Version Control

General

Common Problems

- Frequent changes
- Multiple contributors
- “Experiments”
- Backups

Version Control Solutions

- Project states, reverts
- Project history
- Branches, merges
- Remotes

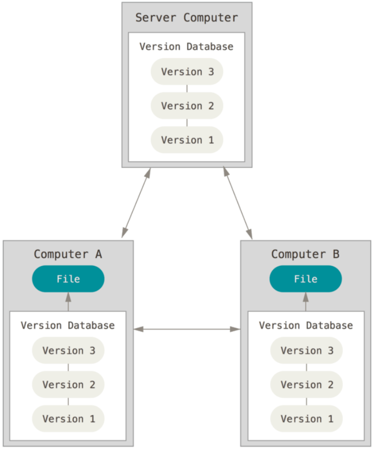
Types

- Centralized (SVN, . . .) — bad
- Distributed (Git, Mercurial, . . .) — good

Git can be easily used in centralized scheme (keeping decentralized advantages)

Git

Distributed Version Control



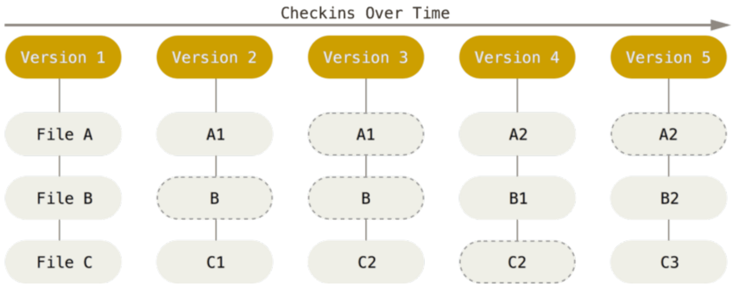
Scott Chacon and Ben Straub. *Pro git*. Apress, 2014.

Basics

Git tracks. . .

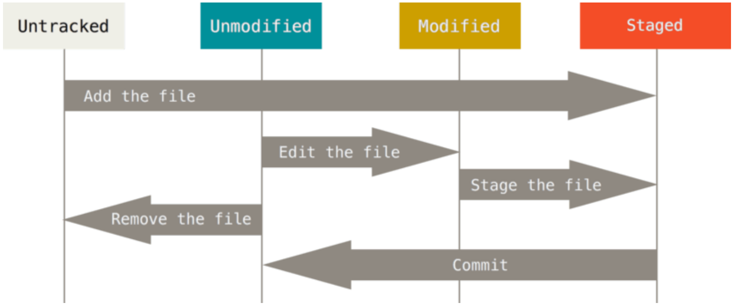
– a history committed changes
– on added files
– in branches.

Snapshots



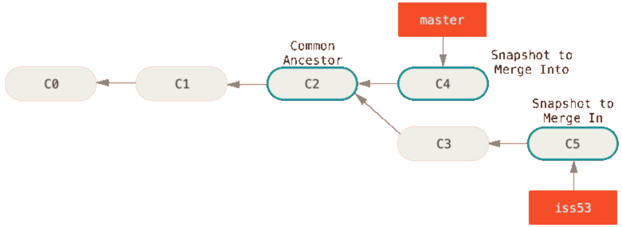
Scott Chacon and Ben Straub. *Pro git*. Apress, 2014.

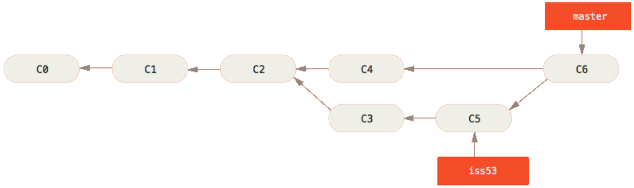
Lifecycle



Scott Chacon and Ben Straub. *Pro git*. Apress, 2014.

Branches





Scott Chacon and Ben Straub. *Pro git*. Apress, 2014.

Common Setup (“Centralized”)

- Bare repository on remote server
- Clients clone repository, work on local copy
- Clients pull from / push to remote

Common Usage

```
git clone user@host:/path/to/project project

git pull
git log

git add file_to_be_added_or_staged
git commit -m "Commit message."
git push
```

Common Usage (cont.)

```
git diff commit_hash
git checkout commit_hash

git branch new_branch
git checkout new_branch

git checkout master
git merge new_branch
```

Ignoring Files

Put files to be ignored (`.cache`, `__pycache__`) into the `.gitignore` file (project root).

Installation

- Linux (ZBH) : Installed
- Mac OS: XCode
- Windows: `https://gitforwindows.org/` (brings it's own bash)

Configuration

Add following lines to `.ssh` or `.ssh/config`:

```
Host bari
User username
Port 7373
Hostname bari.zbh.uni-hamburg.de
```

Clone project repository:

```
git clone bari:path/to/project
```

Alternatively: Use some service (github, bitbucket, ...)

Recommendations

Project Setup

```
repository
  doc
  org
  project
    gui
      __init__.py
      ..
    objective_functions
       __init__.py
       ..
    optimization
       __init__.py
       ..
    visualization
       __init__.py
       ..
  test
    __init__.py
    ..
```

Practices

- Use extensively (ask & read docs)
  - Single commits for discrete steps
  - Proper commit messages (imperative)
- Use feature branches, keep `master` clean
  - Separate branch for each context
  - Sub-branches for implementations
  - Merge completed implementations / clean states into master
  - Use `git squash` for cleaner history

  $\Rightarrow$ `master` contains only (passing) unit tested code