Version Control with GIT

Benjamin Roth

CIS LMU München

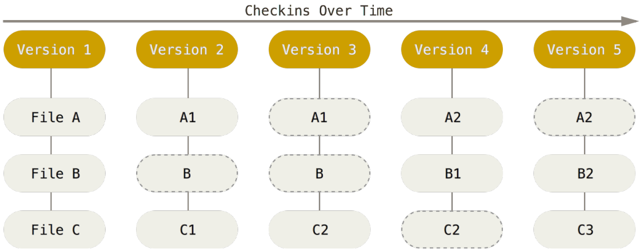
Version Control

- "Version control [...] is the management of changes to documents, computer programs, large web sites, and other collections of information." (Wikipedia)
- Track changes over time. ("What was the reason we changed this?")
- Option to undo and redo.
- Collaboration.
- Not just for managing source code, websites etc:
  also for writing theses, reports, archiving results from experiments, ...
- This class: basic git concepts using the command line.

Git

- Developed in 2005 by Linus Torvalds and other Linux kernel developers.
- Free software under GNU GPL.
- De facto standard for version control today.
- Distributed:
  Every Git working directory is a full-fledged repository
  - complete history and full version-tracking capabilities
  - independent of network access or a central server.
- Rapid branching and merging:
  A change will be merged more often than it is written.

Git Underlying Ideas

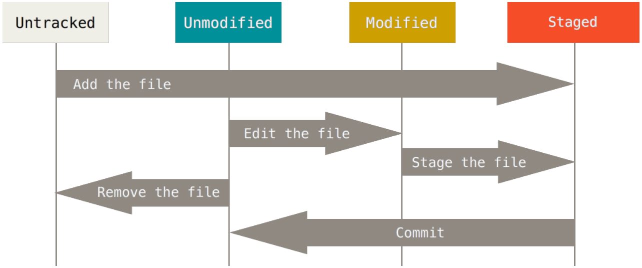


- Git thinks of data like a set of snapshots of a miniature filesystem.
- With every commit, Git takes a snapshot of your files and stores a reference to that snapshot.
- Efficiency: If files have not changed, Git stores just a link to the previous identical file it has already stored.

Git Underlying Ideas

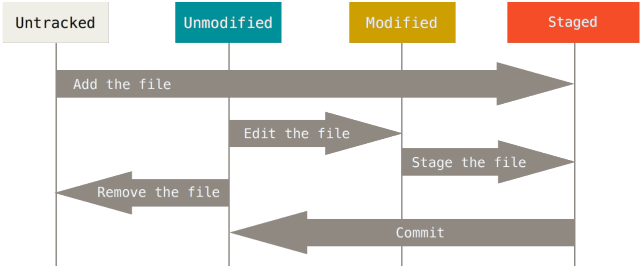
- Most operations in Git only need local files and resources to operate.
- Check-sums:
  - Everything is referred to by a checksum.
  - SHA-1 hashing: 24b9da6552252987aa493b52f8696cd6d3b00373
- Git generally only adds data: hard to do anything that is not undoable (e.g. permanently erase data).

Git: States



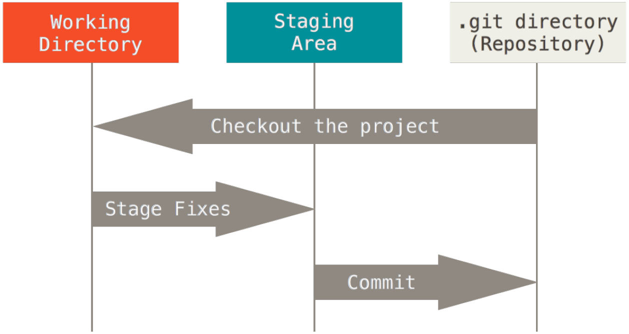
- Files can be in the following states:
  - **unmodified / committed**: data is safely stored in your local database.
  - **modified**: file changed but not committed to database yet.
  - **staged**: modified file is marked to go into your next commit.
  - (**untracked**: file not managed by git)

Recording Changes to the Repo



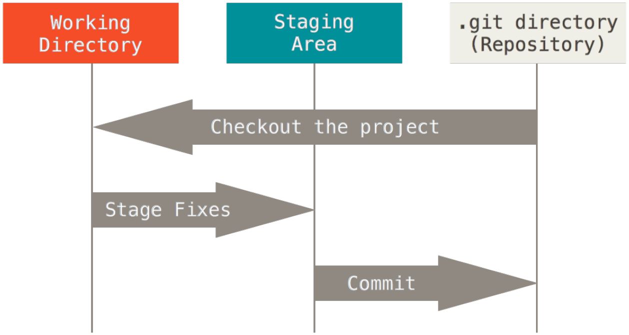
- Tracked files:
  - Files that were in the last snapshot ...
  - ... and newly added files.
  - Can be unmodified, modified, or staged.
- Untracked files: Anything else
- After first clone: All files will be tracked and unmodified.
- Adding untracked file: file will be staged.
- etc.

Git Places and Directories



- **Working directory**: single checkout of one version of the project. Placed on disk to be used and modified.
- **Staging area**: File (in .git directory) storing what will go into the next commit.
- **.git directory**: Contains object database with complete project history and meta-data.

Basic Git Workflow



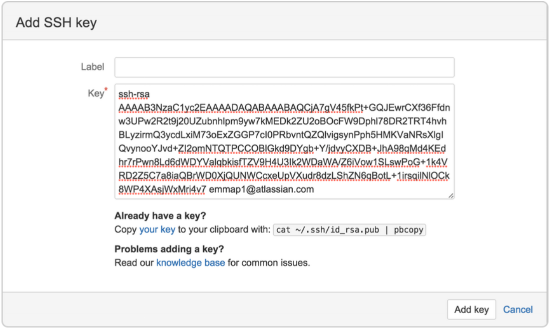
1. Modify files
   - in working directory
2. Stage the files
   - Add snapshot to staging area
3. Do a commit
   - Stores snapshot of staging area permanently in repository.

Configuring Git

- Setting up user name and email:
```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```
- Setting up different than default editor:
```
git config --global core.editor emacs
```
- Getting help:
```
git help config
git-scm.com
```

Setting up SSH for Git

- It can be convenient to connect with SSH to hosting services like GitHub and Bitbucket.
  - No need to enter password all the time.
- To access a remote over SSH:
  - Public ssh key (of your computer) is shared with server.
  - Your computer can hence verify itself (using the corresponding private key).
  - Just upload your public key (`~/.ssh/id_rsa.pub`) to the service.



```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCjA7gV45fkPt+GQJEwrCXf36Ffdn
w3UPw2R2t9j20UZubnhlpm9yw7kMEDk2ZU2oBOcFW9Dphl78DR2TRT4hvh
BLyzirmQ3ycdLxiM73oExZGGP7cl0PRbvntQZQlvigsynPph5HMKVaNRsXlgl
QvynooYJvd+ZI2omNTQTPCCOBIGkd9DYgb+Y/jdvyCXDB+JhA98qMd4KEd
hr7rPwn8Ld6dWDYValqbkisfTZV9H4U3Ik2WDaWA/Z6iVow1SLswPoG+1k4V
RD2Z5C7a8iaQBrWD0XjQUNWCcxeUpVXudr8dzLShZN6qBotL+1irsqilNlOCk
8WP4XAsjWxMri4v7 emmap1@atlassian.com
```

Already have a key?
Copy your key to your clipboard with: `cat ~/.ssh/id_rsa.pub | pbcopy`

Problems adding a key?
Read our knowledge base for common issues.

Using Git with a Server and SSH

- Check wether you already have a key pair:

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAklOUpkDHrfHY17SbrmTIp
GPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7X88XypNDvjYNb
mZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so
NrRFi9wrf+M7Q== beroth@mylaptop.local
```

- If not, create it:

```
$ ssh-keygen
```

- Copy-paste public key to Git hosting service (Github...), which will store it as an authorized key.

Getting a Git Repository

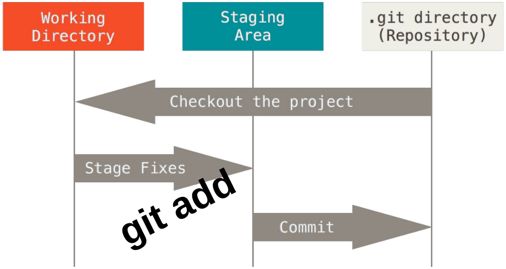
- Either **take an existing directory and import it** into Git or **clone** an existing git repository.
- Importing directory and commit:
  ```
  git init
  git add *.c
  git add LICENSE
  git commit -m 'initial project version'
  ```
- Getting a copy of an existing Git repository:
- With https:
  ```
  git clone https://github.com/username/projectname.git
  ```
- With ssh:
  ```
  git clone git@github.com:username/projectname.git
  ```

Checking the Status of Files

- After clone:
  Clean working directory, there are no tracked and modified files.
  ```
  $ git status
  nothing to commit, working directory clean
  ```
- After creation of a new (untracked) file:
  ```
  $ echo 'My Project' > README
  $ git status
  Untracked files:

  README
  ```

Staging files: `git add`



- Start tracking a file:
  `git add README`
- If a file is modified after staging it will be listed twice:
  - Once as staged: exactly as it was at the time of git add
  - Once as modified: with the new modifications

.gitignore

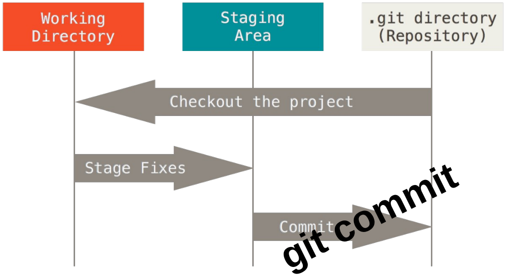
- Ignore a class of files
  - do not add
  - do not show as untracked
  - e.g. binaries, compiled code ...
- .gitignore file:

```
# no .a files
*.a

# ignore the TODO file in the current directory
/TODO

# ignore any build/ (sub)directory
build/
```

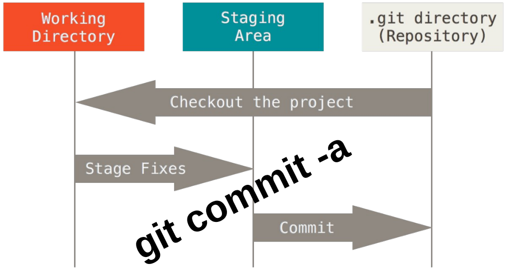
Committing Changes



- Only already staged changes will go into a commit:
  Changes done after `git add` will be ignored.
- `git commit`
  Will launch default editor: write a meaningful commit message!

-
  ```
  $ git commit -m "Story 182:  Fix benchmarks for speed"
  [master 463dc4f] Story 182:  Fix benchmarks for speed
  2 files changed, 2 insertions(+)
  create mode 100644 README
  ```

Add and Commit at the same Time



- Skip staging area (-a flag): automatically stage (add) every file that is already tracked, and commit.

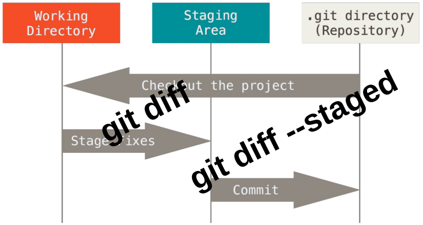
```
git commit -a -m 'added new benchmarks'
```

Removing and moving files

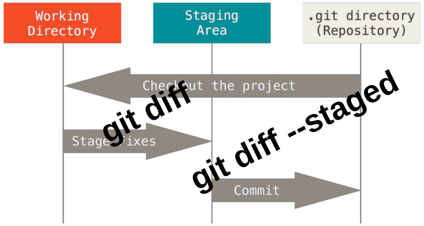
- Files are never deleted from history entirely.
- Remove file from working directory and stage its removal (usual case):
  ```
  git rm README.txt
  ```
  ⇒ Commit after that to make change permanent.
- Remove file from tracked files (but keep in working directory, e.g. if you forgot to add to `.gitignore`):
  ```
  git rm --cached README.txt
  ```
- Rename / move file:
  ```
  git mv README.md README
  ```
- Equivalent to:
  ```
  mv README.md README
  git rm README.md
  git add README
  ```

Git diff

- Detailed overview of changes in file content, line-by-line (instead of file-by-file).
- `git diff`
  What have you changed but not yet staged?
- `git diff --staged`
  What have you staged that you are about to commit?



Git diff: Example



- Stage the README file (created previously):

```
$ git add README
```

- Add to the file:

```
$ echo 'More text.' >> README
```

- Compare new changes to staged version:

```
$ git diff
@@ -1 +1,2 @@
 My project
+More text.
```

Viewing the Commit History: git log

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700
```

Viewing the Commit History: Options

- Show differences for each commit: `git log -p`
- Show last two commits only: `git log -2`
- Show overview statistics: `git log --stat`
- Only hashes and commit messages: `git log --pretty=oneline`
- Only last two weeks: `git log --since=2.weeks`
- Many more options and combinations:

```
$ git log --pretty="%h - %s" --author=beroth \
--since="2015-10-01" --before="2015-11-01"

5610e3b - Fix testcase failure when extended attributes ar
acd3b9e - Enhance hold_lock_file_for_{update,append}() API
f563754 - demonstrate breakage of detached checkout with s
```

Undoing Things

- You already committed, but forgot to add a file, and/or want to amend the commit message:

  ```
  $ git commit -m 'initial commit'
  $ git add forgotten_file
  $ git commit --amend
  ```

- You want to unstage a file that you have just staged:

  ```
  $ git add *
  $ git reset HEAD README.txt
  ```

- Unmodifying a file. You want to revert back to the version of the file that was last committed:

  ```
  git checkout -- CONTRIBUTING.md
  ```

  CAREFUL: All uncommitted modifications are lost irrecoverably!

Remote Repositories

- Several remote repositories possible: Pushing and pulling from them vital for collaboration.
- If project was initially cloned, one remote repository already exists called `origin`:

```
$ git clone https://github.com/beroth/ticgit
$ git remote -v
origin https://github.com/beroth/ticgit (fetch)
origin https://github.com/beroth/ticgit (push)
```

Remote Repositories

- You can add more remote repositories:

```
$ git remote add mynewremote https://github.com/schacon/ticgit
```

- Fetch all the information from `mynewremote`:

```
$ git fetch mynewremote
 * [new branch]      master     -> mynewremote/master
 * [new branch]      ticgit     -> mynewremote/ticgit
```

- The local project now contains a branch `mynewremote/master` that can be merged with the local master branch (more on branching later).

`git pull` and `git push`

- `git pull`: fetch and merge
  → All staged changes must be committed before merge can happen.
- `git push`: push your changes to remote
  → If the remote had changed in the meantime, you need to pull (and merge) again.
- One can specify the remote and branch, if defaults (e.g. `origin` and `master`) are not appropriate.
  ```
  git pull <remotename> <branchname>
  git push <remotename> <branchname>
  ```
- show information about remote repository:
  ```
  git remote show
  ```

A Typical Workflow

1. Get current project state from remote
   - Initially: Clone project.

     `git clone git@github.com:username/projectname.git`
   - Later: Fetch and merge changes from remote.

     → Possibly resolve conflicts.

     `git pull`
2. Make changes
   - Add a File.

     `git add CHANGES.txt`
   - Edit a File.

     `vi README.txt`
3. Add and merge the changes locally.

   `git commit -a -m "Summary of changes."`
4. Fetch and merge changes from remote.

   → Possibly resolve conflicts.

   `git pull`
5. Push changes to remote.

   `git push`

Next Lecture

- Branches
- Merging
- Resolving conflicts
- GitHub
- Pull requests

Questions?