git on irc.freenode.net

comments

978 Comments git - the simple guide Oualid Bo

Recommend 495 Share Sort by Newest

Join the discussion…

Carkod • 3 days ago
shouldn't
git --tag push
Also be included in the tagging section? it took me a while to realize that you have to push the tags....
Reply • Share ›

reddy910 • 4 days ago
wonderful document.
Reply • Share ›

disqus_SDuHGwj5tN • 19 days ago
In your section:

Checkout a repository
Remote Server

There is a much simpler method that requires no setup on the client, at least under Windows.:

git clone https://github.com/yourid/r...

Reply · Share ›

hariharan · 19 days ago

Very good material thanks a lot :)

1 Reply · Share ›

Daniel · 21 days ago

Hi I'm kinda new with git, and I have been facing many issues with my team lately, I was wondering if I'm following the right procedure:

1: take a git pull before working on anything else, or committing anything
2: if I have changes , I use git stash to put them aside, then git pull again, then git stash apply
3: If I got any conflict, I go manually file by file and approve the appropriate change for each using the "VS Code"
4: git add -A
5: git commit
6: git push

They say I comment their code and that it takes a while for them to merge anything from my commits. Also they say they are "basically" following the same procedure.

Am I doing anything wrong that may be causing them conflicts?

Thank you!

Reply · Share ›

Carlos Aleman · a month ago

Thank you!! This is great information :)

Reply · Share ›

Ashish Gupta · a month ago

I think good summary for people who know version control and git. To get a nice introduction, you can check out:

Reply · Share ›

David Kirui · a month ago

Really helpful

Reply · Share ›

seema mittal · a month ago

interesting way to share important things, really helpfull

Reply · Share ›

Pendi Madyana · a month ago

very useful, thanks

Reply · Share ›

bertenvdb • a month ago

Over the years I tried so many git gui's and always returned to CLI, until I found GitKraken! Free for Win+Mac+Linux https://www.gitkraken.com/

1 • Reply • Share ›

Deepak Kumar Singh • a month ago

Simple and easy. No Deep shit!!!

1 • Reply • Share ›

Rodolfo Contreras • a month ago

Best web ever!!!

• Reply • Share ›

Nate Maher • 2 months ago

Cool! Very helpful starting my new job!

• Reply • Share ›

Ahmed Magdi • 2 months ago

Awesome work :D

• Reply • Share ›

Mohneesh Sreegirisetty • 2 months ago

No Deep Shit! thank you for this awesome tutorial...

• Reply • Share ›

Emily Kauffman • 2 months ago

This is great!

• Reply • Share ›

Vikas Almal • 2 months ago

Loved your website!!

• Reply • Share ›

Sreekaanth Ganesan • 2 months ago

Very informative!!

• Reply • Share ›

ernestas • 2 months ago

Nice one!

• Reply • Share ›

Sushant Todkar • 2 months ago

Great information to learn GIT

• Reply • Share ›

Gopi • 3 months ago

Nice information for the beginners. Thanks for the creation.

1 • Reply • Share ›

Rhodimos Okon • 3 months ago

Wao this is great well done and thanks in a million

• Reply • Share ›

Djordje Arsenovic • 3 months ago

Well fucking done, thanks! :)

• Reply • Share ›

Will Ashworth • 3 months ago

Love this guide. Just shared it with everyone :)

• Reply • Share ›

Abhineet Raj • 3 months ago

Excellent work with the guide !!

Reply • Share ›

Tri Nguyen • 3 months ago

Aug 2017 and this is still useful. Thank you so much, man!

1 Reply • Share ›

Igor • 3 months ago

Uhmm, This is.. a good guide! :D

Reply • Share ›

wSafayat Jamil • 3 months ago

Thanks man, Really appreciate it. (From Bangladesh)

Reply • Share ›

Keith Kibler • 3 months ago

How about adding a "git stash" and "git stash pop" section? That way I only need a single web page for 98% of the git stuff I do? Please?

Excellent resource! Thanks!

Reply • Share ›

Rob • 3 months ago

Very usefull !

Reply • Share ›

GOKUL Kathiresn • 3 months ago

A most needed explanation in the best way

Reply • Share ›

victor Caballero • 3 months ago

simple and excellent !

1 Reply • Share ›

Joshua Eirman • 4 months ago

Real nice artistic incorporation!

1 Reply • Share ›

Joshua Eirman • 4 months ago

Is this an appropriate place to ask why there is a remote master and a local master branch? I can't find a place to ask, please help.

Thanks,
Josh

Reply • Share ›

Joshua Eirman → Joshua Eirman • 4 months ago

deleted

Reply • Share ›

Brian Lee → Joshua Eirman • 4 months ago

Git is a distributed version control system. This means each repo is self-contained and operate independently. The remote itself is a repo and your local copy is also it's own repo each tracking changes independently. After you update and create commits on master in your local repo you can use `push` to apply those same commits to a remote repo (usually named origin because that's where it was originally cloned from) and apply them on the remote repo's copy of master.

You technically don't need to have a local branch named master and a branch named master on the remote. You could name them something different, but that would be confusing. The convention is to name branches the same across cloned repos.

1 • Reply • Share ›

Joshua Eirman → Brian Lee • 4 months ago

Thank you Brian Lee, I am just finishing my immediate studies on GitHub and I had found this site.

Reply • Share ›

GitCoder • 4 months ago

Excellent compilation. As a suggestion, please add TortoiseGit to the list of Git clients. It's one of the best, oldest, and doesn't ask you to "create an account" to use.

2 • Reply • Share ›

Paras Gandhi • 4 months ago

great stuff

1 • Reply • Share ›

Sovichea Cheth • 4 months ago

This is very helpful!

Reply • Share ›

BlackYeti • 4 months ago

That was grate, thanks!

Reply • Share ›

ybl → BlackYeti • 4 months ago

I too love to grate my commits to bite size!

Reply • Share ›

Christopher John • 4 months ago

This is awesome

Reply • Share ›

Mova Club • 4 months ago

a good starting point to learn git - many thanks

Reply • Share ›

Rami Bakry • 4 months ago

helpful one ... thanks

Reply • Share ›

Alucaard • 4 months ago

I'm in deep shit

1 • Reply • Share ›

Cees Timmerman → Alucaard • 4 months ago

Why? Here's a sensible branching model that balances between the heavy