

# git - the simple guide

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

Tweet 4,747

by Roger Dudler

credits to @tfnico, @fhd and Namics

n deutsch, español, français, indonesian, italiano, nederlands, polski, português, pyck

မြန်မာ, 日本語, 中文, 한국어 Vietnamese

please report issues on github

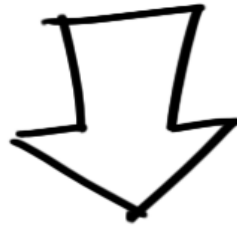


Are You a Front-End Developer?

by Roger Dudler, Author of the Git Simple Guide

Try Frontify

Now Free with  
Github Integration!



## 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

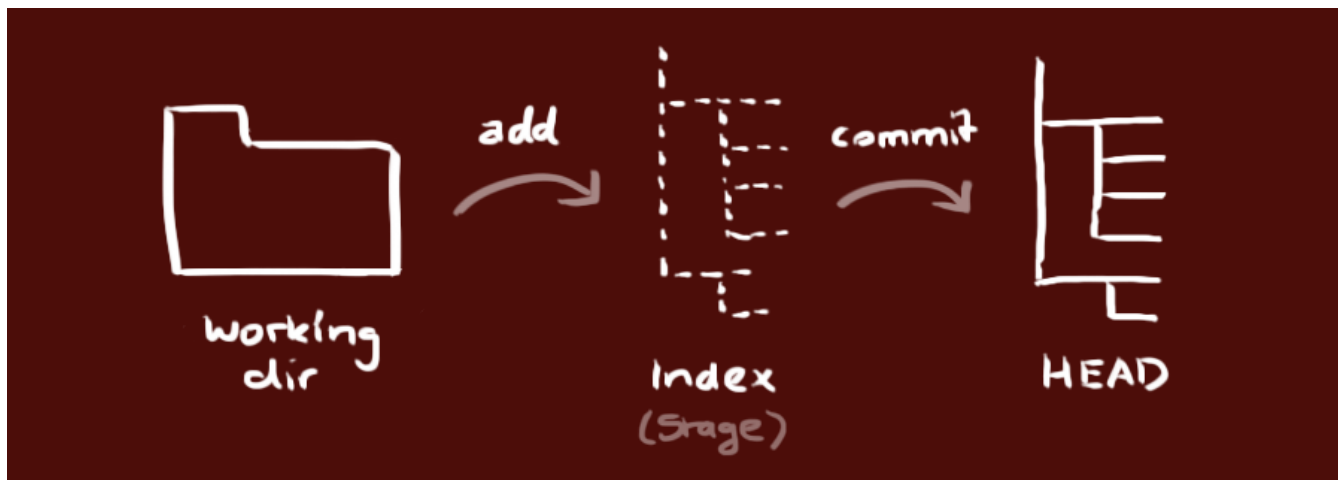
```
git clone /path/to/repository
```

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

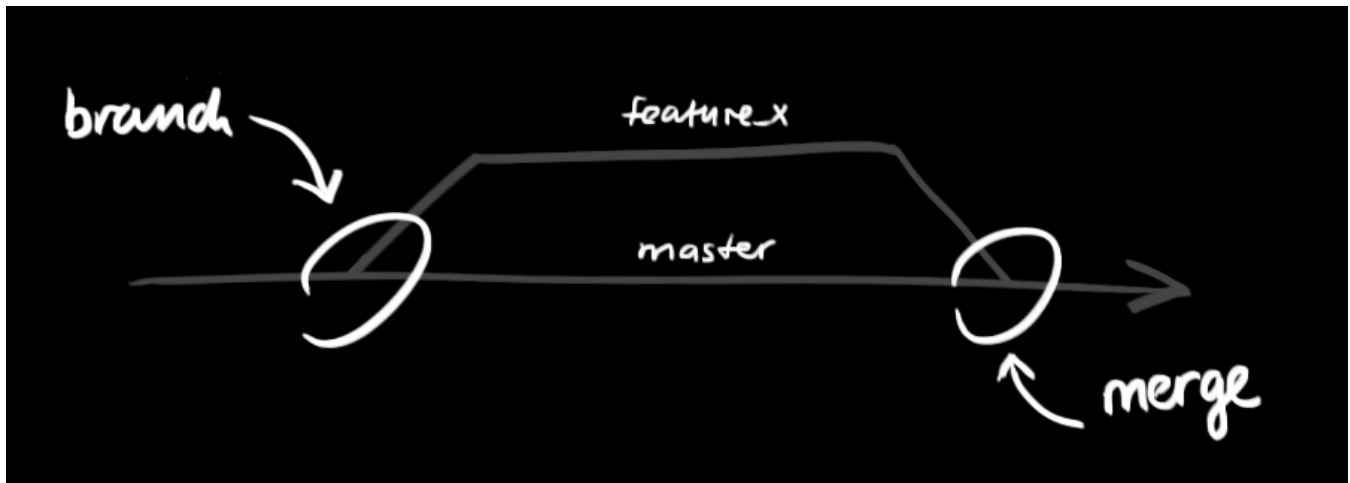
```
git remote add origin <server>
```

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

557 Comments

git - the simple guide

 Login ▾ Recommend 166 Share

Sort by Newest ▾



Join the discussion...

**Sachit Nayak** · 17 hours ago

loved it! I could quickly review all the basics ...

  · Reply · Share ›**Steve M** · a day ago

Neatly summarised. Thanks Roger.

  · Reply · Share ›**Leonardo Echeverria** · 4 days ago

Awesome tutorial... I love the colors :)

  · Reply · Share ›**ovedfs** · 6 days ago

Excellent guide, really nice, thank you!

  · Reply · Share ›**Derrick Johnson** · 9 days 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** · 9 days 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** · 11 days 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 ›**SPIRITED** ➔ TaMereSuceDesOurs · 10 days ago

... Probably felt that way when i just started, it will all make sense soon.

  · Reply · Share ›**TaMereSuceDesOurs** · 11 days ago

new git implementation refuses to push

what is the alternative ?

  · Reply · Share ›**Libert Schmidt** · 16 days ago

good job and thank you, easy to follow and helpful, and have learn more here than 500 videos that i watched.

