git - the simple guide

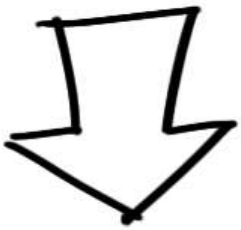
just a simple guide for getting started with git. no deep shit ;)

by Roger Dudler

credits to @tfnico, @fhd and Namics

ı deutsch, español, français, indonesian, italiano, nederlands, polski, português, pycc
မြန်မာ, 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

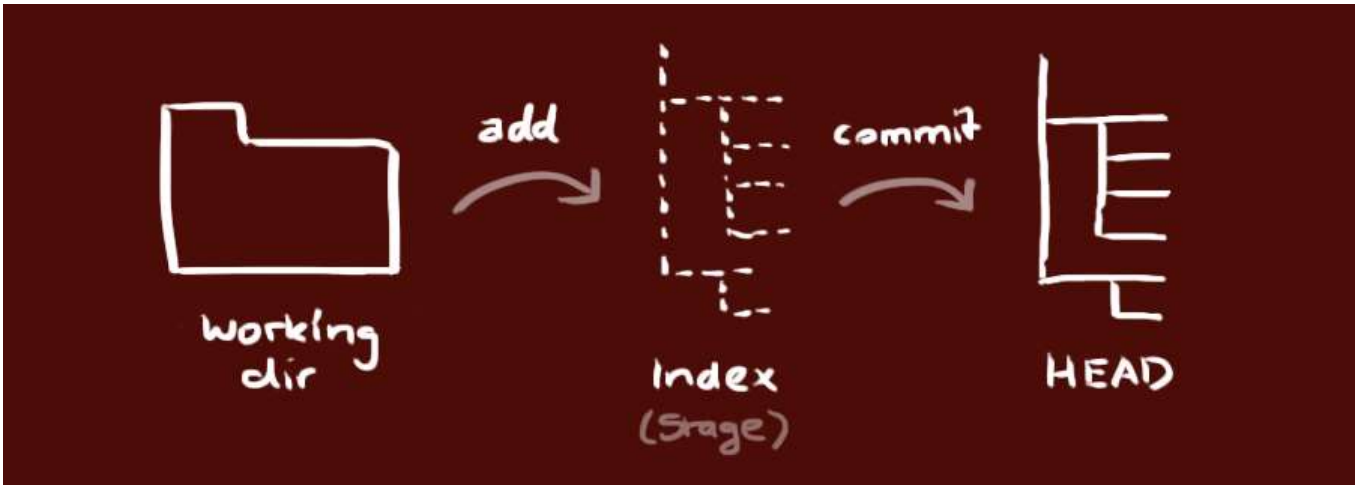
```
git clone /path/to/repository
```

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.

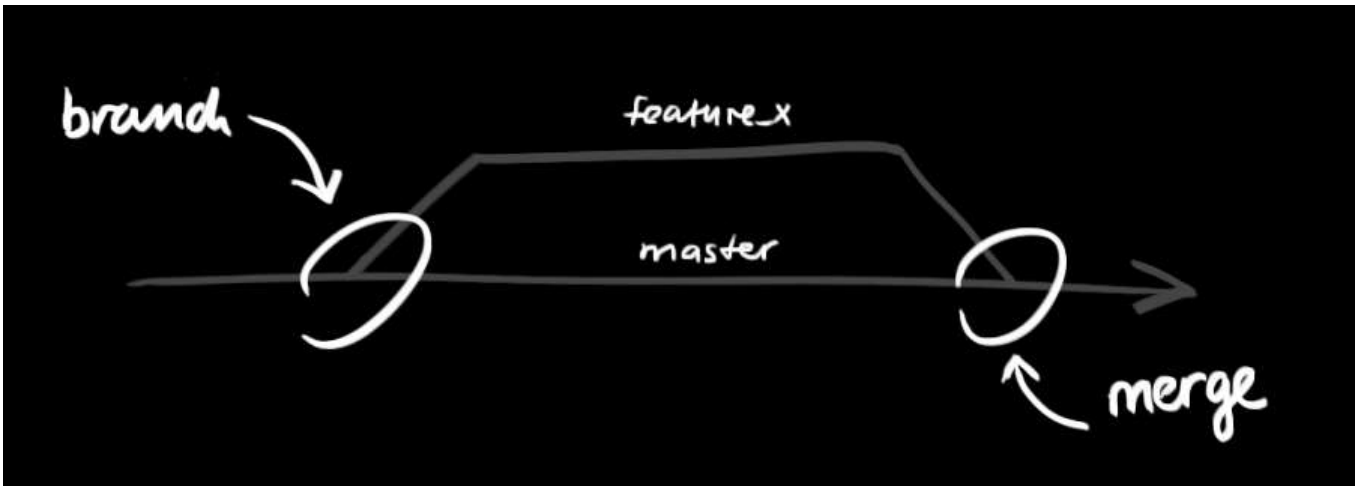
If you have not cloned an existing repository and want to connect your repository to a remote server, you need to add it with

```
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

583 Comments git - the simple guide 1 Login

Recommend 188 Share Sort by Newest

Join the discussion...

lingtalfi · a day ago
I love the fonts and your graphical style.
Reply · Share ›

Tyler Deans · 5 days ago
This tutorial was very helpful. Thanks.
Reply · Share ›

Anna Komnatnaya · 5 days ago
This is the best tutorial!
Reply · Share ›

Mincă Daniel Andrei · 6 days ago
That's the nicest beginner tutorial I've ever seen :)
Reply · Share ›

odnalrock · 7 days ago
is very clear, thanks
Reply · Share ›

moji · 13 days ago
wow! an overgood tutorial
Reply · Share ›

ali_ashoor · 14 days ago
WOW! this is the real deal
Reply · Share ›

mitesh · 18 days ago
to much useful and understandable
Reply · Share ›

MinhNhat Tran · 19 days ago
Awesome!!! Thanks a ton.
Reply · Share ›

yen mrkid · 21 days ago
this is the best I ever see. Clear ,Simple and important.
Reply · Share ›

Raghav · 22 days ago
Best tutorial for learning git(for people who are new to git). Really no deep S*#t

Best tutorial for learning git(for people who are new to git).Really no deep S**t just important stuff that matters.

Reply • Share ›

Star Wars • 23 days ago

Thank you for this nice and brief tutorial, specially workflow diagram.

Reply • Share ›

Felipe Lima • 24 days ago

Thank you my friend. Great post!

Reply • Share ›

wizston • a month ago

Nice and neat!

Reply • Share ›

Anurag Singh • a month ago

This is the shallow shit I was looking for ;). Really helpful, thanks for putting this together.

Reply • Share ›

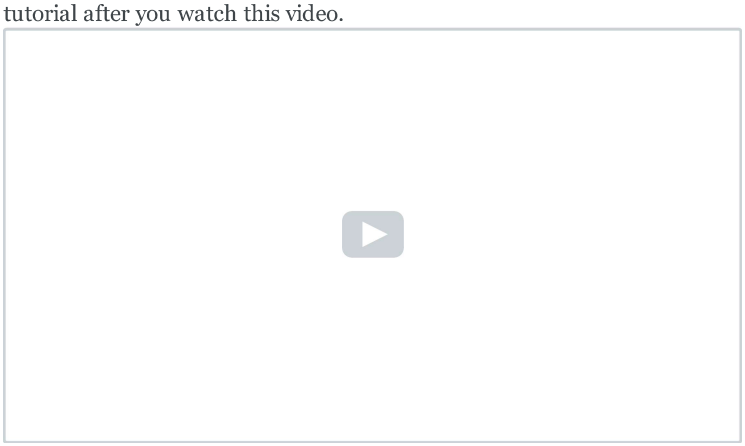
Take Note • a month ago

Useless since contains no example. I am dumb - I cant do anything without example - any explanation I understand in a different ways and see a lot options instead of one clear solution.

Reply • Share ›

Carmelo Rossette ➦ Take Note • a month ago

I know what you mean. This helped me out a lot. Come back to this tutorial after you watch this video.



2 • Reply • Share ›

Take Note ➦ Carmelo Rossette • a month ago

Thank you for this video! :)

Reply • Share ›

Karthik • a month ago

Thank you..!! Really helpful.

Reply • Share ›

Abdul Wahid • a month ago

Git ignore file missing it is very import if you need a complete kickstart git tutoiral, and bookmark this page for quick git refrences. But good job sir on this tutoiral

Reply • Share ›

pradeep • a month ago

Brilliant stuff!!!

Reply • Share ›

Rani • a month ago

awesome!!

Reply • Share ›

Fatemeh Barati · a month ago
best tutorial i've ever seen.thanks
Reply · Share ›

yousuf khan Ratul · a month ago
one of the best article on git.
1 Reply · Share ›

Leo Bastin · 2 months ago
Practical git in a nutshell. Awesome!
Reply · Share ›

Sachit Nayak · 2 months ago
loved it! I could quickly review all the basics ...
Reply · Share ›

Steve M · 2 months ago
Neatly summarised. Thanks Roger.
Reply · Share ›

Leonardo Echeverria · 2 months ago
Awesome tutorial... I love the colors :)
Reply · Share ›

ovedfs · 2 months ago
Excellent guide, really nice, thank you!
Reply · Share ›

Derrick Johnson · 2 months ago
Thanks this helps a lot. So say I have all of my project files on on Master branch and I want to push only the files needed to make my page run on gh-pages. How would I tell it to only push certain files to the new gh-pages branch? For example, when you use gulp or grunt it makes a folder that is your rendered site for previewing your site. How would I push only that site folder to gh-pages?
Reply · Share ›

Gary Jackson · 2 months ago
Haven't used git in 2 years and needed a quick refresher. Wish I had found this page sooner. Great job! I feel ...refreshed.
Reply · Share ›

TaMereSuceDesOurs · 2 months ago
git checkout -b test
Switched to a new branch 'test'

git checkout master
error: pathspec 'master' did not match any file(s) known to git.

git branch -d test
fatal: Couldn't look up commit object for HEAD

GIT SUCKS !!
Reply · Share ›

Zack Techman ↗ TaMereSuceDesOurs · a month ago
Until you get a detached HEAD
Reply · Share ›

SPIRITED ↗ TaMereSuceDesOurs · 2 months ago
... Probably felt that way when i just started, it will all make sense soon.
Reply · Share ›

TaMereSuceDesOurs · 2 months ago
new git implementation refuses to push
what is the alternative ?
Reply · Share ›

Libert Schmidt · 2 months ago
good job and thank you, easy to follow and helpful, and have learn more here than 500 videos that i watched.
1 · Reply · Share ›

Rahul C · 2 months ago
I've struggled learning to use git for the past 3 years. This took me 15 minutes to understand. Superb!
Reply · Share ›

Sam Wang · 2 months ago
recommended! really appreciate your effort.
Reply · Share ›

Igor Souza Indra · 2 months ago
Waited all my life for this readable git guide.
thanks!
Reply · Share ›

JL · 2 months ago
Thanks. I'm sure this will be very useful after I've used github a few times. Haven't yet seen a simple glossary. E.g. What are "origin" and "master"?
Reply · Share ›

Dave Stevens ➔ JL · 2 months ago
origin is your remote repository (i.e. the repo on Github if using github) *

master is your repository's main branch and should be the one true source of your code's production-ready version. If the end product you are creating is live to users, that code should be in the master branch. Alternative versions (ones you are working on, features you are adding and testing locally/on a staging or beta version etc) have their code changes in a separate branch by other names, and you merge that in to your master branch when it's ready to be live.

* "origin" is just the default name. You could call it something else, but for simplicity, the author just refers to the default.
1 · Reply · Share ›

Hollay-Horváth Zsombor ➔ JL · 2 months ago
In the nutshell, origin is local and master is server repository.
Reply · Share ›

Dave Stevens ➔ Hollay-Horváth Zsombor · 2 months ago
Unfortunately this is not correct. "origin" is the default name for the remote repository. Master is the main branch name.
Reply · Share ›

Hollay-Horváth Zsombor ➔ Dave Stevens · 2 months ago
Thx, correct.
Reply · Share ›

Alex Fortin · 2 months ago
Finally...a readable git guide
thumbs up
2 · Reply · Share ›

Michael Ding · 2 months ago
recommend++
Reply · Share ›

asdf · 2 months ago
+1
Reply · Share ›

Pedro Amaral · 3 months ago

best guide i've seen yet. thanks

▲ | ▼ • Reply • Share ›

David • 3 months ago

real nice

▲ | ▼ • Reply • Share ›

Mikhail Zoupas • 3 months ago

What a life saver !! Thanks thanks thanks

2 ▲ | ▼ • Reply • Share ›

Load more comments