TECHNISCHE UNIVERSITÄT DRESDEN

Zentrum für Informationsdienste und Hochleistungsrechnen – TU Dresden

Introduction into Git

Sebastian Oeste

May 16, 2017

FAL 245
Chemnitzer Strasse 46b
01069 Dresden

Telefon: +49 0351 - 463 32405
E-Mail: sebastian.oeste@tu-dresden.de

ZIH
Center for Information Services & High Performance Computing

Table of Contents

What is Git?

Git is a decentralized VCS (Version Control System).

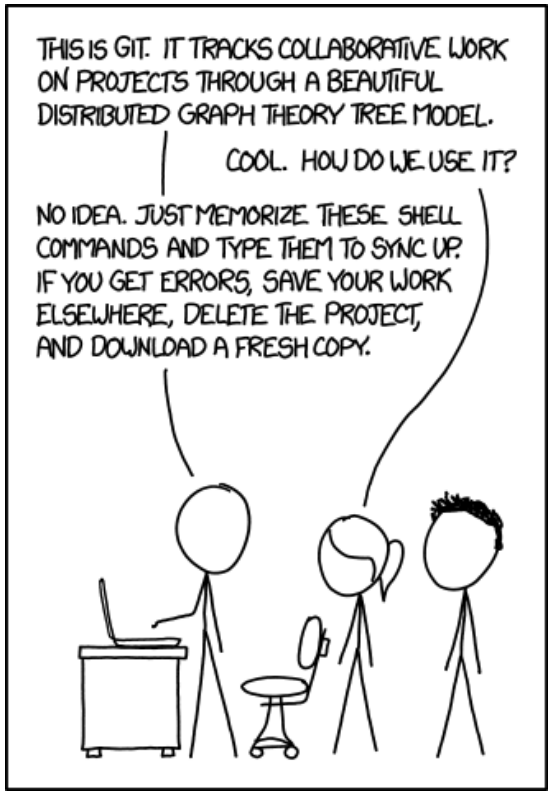
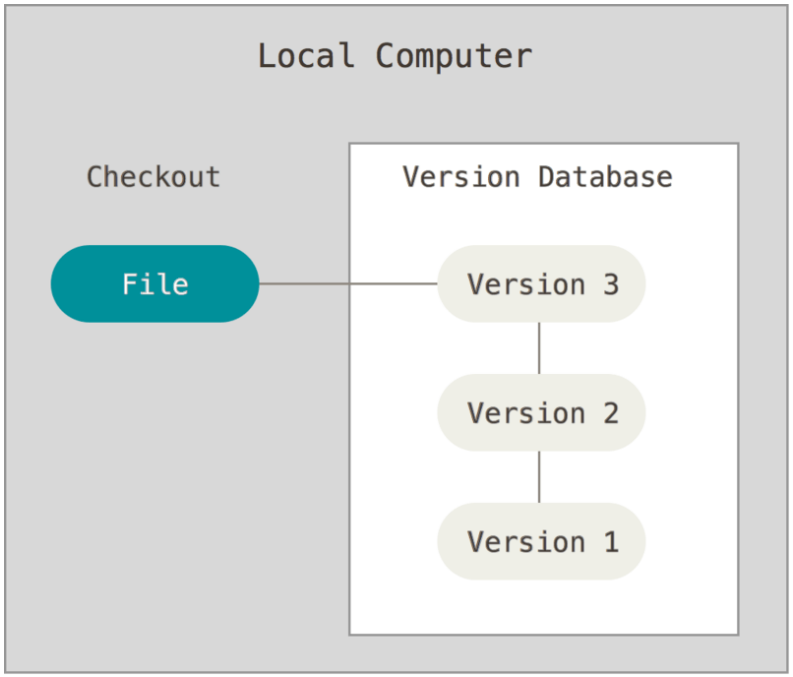


Figure: *https://xkcd.com/1597/*

Version Control Systems

Local Version Control System

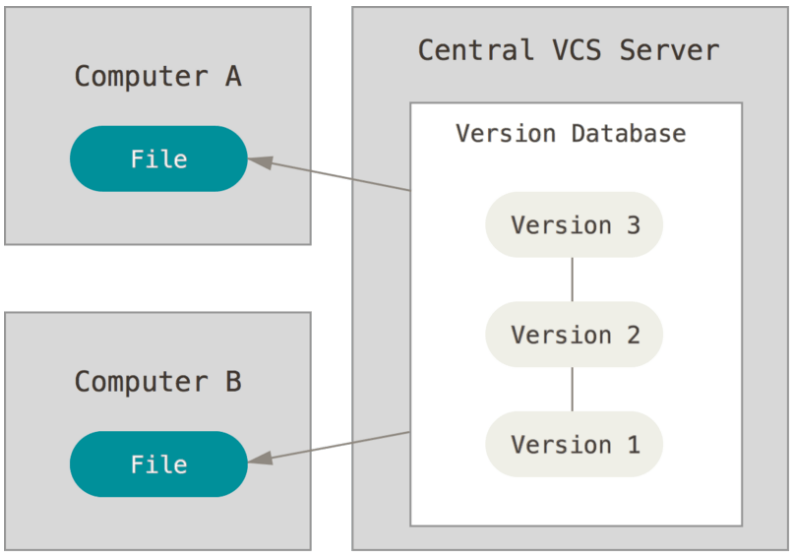
- store different versions of a file
- simple but error prone
- *rcs* - Revsion control system



Version Control System

Centralized Version Control Systems

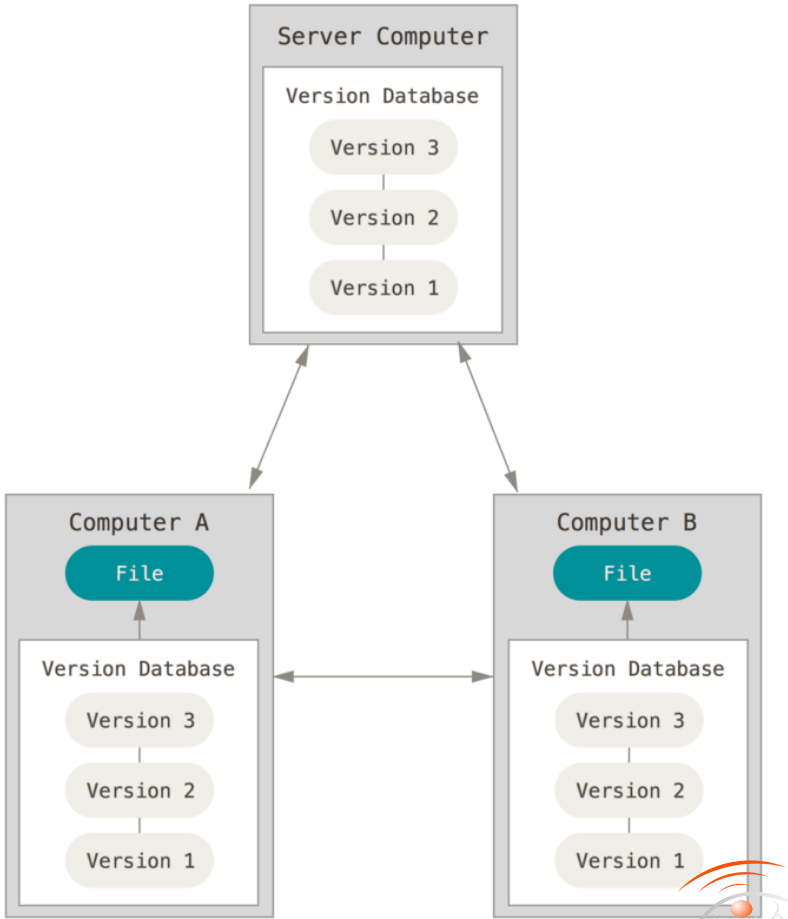
- single server that conatins all the versioned files
- clients check out files from that central place
- single point of failure
- *Subversion*, *CVS*



Version Control System

Distributed Version Control System

- clients fully mirror the repository
- serveral remote repositories possible
- *Git*, *Mercurial*, *Barzaar*



Installing git

Debian:

```
1     apt install git
```

Arch Linux:

```
1     pacman -S git
```

Fedora:

```
1     yum install git
```

For Linux: `http://git-scm.com/download/linux`

For Mac: `http://git-scm.com/download/mac`

For Windows: `http://git-scm.com/download/win`

How it NOT works

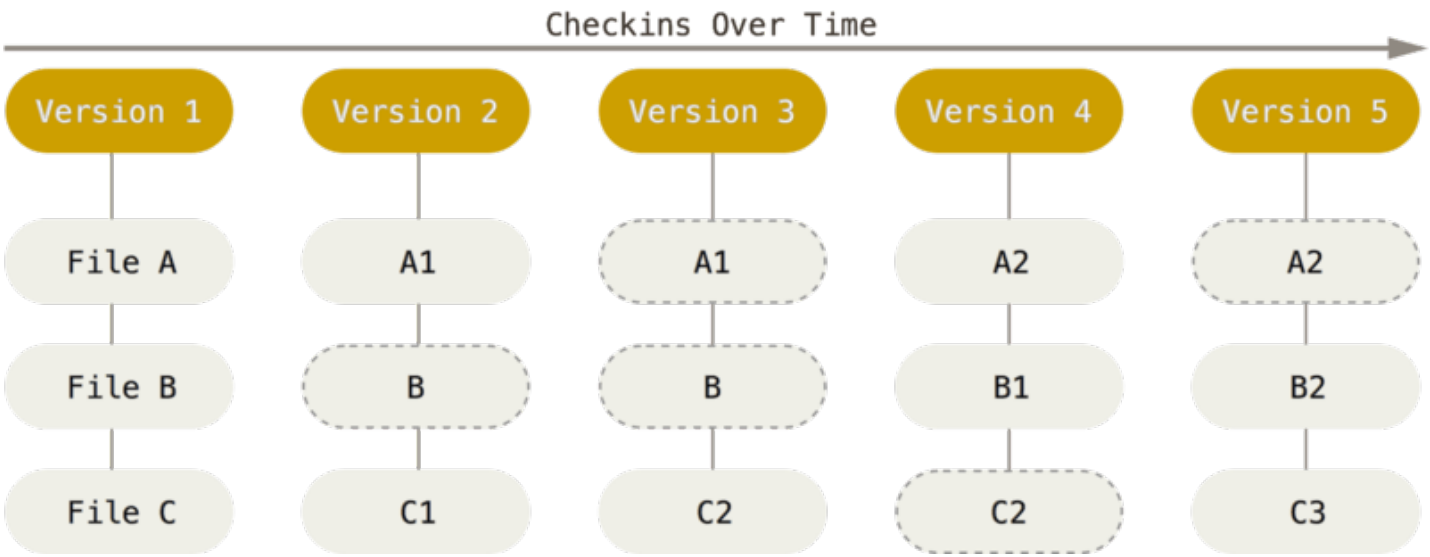


Figure: storing snapshots over time

How it works

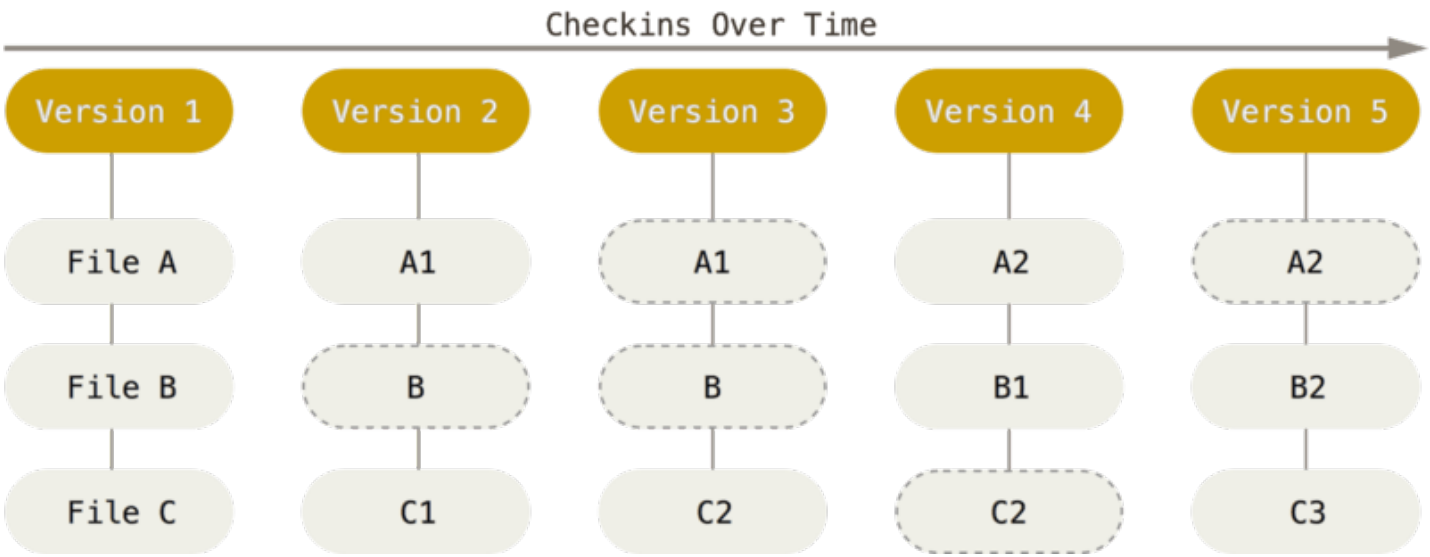
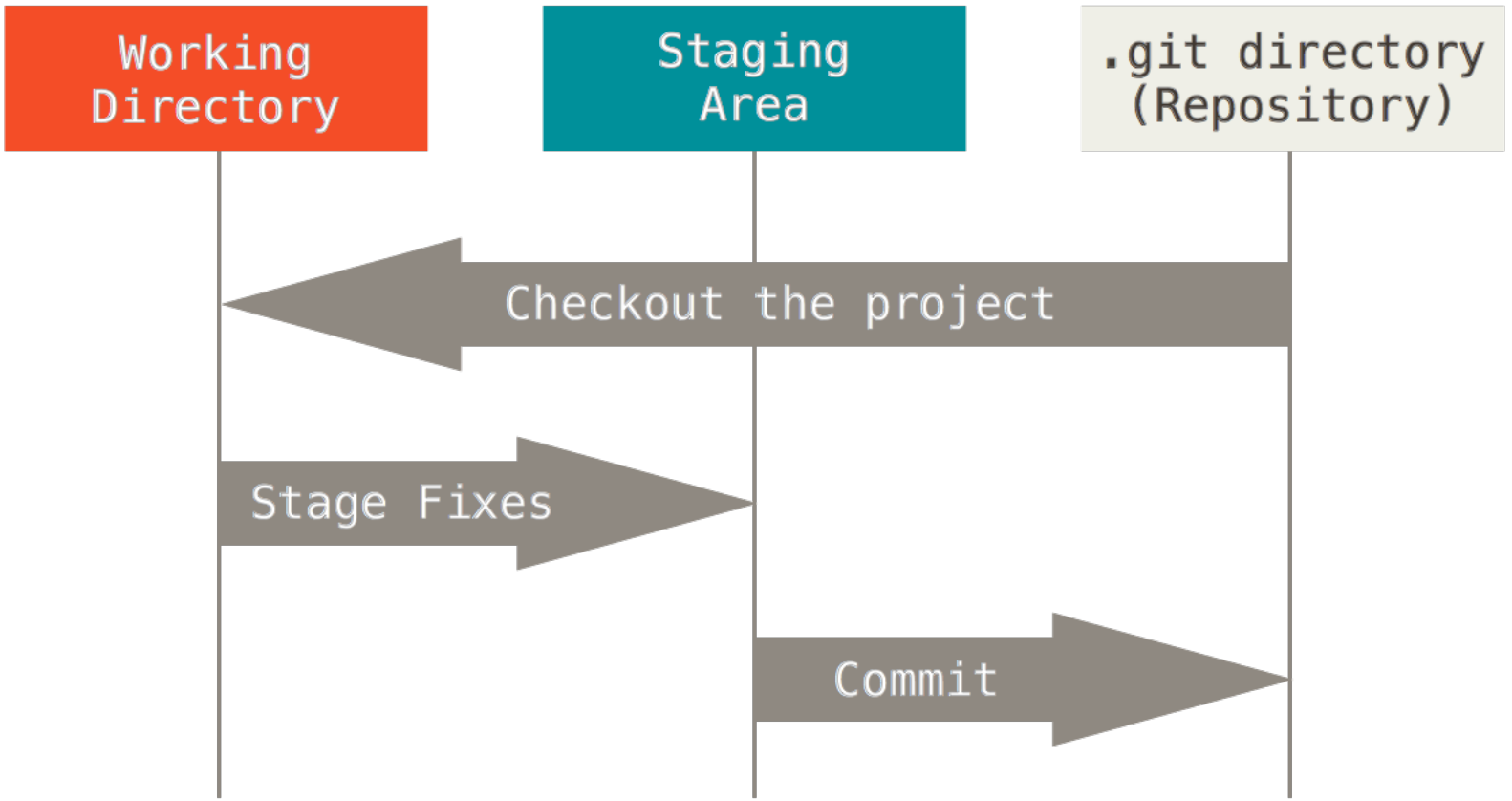


Figure: storing snapshots over time

What is a git repository?



Basic Workflow

1. Modify files in your working directory.
2. Stage the files, adding snapshots of them to your staging area.
3. Commit, stores the snapshot in the staging area permanently in your git directory.
4. Push your commits to an remote repository on a server.

Configure Git

Place where the git configuration can live:

- **/etc/gitconfig** system wide configuration *--system*
- **/.gitconfig** or **/.config/gitconfig** user wide configuration *--global*
- **.git/config** repository specific configuration

Configure Git

Show your configuration:

```
1          git config --list
```

Show specific configuration value:

```
1          git config user.name
```

Define an alias:

```
1          git config alias.st=git status
```

Enable highlighting:

```
1          git config --global color.ui=always
```

Setup Your Environment

Three essentail configuration values you should have set.

Your name:

```
1          git config --global user.name "Max Mustermann"
```

Your email address:

```
1          git config --global user.mail "max@example.org"
```

Your editor:

```
1          git config --global core.editor "vim"
```

Lets Start...

Start from scratch:

```
1 mkdir my_new_project
2 cd my_new_project
3 git init
```

Get a local copy of a repository that already exist.

```
1 git clone https://github.com/blastmaster/ta-git_intro.git
2 git clone -b <branchname> <GIT_URL>
```

Follow the changes

What is the status of your local repo?

```
1          git status
```

What happens so far?

```
1          git log
```

What has changed?

```
1          git diff [--staged]
```

Who has changed?

```
1          git blame
```

Ignoring Files

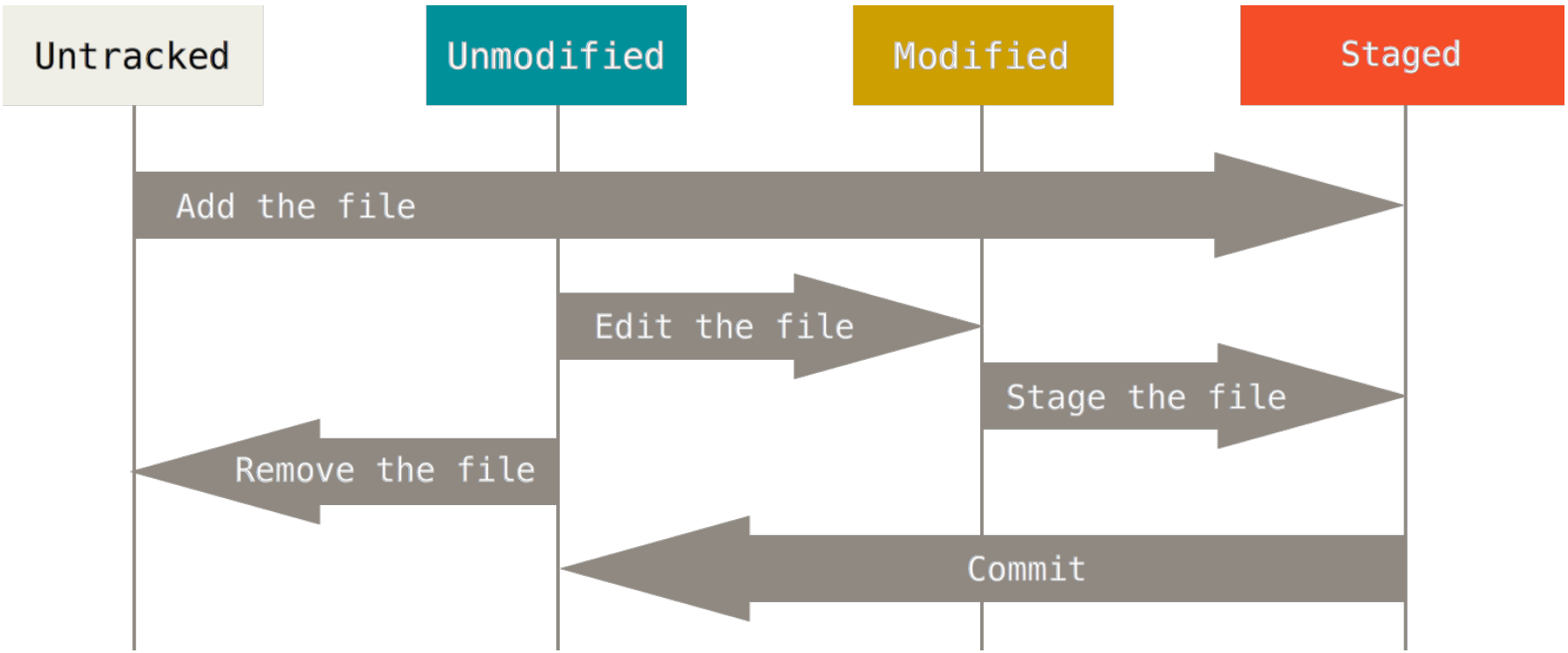
Ignore files that follow a specific pattern with a **.gitignore** *file*
.gitignore rules:

- Black lines or lines starting with # are ignored.
- Standard glob pattern work.
- You can start patterns with a forward slash (/) to avoid recusivity.
- You can negate a pattern by starting it with an exclamation point (!).

Example:
`https://github.com/github/gitignore`

Recording Changes

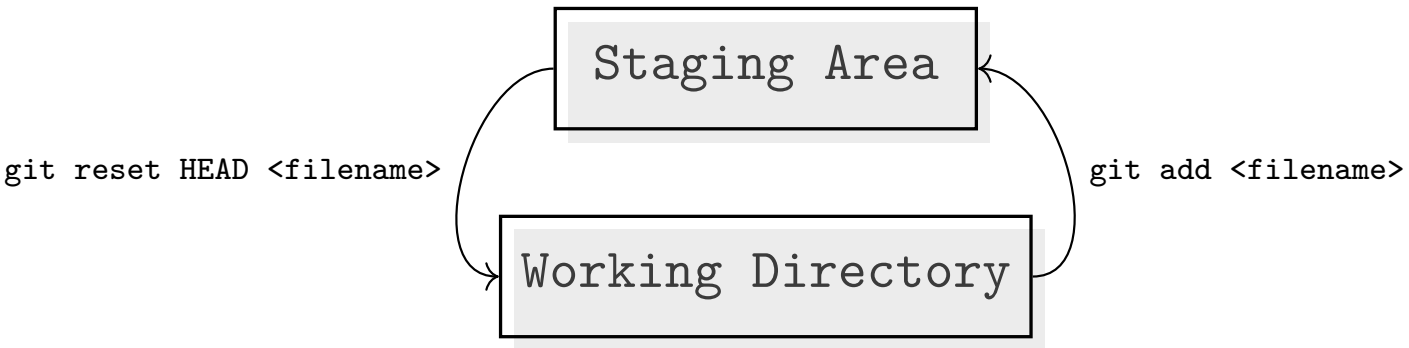
Each file in your working directory can be in one of two states: tracked or untracked.



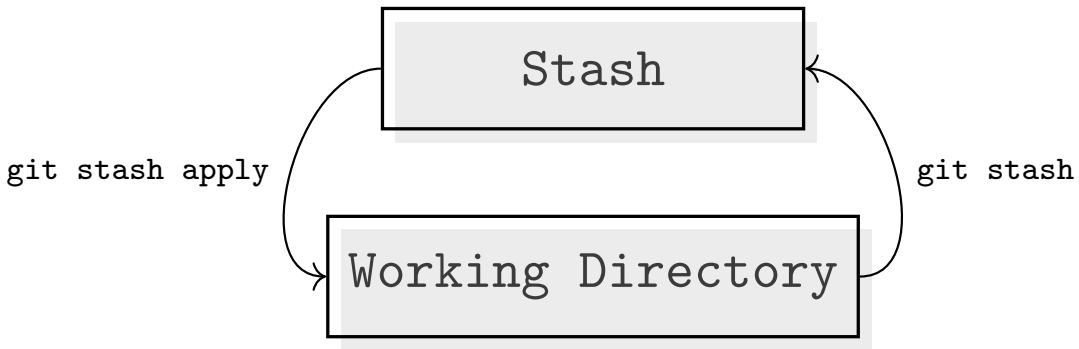
Git Terminology

Repository on the internet or network.	remote
Local repository that contains complete history.	Local Repository
Snapshot of the working tree for next commit.	Staging Area
A place to hide modification if you need a clean workspace.	Stash
The direcotries and files on your filesystem.	Working Directory

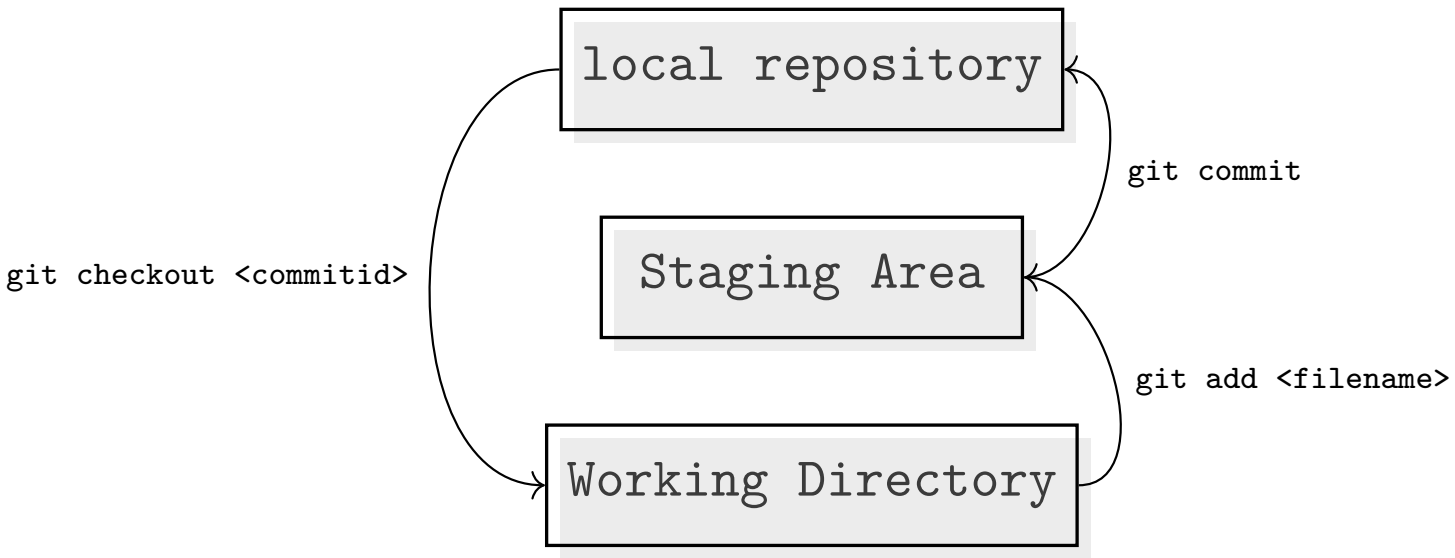
Staging files



Using the Stash



Commit Changes



Undoing things

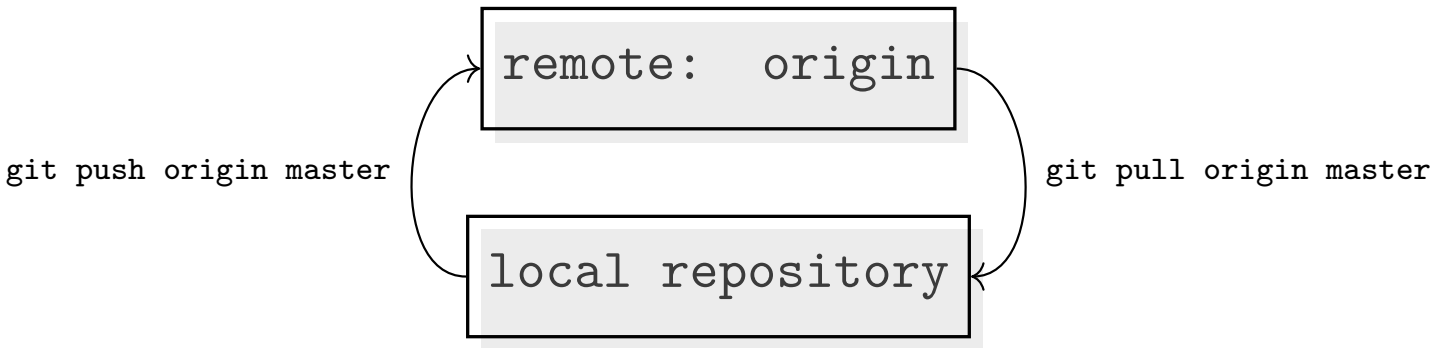
Unstating a Staged File:

```
1          git reset HEAD <file>
```

Unmodifying a Modified File:

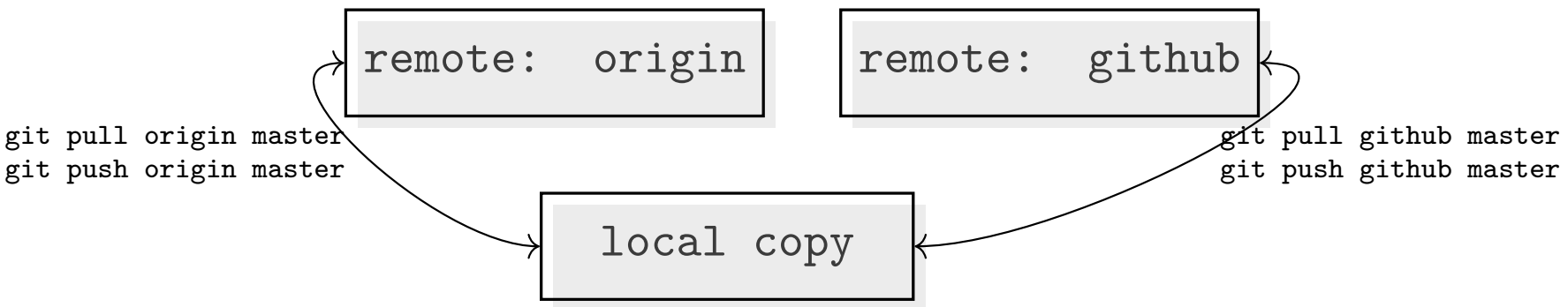
```
1          git checkout -- <file>
```

Working Remotes



Working with Remotes

```
git remote add github git@github.com:username/repo.git
```



Working with Remotes

Showing your remotes:

```
1          git remote -v
```

Showing remote information:

```
1          git remote show <remote>
```

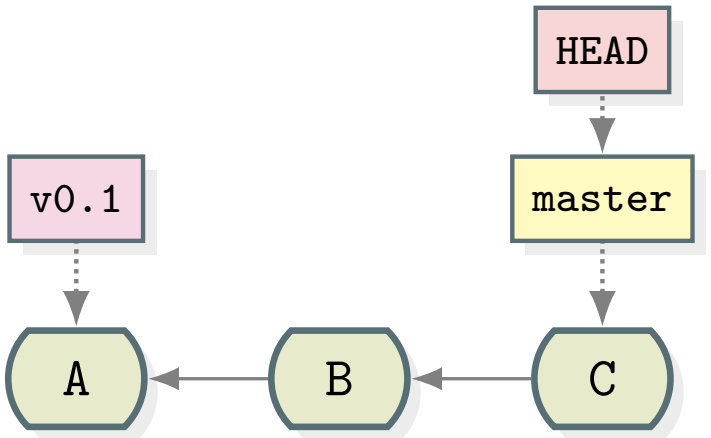
Rename a remote:

```
1          git remote rename <oldname> <newname>
```

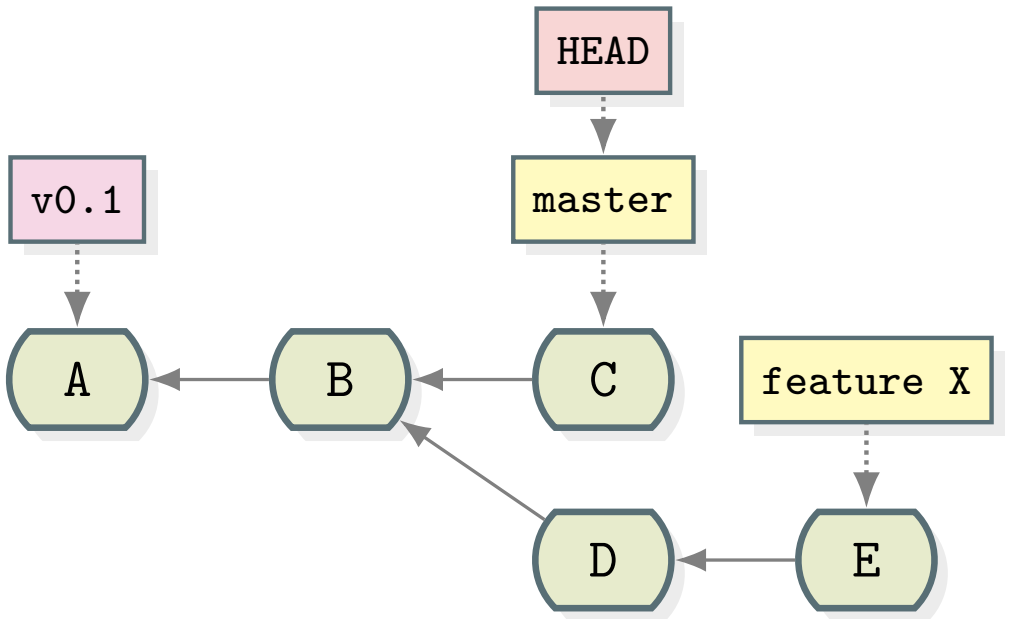
Remove a remote:

```
1          git remote rm <remote>
```

Branching



Branching



Branch Syntax

create new branch:

```
1               git branch <branchname>
```

```
1               git checkout -b <branchname>
```

delete branch:

```
1               git branch -d <branchname>
```

list local branches:

```
1               git branch
2               git branch --merged
3               git branch --no-merged
```

Merging

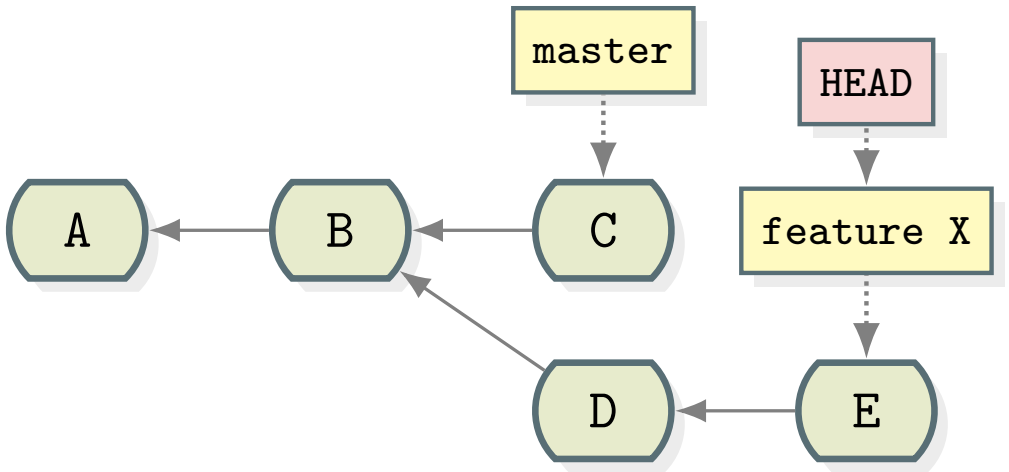


Figure: Before...

Merging

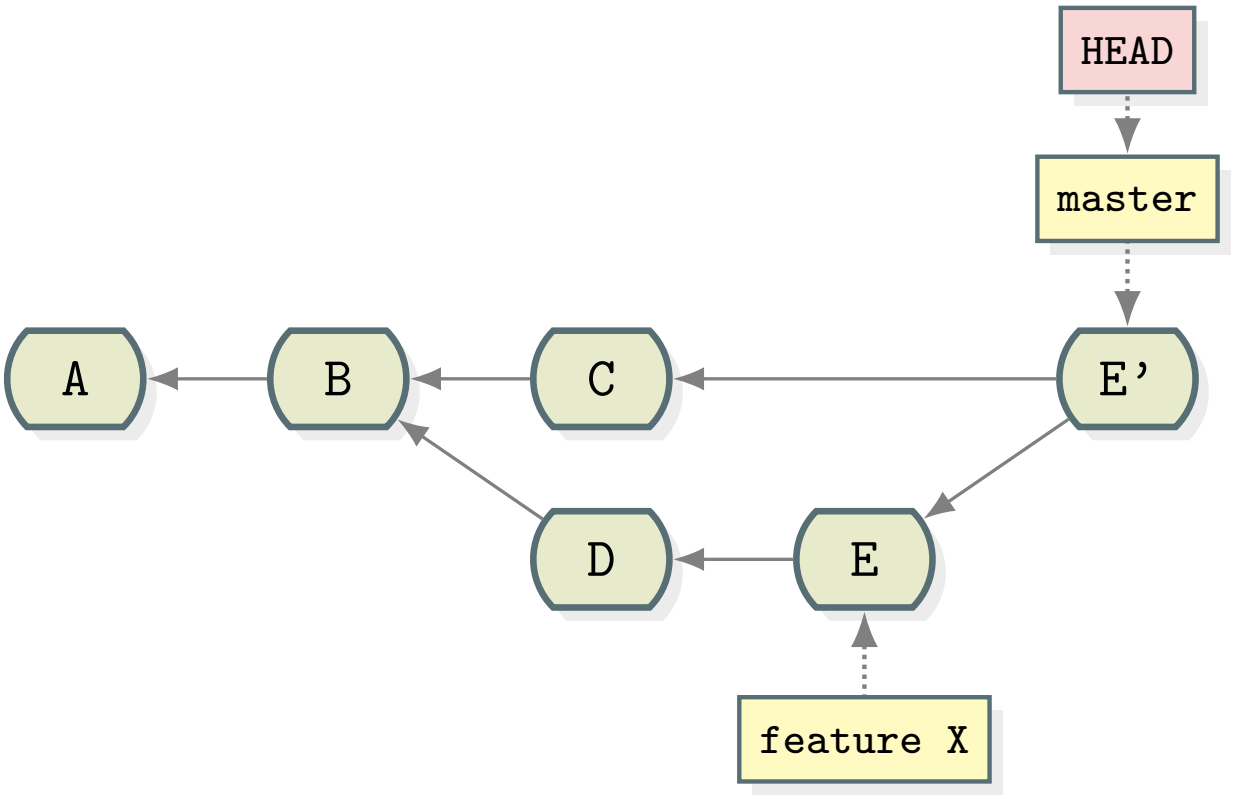
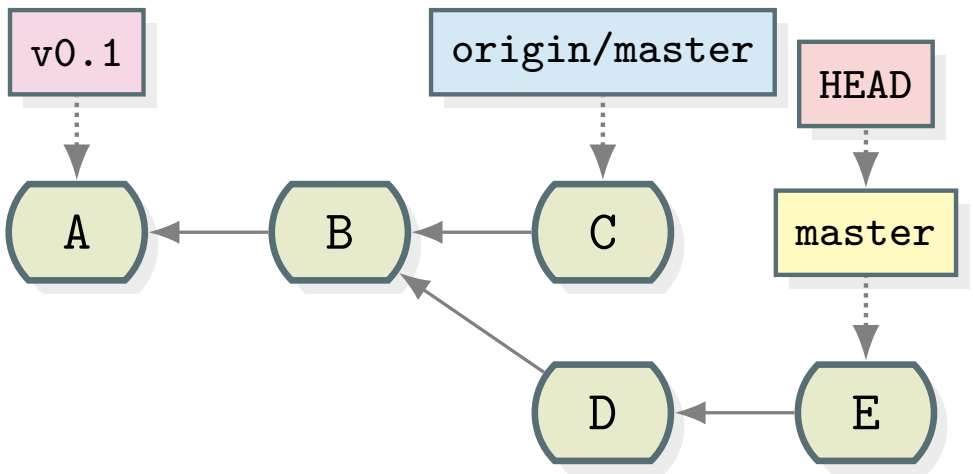
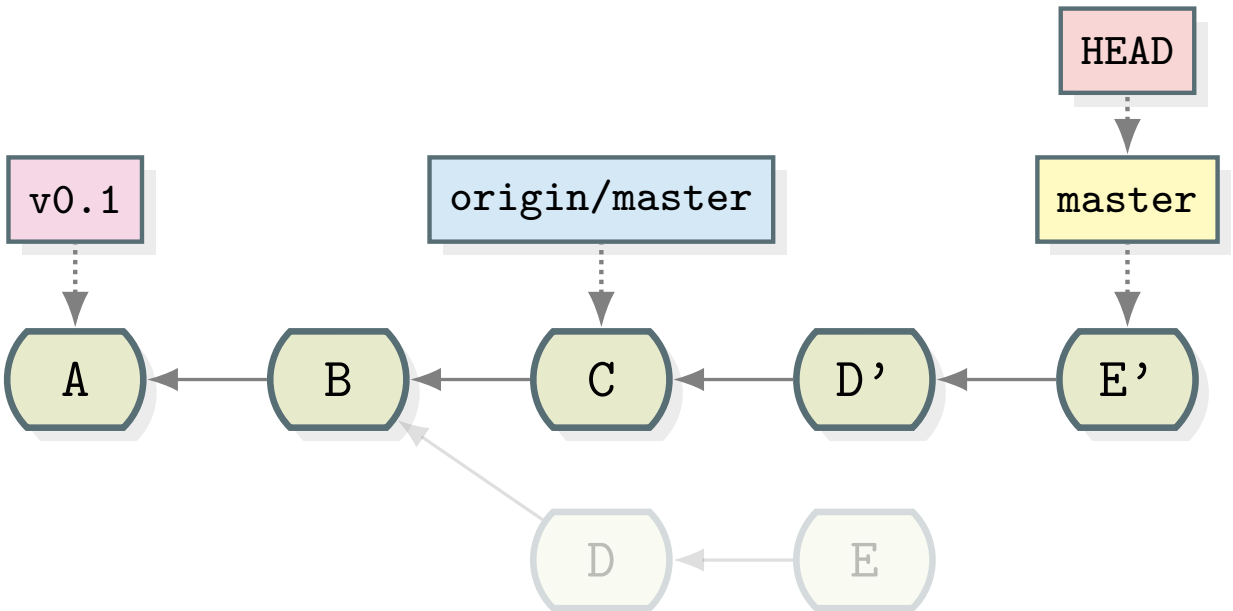


Figure: after: `git merge feature X`

Rebasing



Rebasing



Reference and Resources

- http://www.git-scm.com/docs
- http://www.git-scm.com/book/en/v2
- http://ohshitgit.com/
- https://git.wiki.kernel.org
- https://sandofsky.com/blog/git-workflow.html
- http://ndpsoftware.com/git-cheatsheet.html
- http://yasoob.me/learn-git/
- http://learngitbranching.js.org
- https://try.github.io/levels/1/challenges/1

Tools

Full Clients:

tig: `https://jonas.github.io/tig/`

gitk: `https://git-scm.com/docs/gitk`

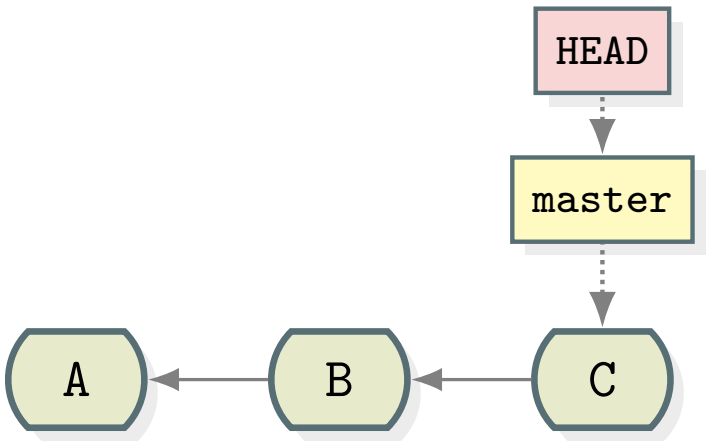
Merge / Diff Tools:

meld: `http://meldmerge.org/`

kompare: `https://www.kde.org/applications/development/kompare`

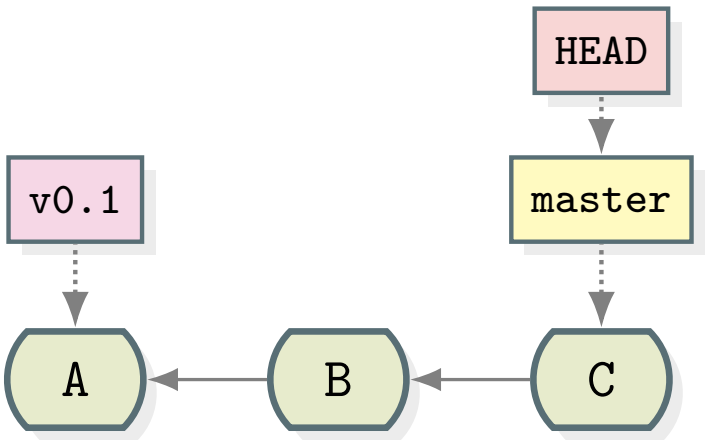
Tags

Tags are like bookmarks on commits.



Tags

```
git tag -a v0.1 A
```



Tags Syntax

Create tag:

```
1     git tag -a <commitid>
```

Delete tag:

```
1     git tag -d <tagname>
```

Filter tag:

```
1     git tag -l <pattern>
```

Sign tag:

```
1     git tag -s <tagname>
```

Showing a tag:

```
1     git show v1.4
```

Sharing Tags

You need to explicitly transfer tags to remote.

```
1          git push origin <tagname>
```

or:

```
1          git push origin --tags
```

Checking out Tags:

```
1          git checkout -b <branchname> <tagname>
```

Submodules

Submodules allow you to include or embed one or more repositories as a sub-folder inside another repository.

Adding a new submodule:

```
1     git submodule add <GIT_URL> <PATH>
```

Update a submodule:

```
1     git submodule update --remote
```

The **.gitmodules** file stores the mapping between the projects url and the local subdirectory you've pulled it into.

Submodules

Cloning a Repository with Submodules.

```
1    git clone <GIT_URL>
2    git submodule init
3    git submodule update --recursive
```

or shorter:

```
1    git clone <GIT_URL>
2    git submodule update --init --recursive
```

or even shorter:

```
1    git clone --recursive <GIT_URL>
```

Submodules

To remove a submodule you need to:

1. Delete the relevant line from the **.gitmodules** file.
2. Delete the relevant section from **.git/config**.
3. Run *git rm --cached path_to_submodule* (no trailing slash).
4. Commit the superproject.
5. Delete the now untracked submodule files.

Oh shit, I did something terribly wrong, please tell me git has a magic time machine!?!

```
1          git reflog
2          # you will see a list of every thing you've done in
               git, across all branches!
3          # each one has an index HEAD@{index}
4          # find the one before you broke everything
5          git reset HEAD@{index}
6          # magic time machine
```

Oh shit, I committed and immediately realized I need to make one small change!

```
# make your change
git add . # or add individual files
git commit --amend
# follow prompts to change or keep the commit message
# now your last commit contains that change!
```

Oh shit, I need to change the message on my last commit!

```
1          git commit --amend
2          # follow prompts to change the commit message
```

Oh shit, I accidentally committed something to master that should have been on a brand new branch!

```
1          # create a new branch from the current state of
               master
2          git branch some-new-branch-name
3          # remove the commit from the master branch
4          git reset HEAD~ --hard
5          git checkout some-new-branch-name
6          # your commit lives in this branch now :)
```

Oh shit, I accidentally committed to the wrong branch!

```
1  # undo the last commit, but leave the changes
       available
2  git reset HEAD~ --soft
3  git stash
4  # move to the correct branch
5  git checkout name-of-the-correct-branch
6  git stash pop
7  git add . # or add individual files
8  git commit -m "your message here"
9  # now your changes are on the correct branch
```

Fuck this noise, I give up.

```
1           cd ..
2           rm -rf git-repo-dir
3           git clone https://some.github.url/git-repo-dir.git
4           cd git-repo-dir
```

End

Figure:
https://xkcd.com/1296/