

GIT Tutorial

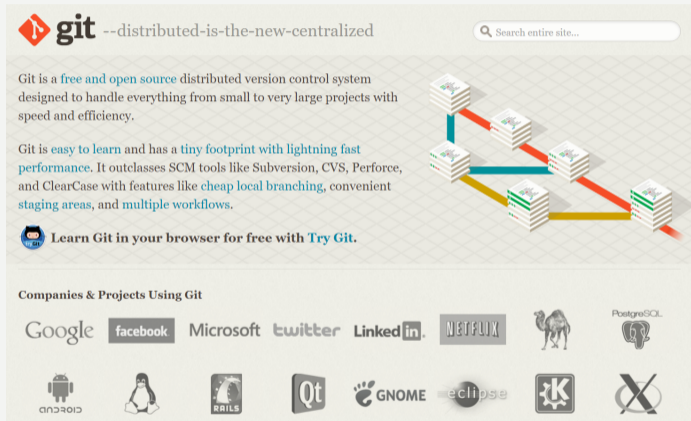
Martin Ritter

LMU Munich

Bachelor Einführung, 2019-05-02



Distributed Version Control created in 2005




The screenshot shows the Git website homepage. At the top left is the Git logo and the tagline "--distributed-is-the-new-centralized". To the right is a search bar. Below the logo is a paragraph describing Git as a free and open source distributed version control system. Further down, it mentions that Git is easy to learn and has a tiny footprint with lightning fast performance. At the bottom, there is a section titled "Companies & Projects Using Git" with logos for Google, Facebook, Microsoft, Twitter, LinkedIn, Netflix, Camel, PostgreSQL, Android, Linux, Rails, Qt, GNOME, Eclipse, and Jekyll.

git --distributed-is-the-new-centralized

Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.

 Learn Git in your browser for free with **Try Git**.

Companies & Projects Using Git

Google facebook Microsoft twitter LinkedIn NETFLIX camel PostgreSQL

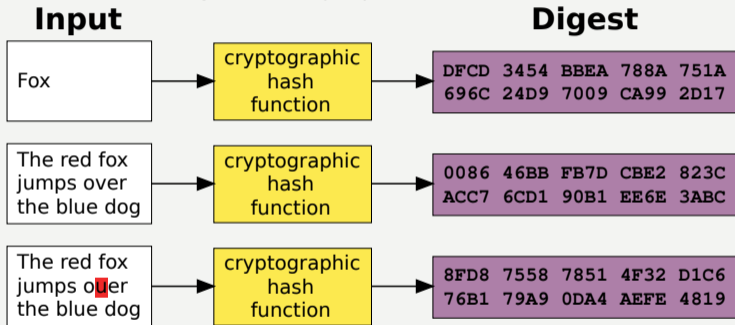
android linux rails Qt GNOME eclipse Jekyll

<https://git-scm.org>

- ▶ tracks changes to files over time
- ▶ does not need a central server
- ▶ every user keeps full history (aka clone)
- ▶ synchronized using *push* and *pull*



Create a checksum with fixed length from any input



- ▶ small changes to input → big changes to the hash value
- ▶ **one-way function**: practically impossible to calculate input from hash
- ▶ **collision resistance**: practically impossible to determine a different input with same hash



- ▶ Git tracks how your files evolve over time
- ▶ **You have to tell it what, when and why** to track
- ▶ Every such point is called a **commit** and contains
 - ▶ the date
 - ▶ the name and content of all the tracked files
 - ▶ the author of the commit
 - ▶ a message identifying why the commit was made
 - ▶ links to one or more previous commits

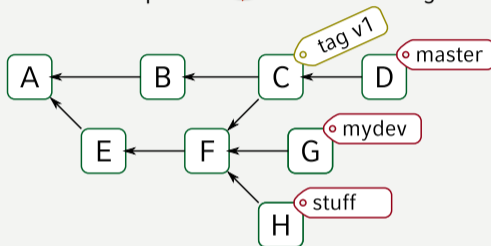
Commit ID

- ▶ SHA1 hash of the whole commit (all files + information)
- ▶ any change to any part of the commit would change the id
- ▶ usually abbreviated to 6-8 characters





A commit usually has one or more parents → All commits in git form a merkle tree



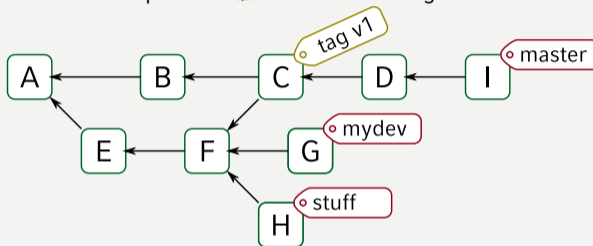
References

- ▶ a **branch** points to any commit and gets updated when adding a new commit to this branch
- ▶ a **tag** points to any commit, can be annotated.

→ Everything else is efficient synchronization and working with this tree



A commit usually has one or more parents → All commits in git form a merkle tree



References

- ▶ a **branch** points to any commit and gets updated when adding a new commit to this branch
- ▶ a **tag** points to any commit, can be annotated.

→ Everything else is efficient synchronization and working with this tree



Creating a git repository is easy:

```
git init
```

- ▶ configure git to track changes in the **current directory** and all subdirectories
- ▶ will create a `.git/` directory containing the whole history
- ▶ this is **not a backup**
- ▶ it only records changes in this directory

```
ritter@proton: ~  
[~]$ git clone ssh://git@stash.desy.de:7999/~ritter/git_tutorial.git  
Cloning into 'git_tutorial'...  
remote: Counting objects: 21, done.  
remote: Compressing objects: 100% (14/14), done.  
remote: Total 21 (delta 9), reused 0 (delta 0)  
Receiving objects: 100% (21/21), done.  
Resolving deltas: 100% (9/9), done.  
[~]$ cd git_tutorial/  
[~/git_tutorial]$ ls  
README.md  
[~/git_tutorial]$ git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
nothing to commit, working tree clean  
[~/git_tutorial]$ echo "print('Hello World')" > hello_world.py  
[~/git_tutorial]$ git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
      hello_world.py  
  
nothing added to commit but untracked files present (use "git add" to track)  
[~/git_tutorial]$ █
```



Working with existing repositories:

```
git clone other_git_repository
```

- ▶ other_git_repository can be just another directory, a http url or a ssh url
- ▶ will make a full local copy of the remote repository
- ▶ it will contain the **complete history**
- ▶ git pull will update the local copy
- ▶ git push will try to add your changes to the remote repository

```
ritter@proton: ~  
[~]$ git clone ssh://git@stash.desy.de:7999/~ritter/git_tutorial.git  
Cloning into 'git_tutorial'...  
remote: Counting objects: 21, done.  
remote: Compressing objects: 100% (14/14), done.  
remote: Total 21 (delta 9), reused 0 (delta 0)  
Receiving objects: 100% (21/21), done.  
Resolving deltas: 100% (9/9), done.  
[~]$ cd git_tutorial/  
[~/git_tutorial]$ ls  
README.md  
[~/git_tutorial]$ git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
nothing to commit, working tree clean  
[~/git_tutorial]$ echo "print('Hello World')" > hello_world.py  
[~/git_tutorial]$ git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
    hello_world.py  
  
nothing added to commit but untracked files present (use "git add" to track)  
[~/git_tutorial]$ █
```




Before git tracks anything we have to **add** files

```
git add filename
```

Git has a separate staging area

- ▶ record of all modified files marked for commit
- ▶ git add is needed for **each** commit.
- ▶ alternatively, git commit -a commits all changes to tracked files
- ▶ git status will show files added to the index, modified files, unknown files

```
ritter@proton: ~  
[~/git_tutorial]$ git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Changes to be committed:  
  (use "git reset HEAD <file>..." to unstage)  
  
    modified:   README.md  
    new file:   functions.py  
  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git checkout -- <file>..." to discard changes in working directory)  
  
    modified:   README.md  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
    hello_world.py  
  
[~/git_tutorial]$
```



You can always inspect changes

- ▶ to see the unstaged changes

```
git diff
```

- ▶ or to see the staged changes

```
git diff --staged
```

- ▶ pro tip: for plain text or latex try

```
git diff --word-diff
```

```
ritter@proton: ~  
[~/git_tutorial]$ git diff  
diff --git a/README.md b/README.md  
index 02181d1..2a98f4d 100644  
--- a/README.md  
+++ b/README.md  
@@ -5,8 +5,11 @@ You can use [Markdown](https://confluence.atlassian.com/display/STASH038/Markdow  
to make a nice readme file for other users which includes  
  
1. ordered lists  
-2. unordered lists  
-3. links  
+2. unordered lists, for example  
+ - one item  
+ - another item  
+3. links, either just as https://myurl.com or as  
+ [link with text](https://myurl.com)  
4. tables  
  
and so forth. Markdown is designed to look as readable as possible also in tex  
t  
@@ -18,4 +21,4 @@ editors and for basic things it's very easy to use:  
| row2, col1 | row2, col2 |  
  
-TODO: add more stuff  
+I just added more stuff  
[~/git_tutorial]$
```



Pro Tip: `git add` can be run interactively

- ▶ to mark all files in current directory for commit if they are already known to git:

```
git add -u .
```

- ▶ to ask you for all changes if you want to stage them

```
git add -p .
```

```
ritter@proton: ~  
[~/git_tutorial]$ git add -p  
diff --git a/README.md b/README.md  
index 02181d1..2a98f4d 100644  
--- a/README.md  
+++ b/README.md  
@@ -5,8 +5,11 @@ You can use [Markdown](https://confluence.atlassian.com/display/STASH038/Markdow  
to make a nice readme file for other users which includes  
  
1. ordered lists  
-2. unordered lists  
-3. links  
+2. unordered lists, for example  
+ - one item  
+ - another item  
+3. links, either just as https://myurl.com or as  
+ [link with text](https://myurl.com)  
4. tables  
  
and so forth. Markdown is designed to look as readable as possible also in tex  
t  
Stage this hunk [y,n,q,a,d,j,J,g,/,e,?]? y  
@@ -18,4 +21,4 @@ editors and for basic things it's very easy to use:  
| row2, col1 | row2, col2 |  
  
-TODO: add more stuff  
+I just added more stuff  
Stage this hunk [y,n,q,a,d,K,g,/,e,?]? n
```



Once you are happy with the changes

- ▶ run `git commit` to open up an editor to write commit message

```
git commit
```

- ▶ the editor can be configured. E.g. to use VS Code:

```
git config --global core.editor "code --wait"
```

- ▶ alternatively you can supply the commit message on the command line

```
git commit -m "Here is my commit message"
```

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Please take the time to write descriptive commit messages

- ▶ it really helps others to understand the changes
- ▶ it will also help you to remember what you did



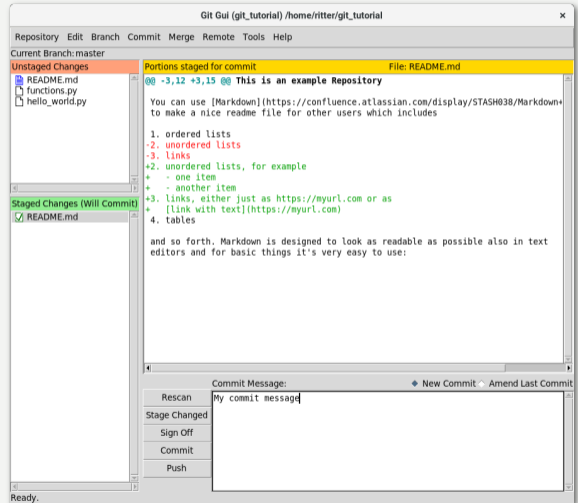
Alternative you can use `git gui`

- ▶ prepare commits in a graphical way
- ▶ preview the changes
- ▶ prepare commit message

Most IDEs or modern editors have GIT integration:

- ▶ almost no need to use git on command line
- ▶ can show changes directly in file or prepare commits

➔ well worth to check out





1. Create a local git repository on your computer
2. Add one or two files to it and create a few commits
3. Familiarize yourself with the basic commands

Commands to try

- ▶ `git init` – to create a repository
- ▶ `git status` – to see the current status
- ▶ `git add` – to add files/changes
- ▶ `git reset` – to “unstage” changes
- ▶ `git commit` or `git gui` – to create commits
- ▶ `git diff` – to show differences to
- ▶ `git log` – to show commits
- ▶ `gitk` – to show commits and their changes in gui



Multiple popular web services offer advanced features for git

- ▶ web interface to git
- ▶ branch creation/deletion
- ▶ commenting and discussion of commits/changes
- ▶ pull/merge requests
- ▶ user repositories

➔ “Social Coding”, focus on collaboration.

- ▶ <https://gitlab.physik.lmu.de>
- ▶ <https://gitlab.lrz.de>





So far we learned “linear” history: working on one branch

- ▶ fine when working alone on a small project
- ▶ using multiple branches can be very powerful

Git allows to easily switch between different branches

- ▶ list all branches (and allow to modify/delete them)

```
git branch
```

- ▶ create a new branch and check switch to it

```
git checkout -b branchname
```

- ▶ switch to a branch

```
git checkout branchname
```




➔ remote branches are referenced with prefix: “origin/master” usually refers to the branch “master” on the remote repository “origin”

```
git remote show
```

Check out remote branch

If a branch with the same name exists on the server `git checkout branchname` does exactly what you'd expect: check out the server branch as a local branch

Branches can have an upstream branch

- ▶ `git push` will push current branch to its upstream branch
- ▶ if the branch has been created from remote it's usually setup automatically
- ▶ if you created a branch locally you can set it manually using `git push --set-upstream origin branchname`

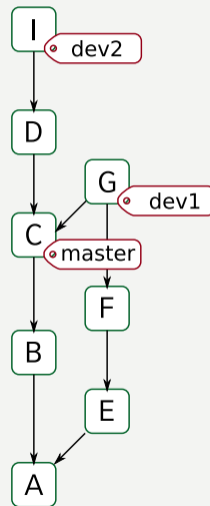


After some time you usually want to integrate the changes of one branch into another branch

- ▶ this is called “merge”
- ▶ `git merge branchname` will merge the changes from the given branch in the current branch

➔ There are three possible outcomes:

1. **fast forward:** the branch to merge just adds commits at the end. Just move the reference.
2. **normal merge:** there are some commit different but everything can be determined automatically. creates new “merge commit”
3. **merge with conflict:** both branches modified the same part of a file. Git cannot determine what to do. Will pause the merge and ask the user to resolve the conflicts.





If your merge as conflicts:

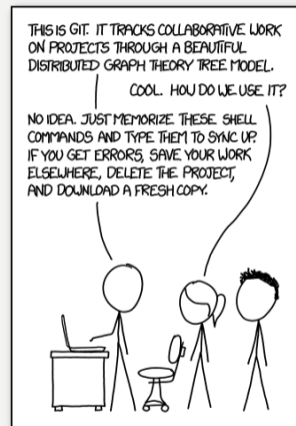
- ▶ git will pause the merge and leave conflict markers in the file
- ▶ resolve conflicts by hand and call `git add` on all conflicted files
- ▶ alternatively, `git mergetool` can be called to use a variety of graphical merging tools (meld, gvimdiff, kdiff, ...)
- ▶ `git commit` finishes the merge, `git merge --abort` aborts

```
<<<<<< HEAD
This is the version in our current
branch
|||||| merged common ancestors
This it was was in the last
common ancestor
=====
This is the version in the branch
we merge
>>>>>> branchname_to_be_merged
```



In the software world, git is everywhere

- ▶ there is a massive amount of documentation including a [online book](#)
- ▶ google knows all the answers
- ▶ don't be afraid but try to be a bit careful
- ▶ if you're unsure, make a backup of the directory before continuing



1

¹If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of "It's really pretty simple, just think of branches as..." and eventually you'll learn the commands that will fix everything.

Thank you
for your attention

