git - the simple guide

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

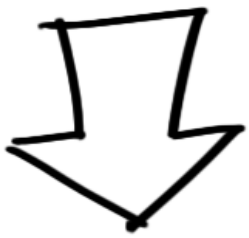
Tweet

by Roger Dudler

credits to @tfnico, @fhd and Namics

de in deutsch, español, français, indonesian, italiano, nederlands, polski, português, русский, 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

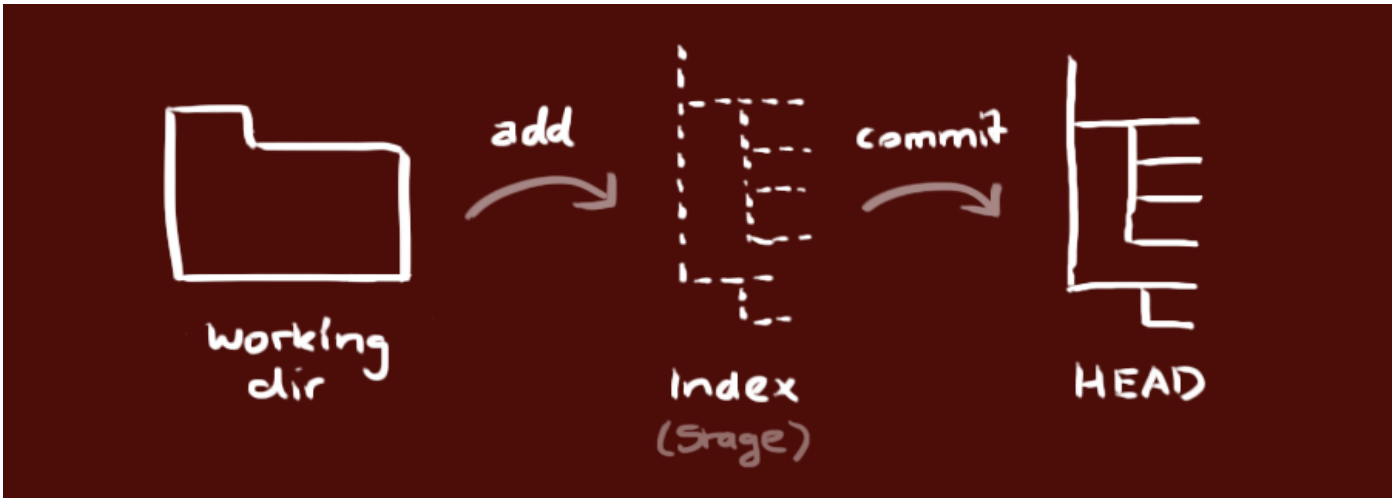
```
git clone /path/to/repository
```

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.

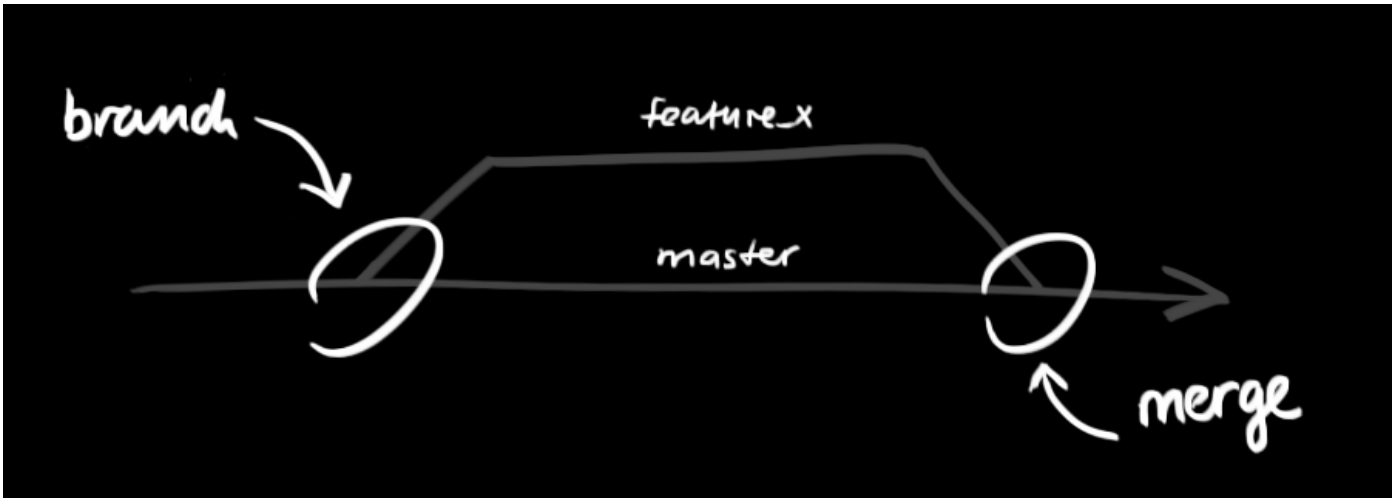
If you have not cloned an existing repository and want to connect your repository to a remote server, you need to add it with

```
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

34 Comments git - the simple guide Chris Hamon

Recommend 28 Tweet Share Sort by Newest

Join the discussion…

Samuel Torres • 6 days ago
If I am on master and perform a git pull from the master branch in my github repo, will I automatically have the most update version of the repo or is there a command I need to run after git pull?
• Reply • Share ›

Alex Samuel Torres • 2 hours ago
no, you need to do 'git fetch' for that
• Reply • Share ›

Sky Lee • a month ago

You can get the commit id by looking at the... ? Where?

• Reply • Share ›

Hollay-Horváth Zsombor ➚ Sky Lee • a month ago

looking at the... log. The command is `git log` - the next chapter is about that

• Reply • Share ›

Chloe LaPointe • a month ago

THANK YOU for this!
It really boils down the necessities for a quick reference with no fluff. Very practical.

1 • Reply • Share ›

Mike. • a month ago

Thank you very much for this. Really helped a lot. the Graphics were also perfect. Love from Germany.

1 • Reply • Share ›

Deepak Mecheri • 2 months ago

Adequate and I like it

1 • Reply • Share ›

Evils Wink • 2 months ago

Thanks for sharing. Very clear guide :).

1 • Reply • Share ›

Aislan Diego • 7 months ago

Dude, this thread is awesome. Great thx

3 • Reply • Share ›

Jerome • a year ago

BEST! shared...

2 • Reply • Share ›

trevahok • a year ago

this is dope ! I bookmarked it and shared the link with friends

2 • Reply • Share ›

Vinay Chitrakathi • a year ago

Nice tutorial

44 • Reply • Share ›

j3dy • a year ago

dear @rogerdudler

Come to steem we have cookies :D

Check out the repo on git, make an account on the blockchain(few ways or steemit.com) come to utopian(.io so) and make some extra hundred bucks every time you work and contribute to Open Source Projects :)

Thank me later :)

Also thank you very much for the guide, I will go through it and check out the extra links, looks interesting, thank you for sharing your insights :)

• Reply • Share ›

TeFa Zapata Naranjo • a year ago

Hey, Thanks for your big colaboration!

• Reply • Share ›

bmnnoboz • 2 years ago • edited

THANK YOU BOSSMAN! Quick and easy. Just what I needed

Reply • Share ›

Jayr Magave • 2 years ago

Thanks for sharing. Very usefull and objective.

Reply • Share ›

Santosh Kumar • 2 years ago

Loved the `git log --graph --oneline --decorate --all`.

Reply • Share ›

Zuhayer Tahir • 2 years ago

thanks for sharing

Reply • Share ›

Gautam Dogra • 2 years ago

you know what's so great on this site, other than the $$ info, is that there isn't even a hint of any negativity in any of the commenters either. So refreshing to see feedback that's kind and grateful and that's it! Bravo everyone!

Reply • Share ›

dentex • 2 years ago

That's a great resource... Thanks man.

Reply • Share ›

seymour1 • 2 years ago

This is lovely, thanks

Reply • Share ›

nocdib • 2 years ago

Beautifully succinct! Can't believe that I just saw this now but thanks!

Reply • Share ›

ndeans • 3 years ago

FINALLY someone figures out how to explain the whole thing without tripping over themselves!

1 Reply • Share ›

Sol • 3 years ago

Almost forgot. Use git log to find the version you want to go back to. (in our example 7e58100 (you don't need the whole long number they give)
)

Reply • Share ›

Sol • 3 years ago

git reset --hard 7e58100 or git reset --soft 7e58100. Just read about the difference. Soft is better for pointing your undo's. Make more branches as you go along so that you undo less often.

Reply • Share ›

ThePenguin • 3 years ago

What's the command for un-adding a file you mistakenly added but don't want to track? Please append it to your guide because I do this all the time.

1 Reply • Share ›

LeoTM • 3 years ago

Rolling in the deep my brother.

Reply • Share ›

Becoming_I • 3 years ago

You are absolutely amazing. This stuff is gold!!!

Reply • Share ›

CitizenX • 3 years ago

Git is such a massive pain in the ass. Thank you for trying to make it more straightforward.

Reply • Share ›

> **jazz** ➚ CitizenX • 2 years ago
>
> you could just use subversion....
>
> they support it fully now.
>
> Reply • Share ›

unx npx • 3 years ago

Thank you for a beautiful design, by the way, thanks for a good article. :)

2 Reply • Share ›

Ladna Meke • 3 years ago

nice

1 Reply • Share ›

anon • 4 years ago

i luv you

2 Reply • Share ›

Jeff • 4 years ago

Incredibly thoughtful and insightful. Used to learn Git. Thx so much!

4 Reply • Share ›