Java

Git

Tobias Hanf, Manik Khurana

31. Januar 2022

Java-Course

Overview

1. Theory
2. Branches
3. Repositories
4. Best Practice

Theory

Git

'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' `https://git-scm.com/`

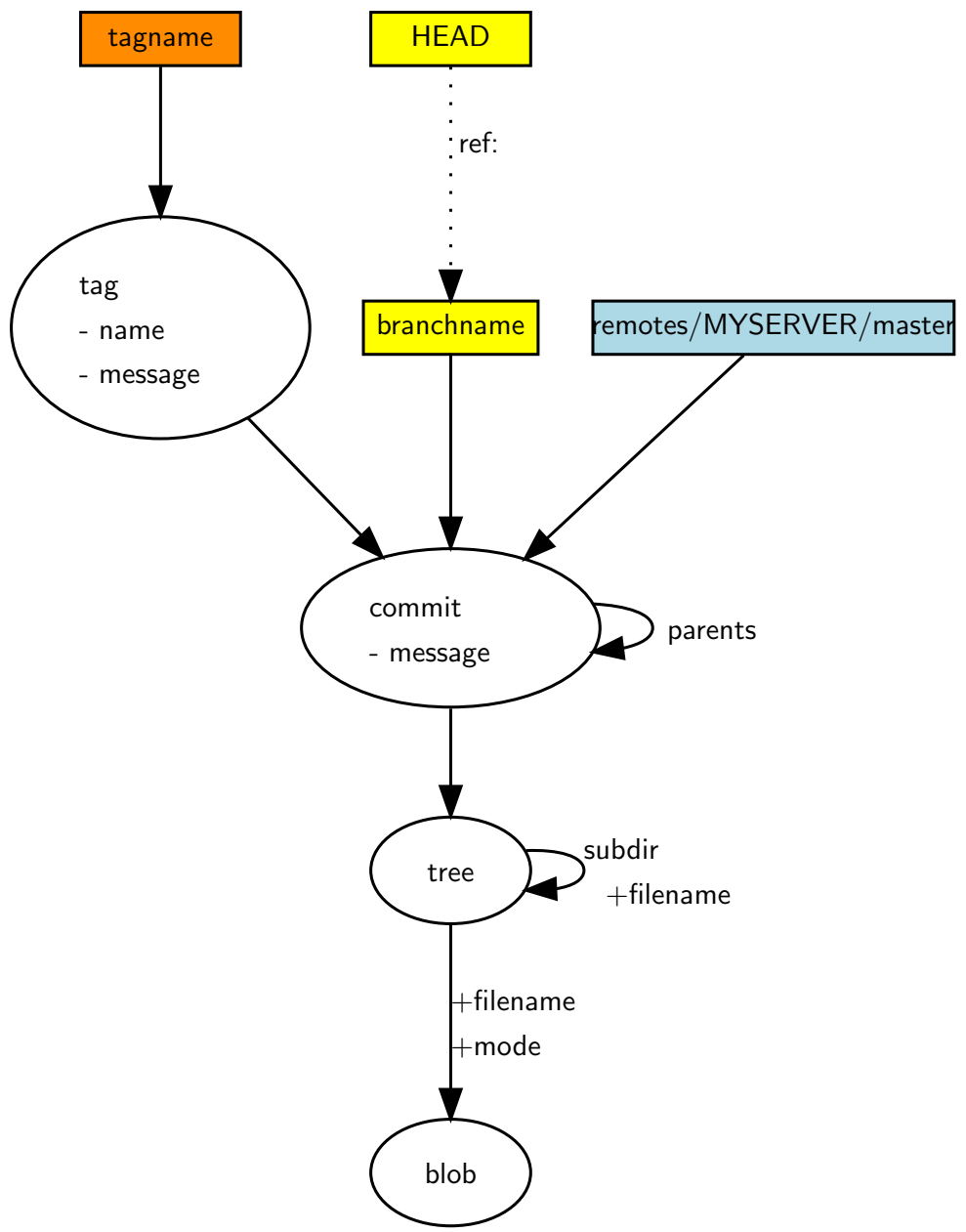


Abbildung 1:

`https://eagain.net/articles/git-for-computer-scientists/`

Data

- commit stores snapshot of data
- a commit can have one or more parent commits
- commits from an Directed Acyclic Graph (DAG)
- a commit can have a "post-it"

Stages

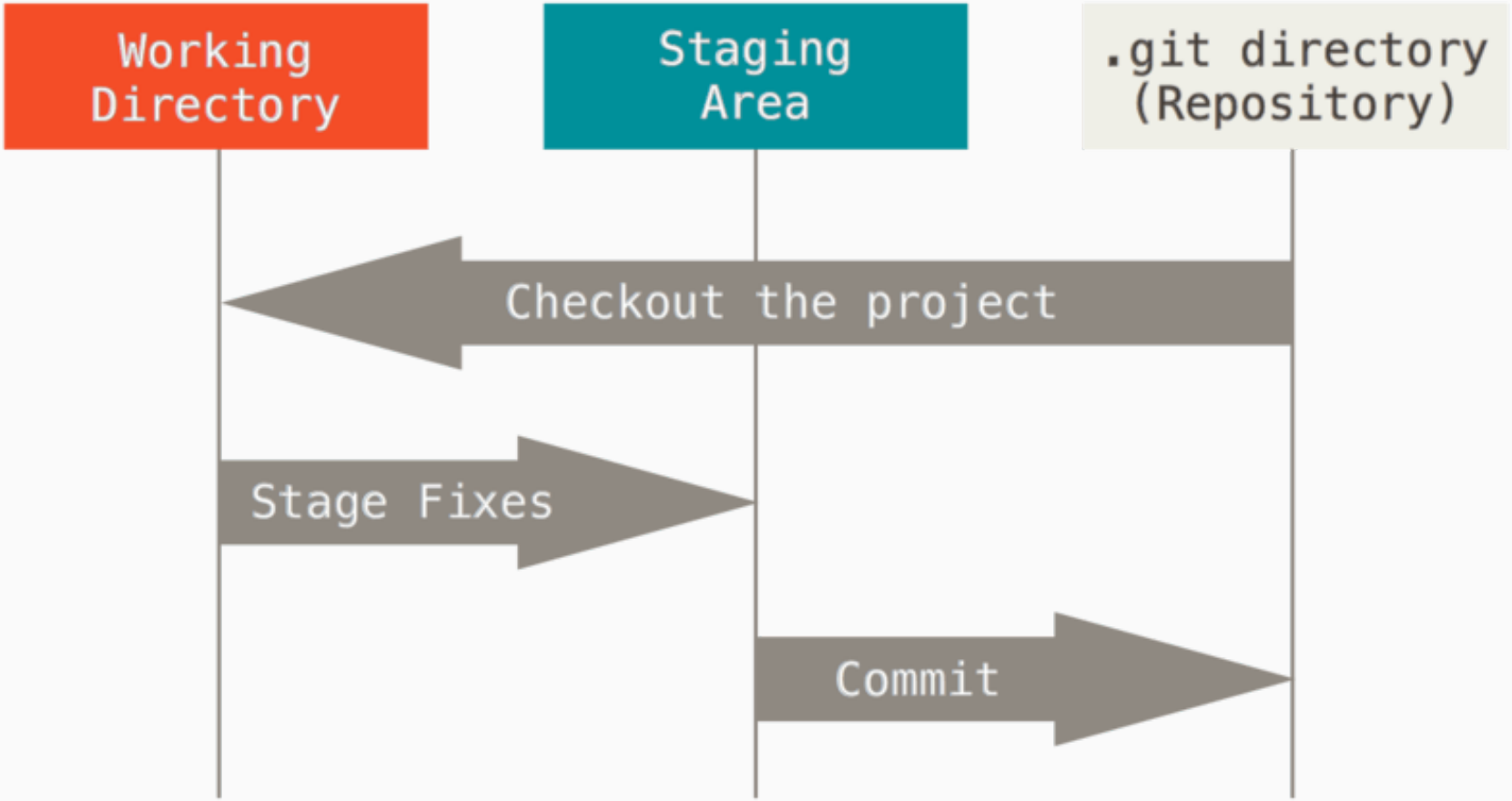


Abbildung 2: `https://git-scm.com/book/en/v2/images/areas.png`

Branches

Branches i

A Branch is a name "referencing" a commit. The current branch defines the position of the HEAD pointer.

Images taken from: `https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell`

Branches ii

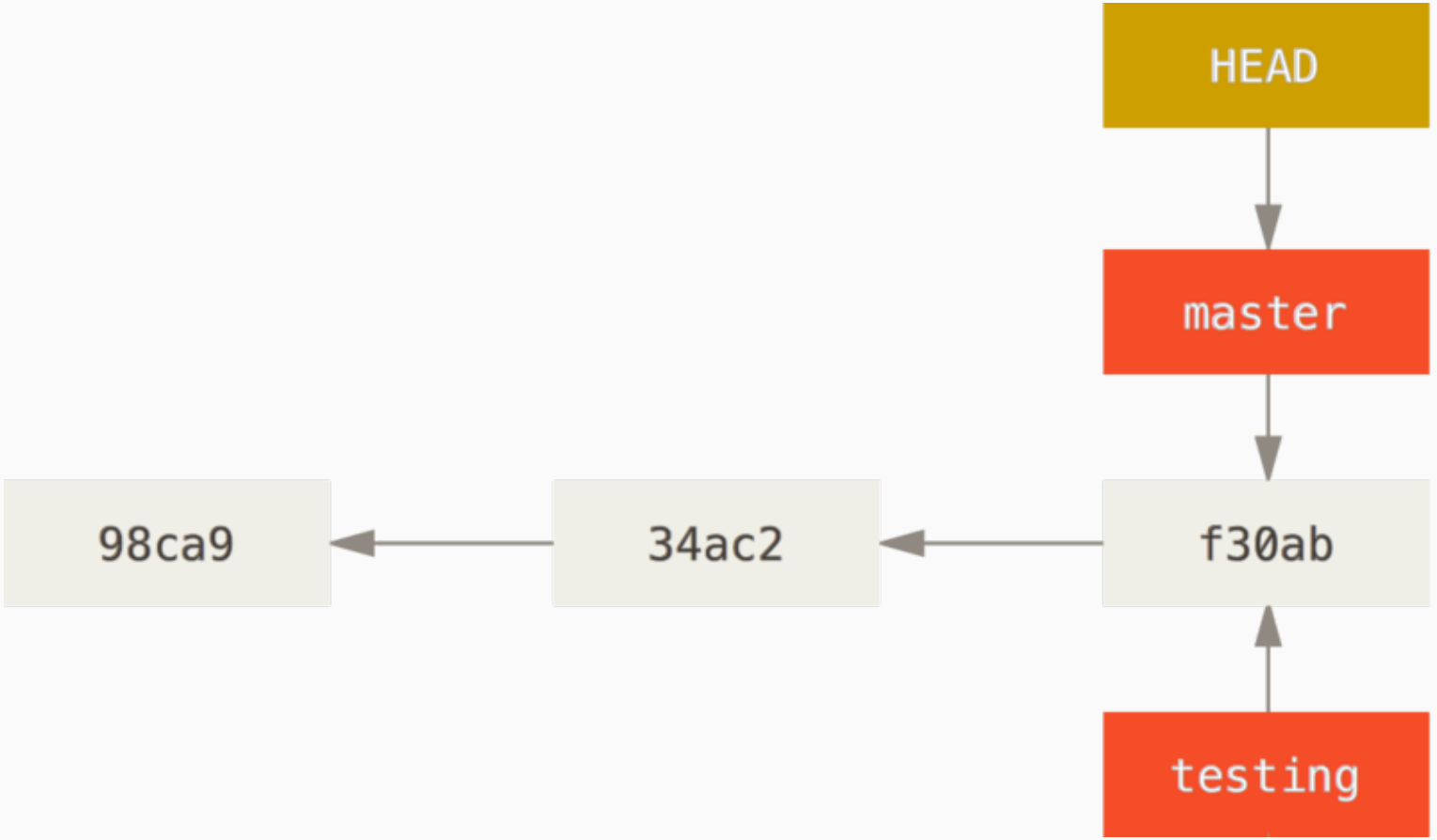


Abbildung 3: git checkout master

Branches iii

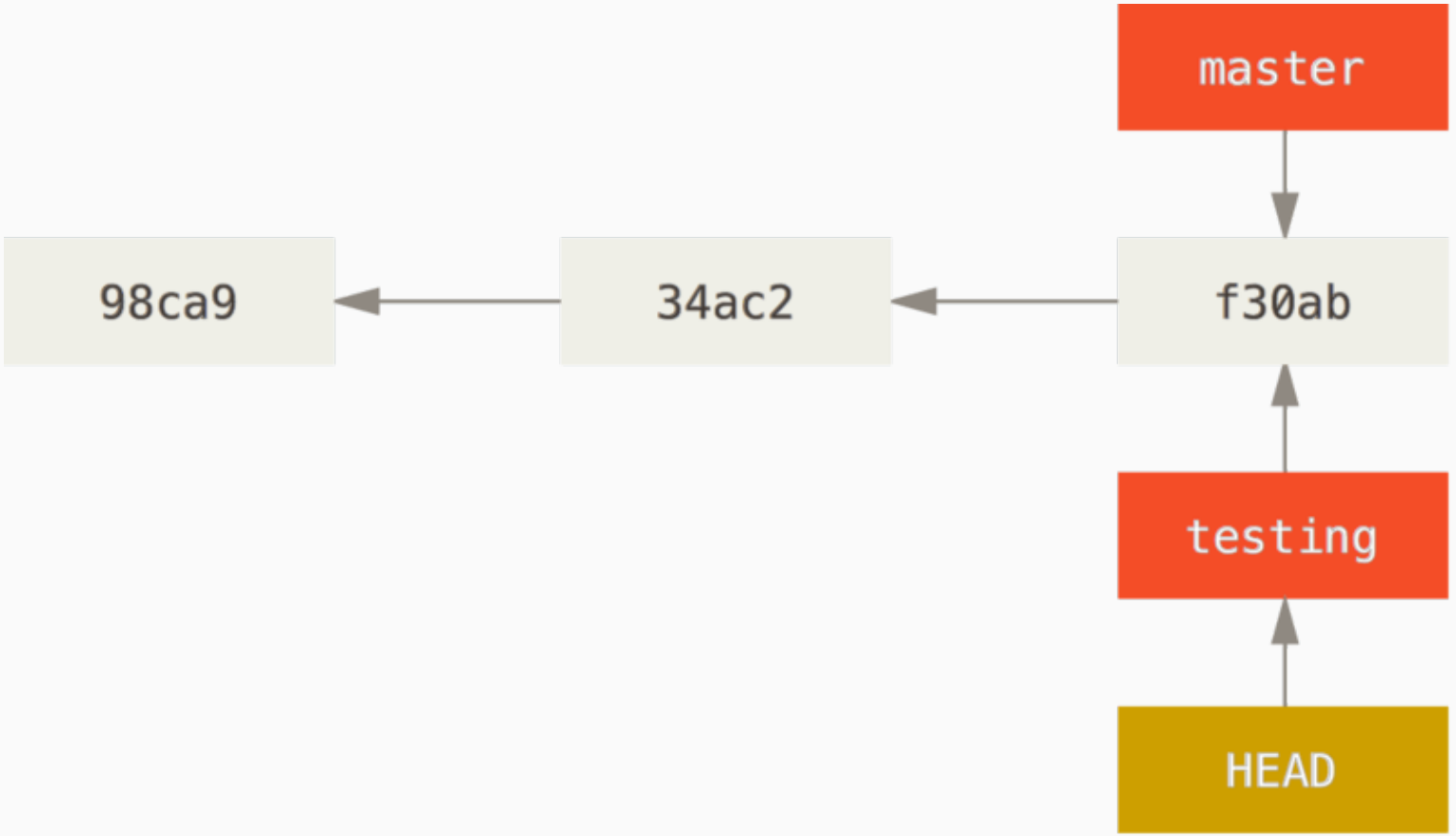


Abbildung 4: git checkout testing

Branches iv

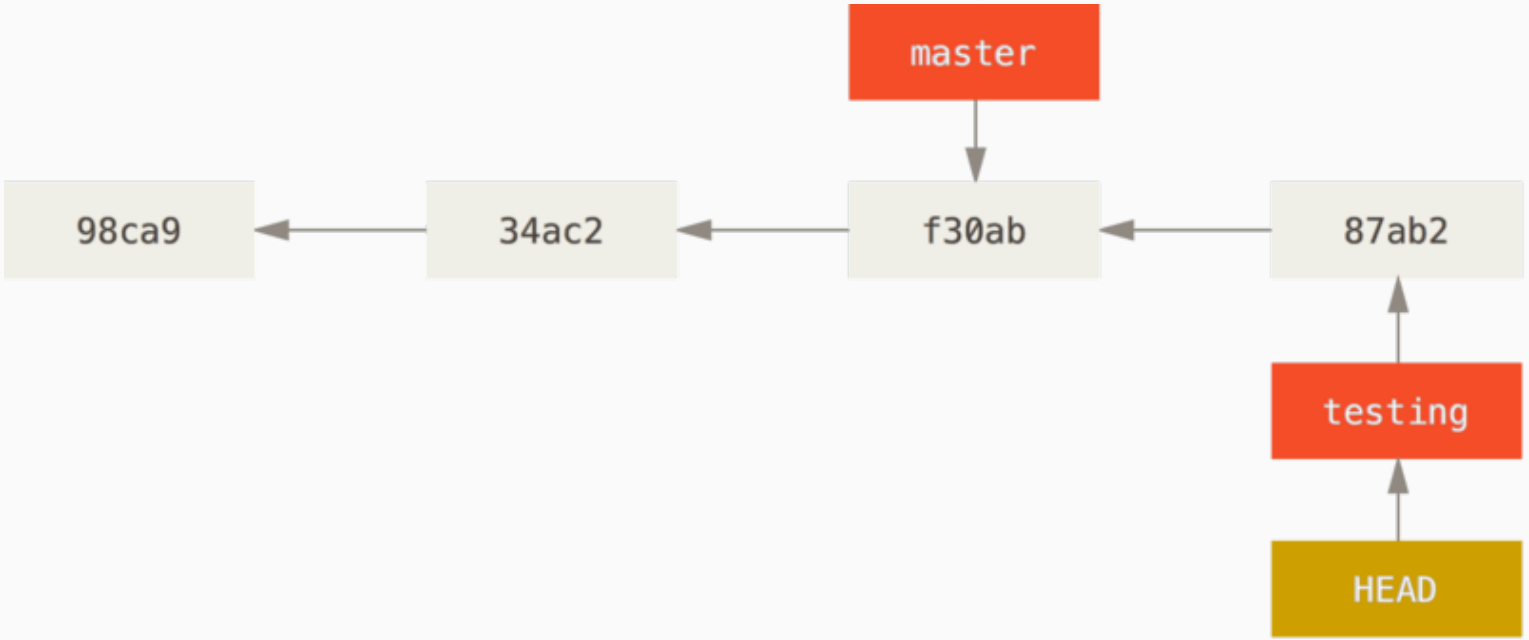


Abbildung 5: git commit -m 'something something'

Branches v

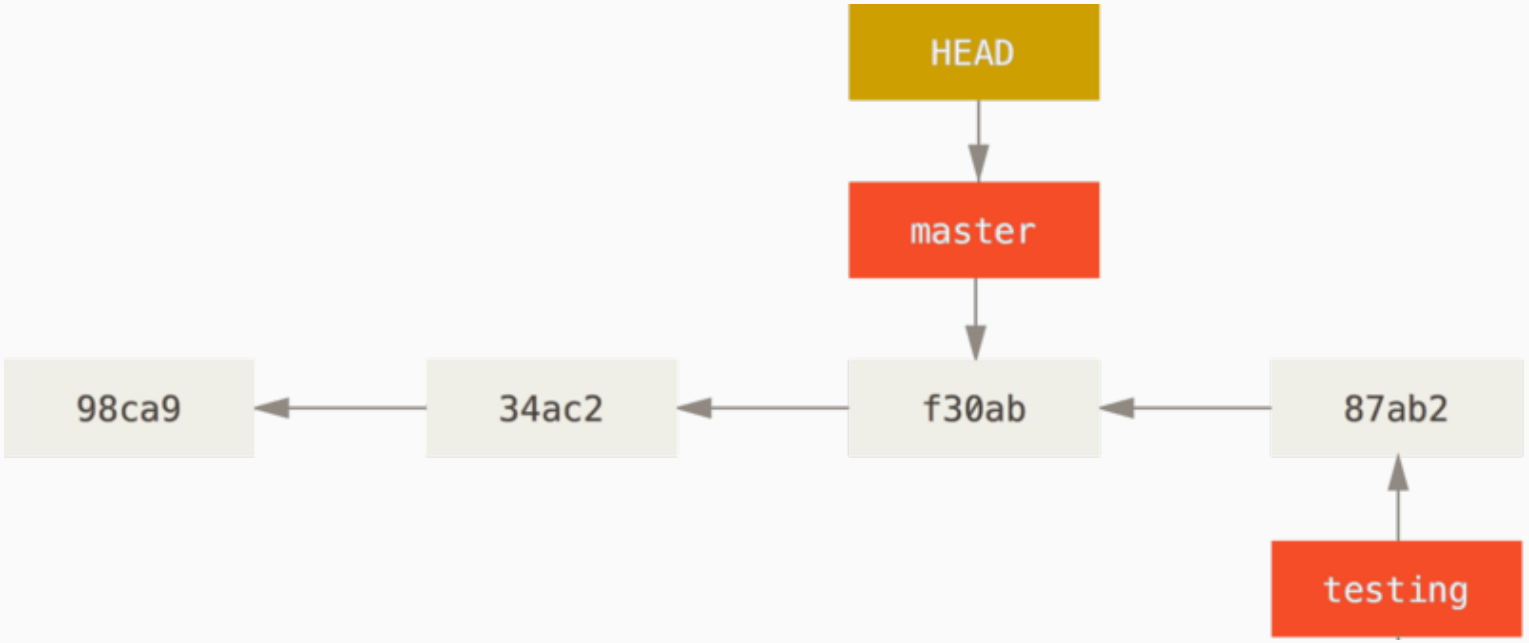


Abbildung 6: git checkout master

Branches vi

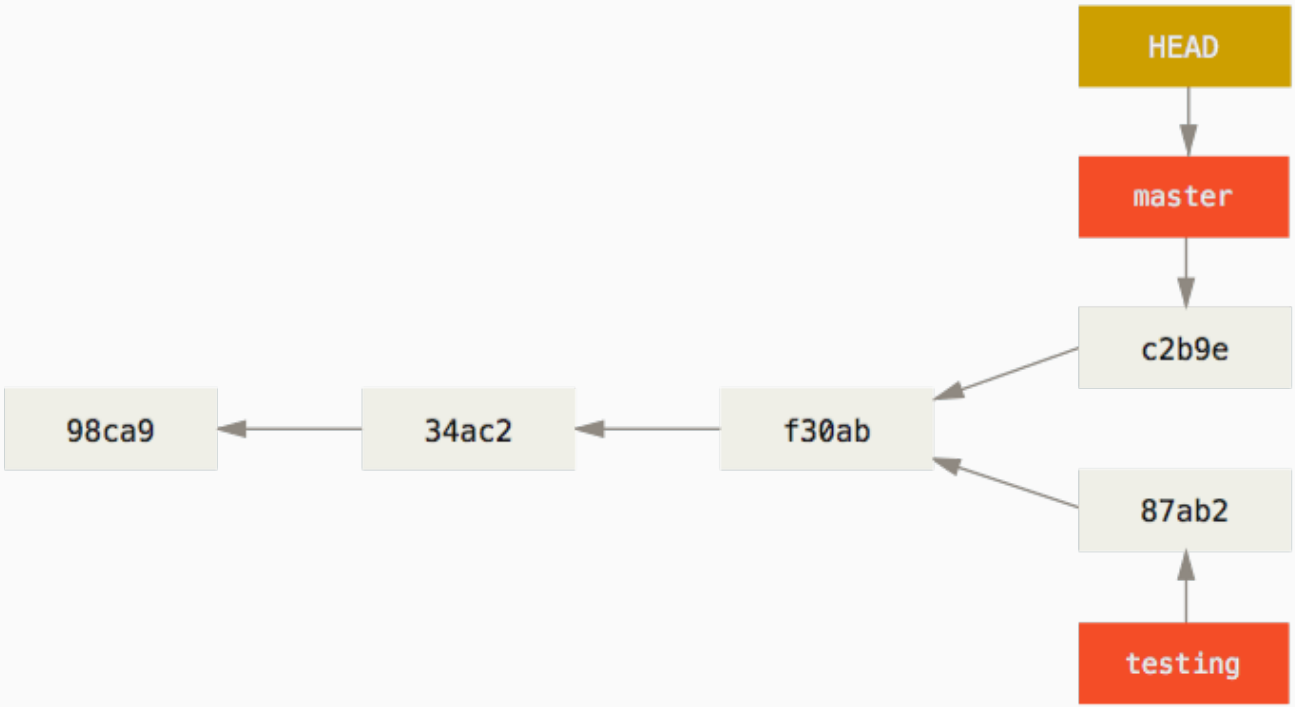


Abbildung 7: git commit -m 'other things'

Branches vii

Useful commands:

- Move branch to different commit:
  `git branch -f <branch-name> <new-tip-commit>`
- Create new branch and checkout:
  `git checkout -b <branch-name>`

Merging i

Merging: combining two or more branches

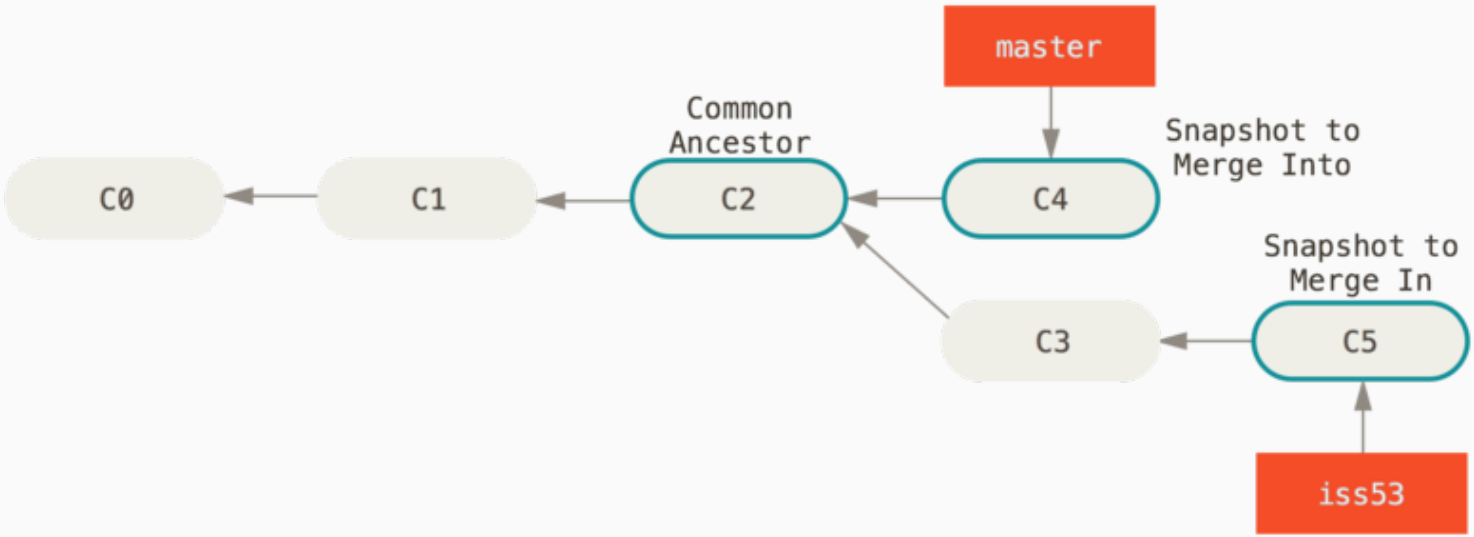


Abbildung 8: git checkout iss53

Merging ii

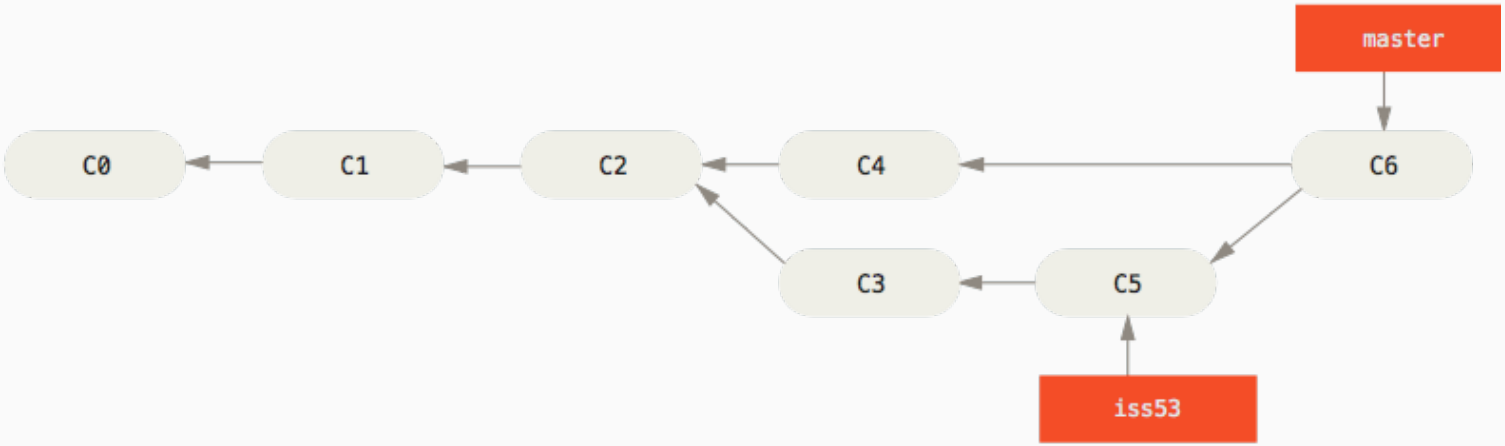


Abbildung 9: git checkout master; git merge iss53

https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Rebaseing i

Rebase: Moves commits from one branch (and the branch) onto the start of another

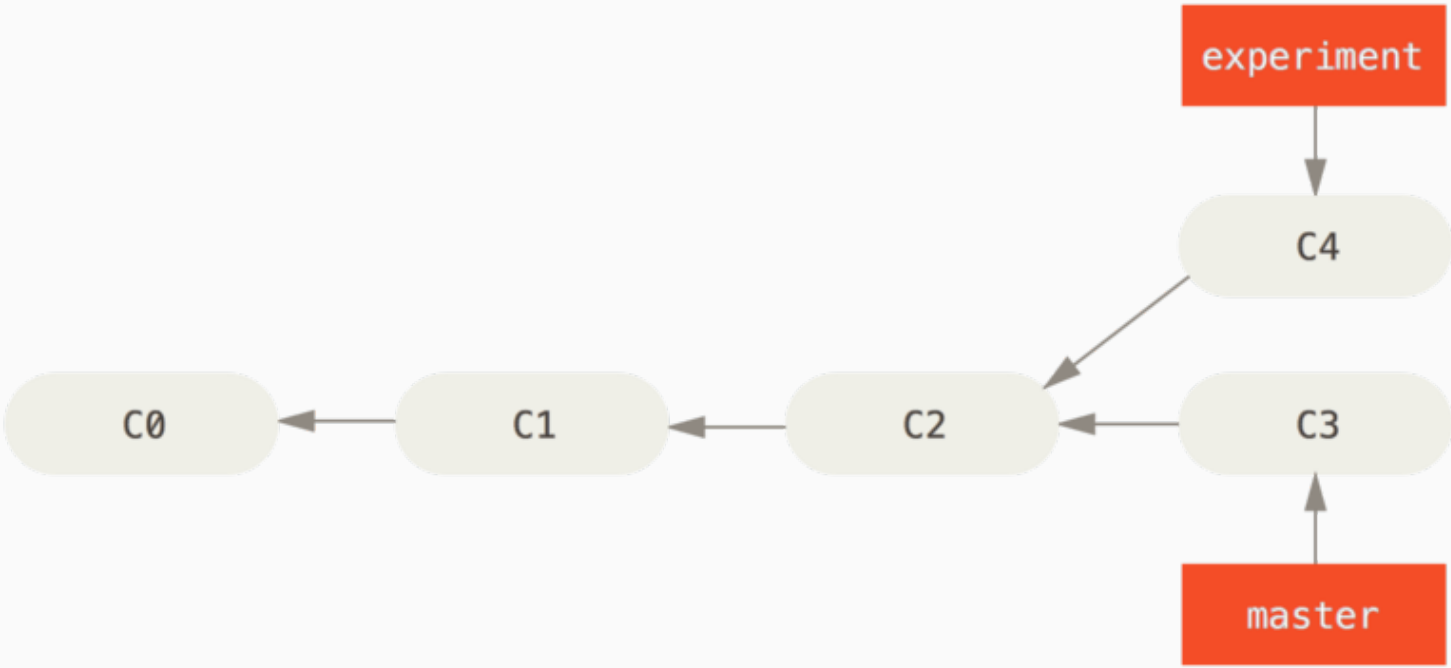


Abbildung 10: git checkout experiment

Rebaseing ii

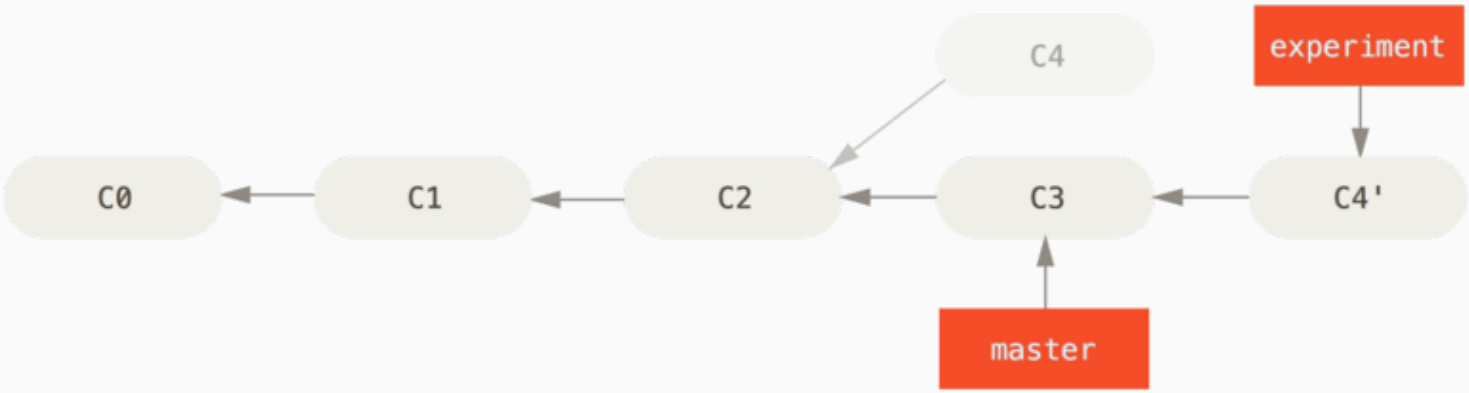


Abbildung 11: git rebase master

Rebaseing iii

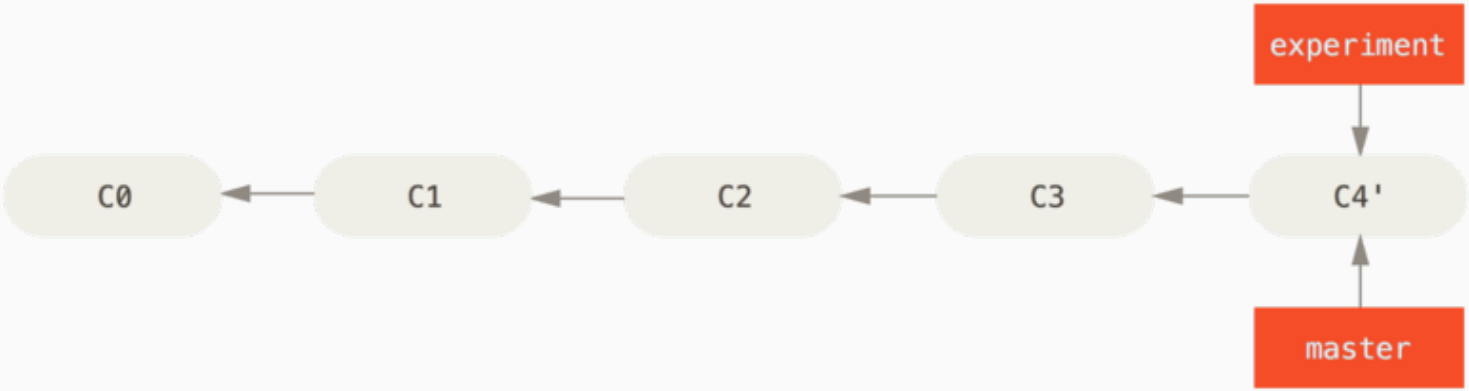


Abbildung 12: git checkout master; git merge experiment

https://git-scm.com/book/en/v2/Git-Branching-Rebasing

Cherry-Picking

```
git cherry-pick <commit>
```

- Cherry-Picking copy a specific commit to the branch
- can be handy
- but can lead to problems (duplicate commits)
- some people like it others not

Merge vs Rebase (vs FF-Merge)

Merge

- keeps history (non-destructive)
- more complicated structure
- every merge adds an extra commit

Rebase

- re-writes project history
- much cleaner history (structure)
- **never use it on public branches**

Merge Conflict i

Abbildung 13: `https://webdevkin.ru/media/img/courses/git/meme/git-merge-without-conflict.jpg`

Merge Conflict ii

Merge Conflict: occurs if you merge and two commits of different branches change the same part of a file.

```
1         $: git merge conflict
2             Auto-merging second.txt
3             CONFLICT (content): Merge conflict in second.txt
4             Automatic merge failed; fix conflicts and then
    commit the result.
5
```

Merge Conflict iii

The file on conflict will be changed to something like this

```
$: cat second.txt
<<<<<<< HEAD
First file
=======
Third file
>>>>>>> conflict

```

Select what you want and remove everything else. Then add and commit.

```
git add second.txt
git commit -m "Merge successful"
```

Repositories

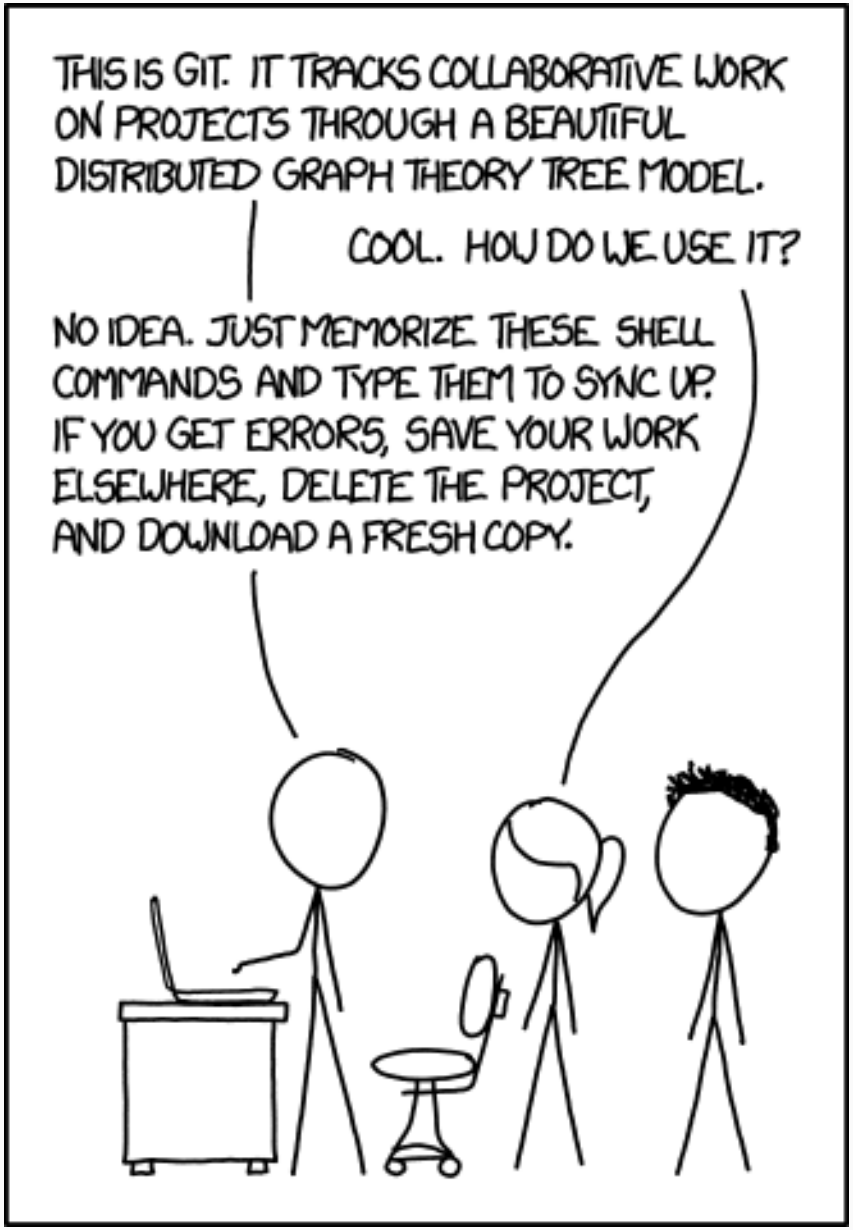


Abbildung 14: `https://xkcd.com/1597/`

Local vs Remote i

- git is a distributed VCS
- there is no **ONE** repository
- every one has a different version of a repository
- changes can be exchanged via a remote repository
- most errors occur while working with a remote repo

Local vs Remote ii

To get a local copy of a remote repo: `git clone <repository-url>`

Changes you make are only locally

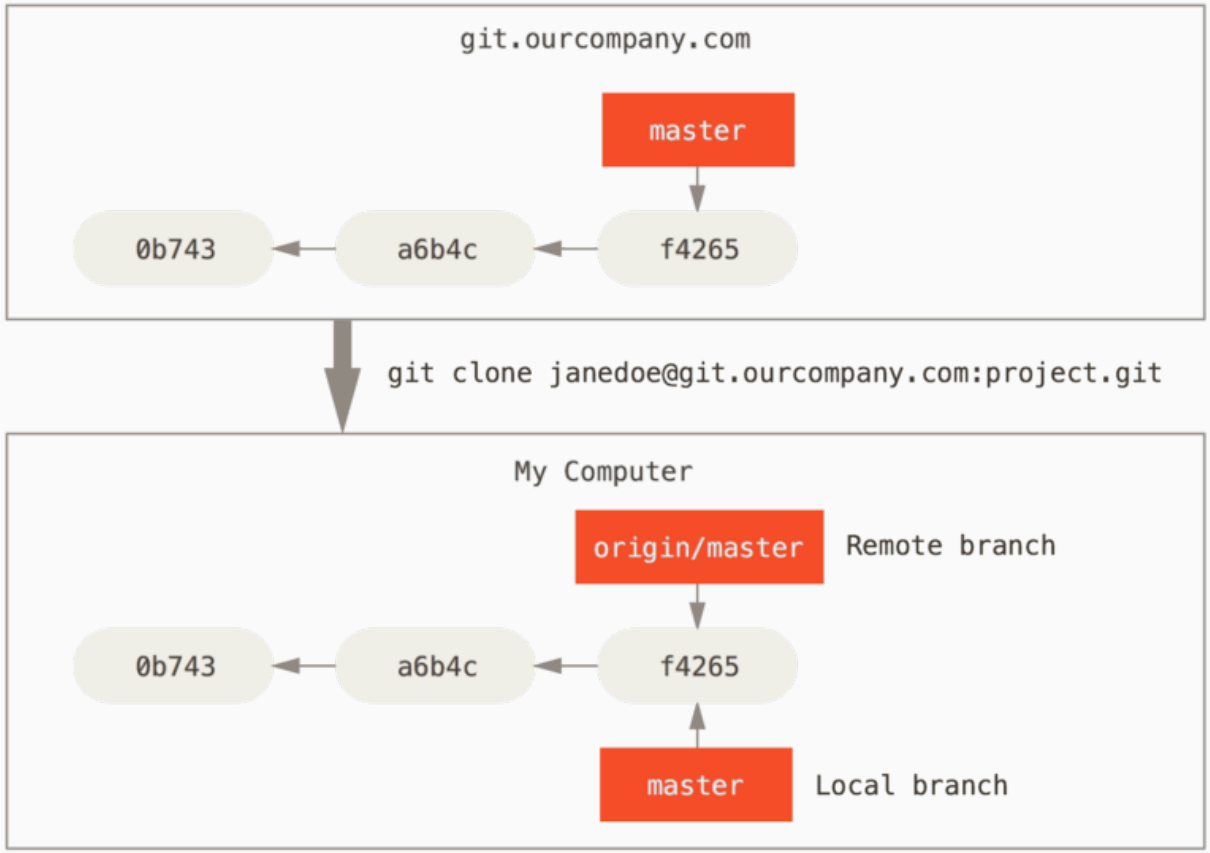
You have to explicitly `push` them to the remote repo.

There are this a local and remote version for a remote branch in your local repo.

These remote versions have a special name: `origin/<branch-name>`

Fetch i

`git fetch`: get all changes from the remote repository



Fetch ii

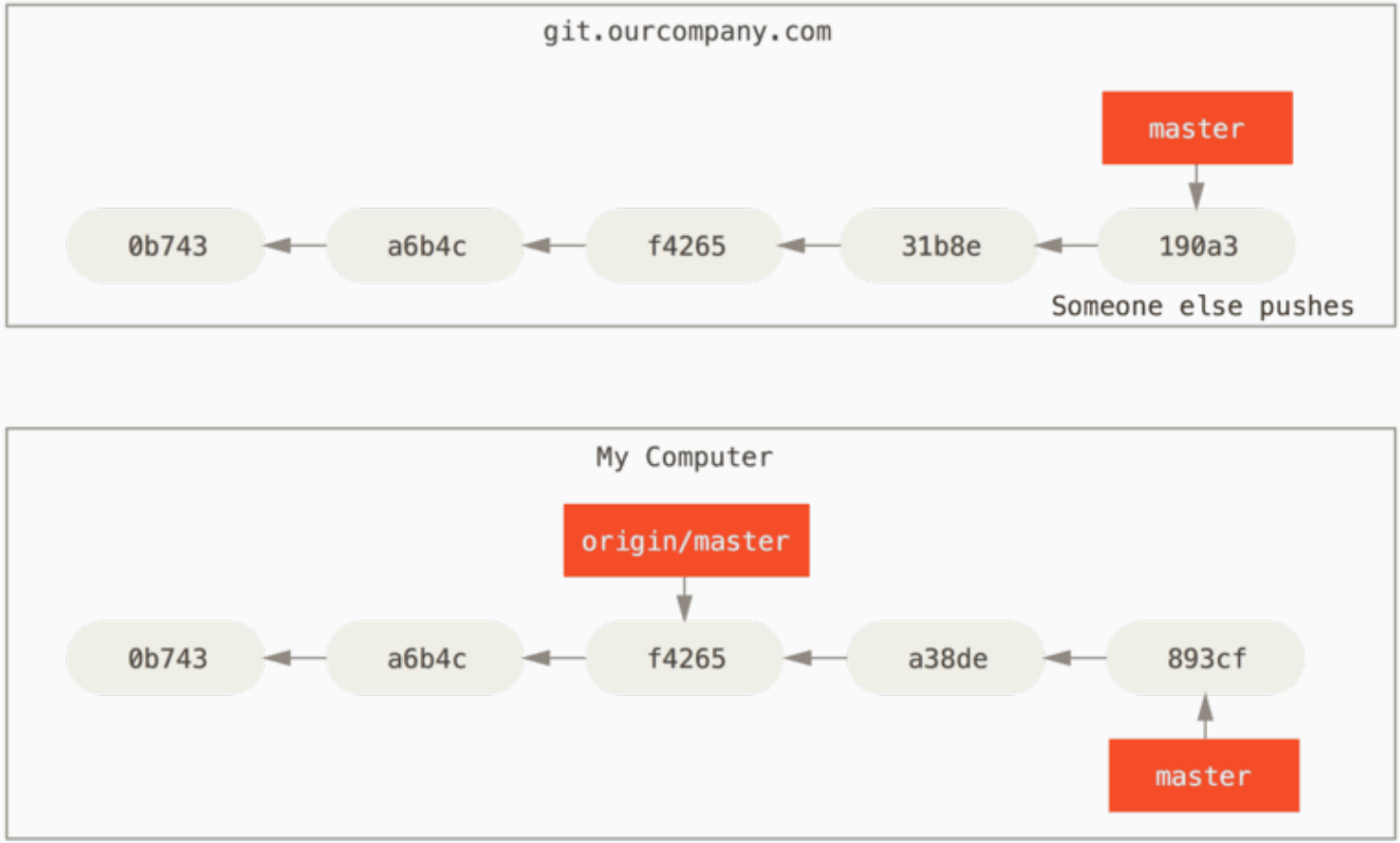


Abbildung 15: Add new commits to master

Fetch iii

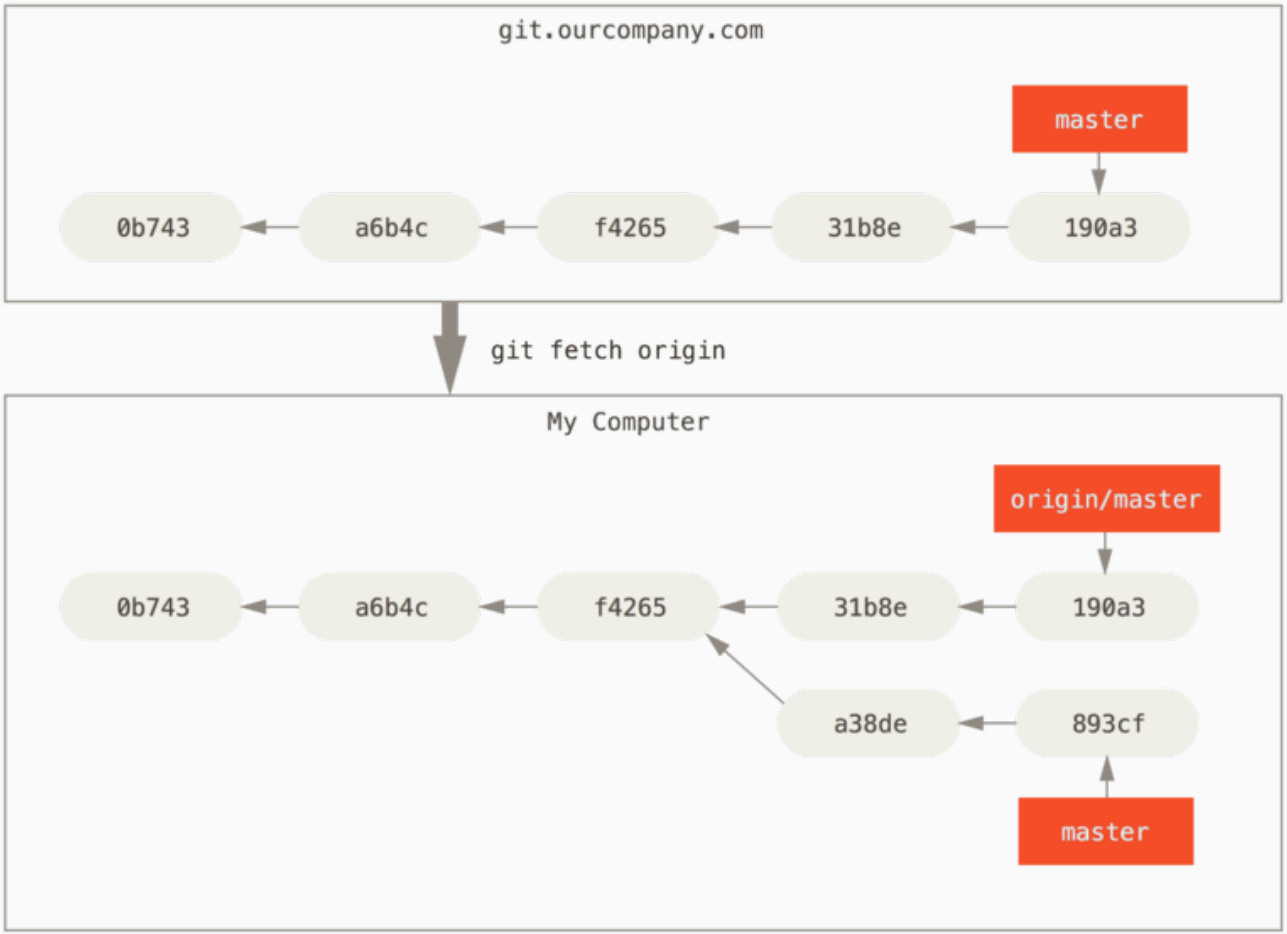


Abbildung 16: `git pull`

Pull

Combines: `git fetch` und `git merge origin/<branch-name>`

Other option: `git merge --rebase`
Does a rebase instead of a merge

Push

`git push`: pushes your locale changes to the remote repository

For this to work all changes from the remote repo need to be in your local repo.

```
        Pushing to ...
        To ...
        ! [rejected]        master -> master (fetch first)
        error: failed to push some refs to '...'
        hint: Updates were rejected because the remote contains work that you do
        hint: not have locally. This is usually caused by another repository pushing
        hint: to the same ref. You may want to first integrate the remote changes
        hint: (e.g., 'git pull ...') before pushing again.
        hint: See the 'Note about fast-forwards' in 'git push --help' for details.

```

Saving your repo

- `git checkout <branch>`: moves uncommited changes to branch
- `git stash`: saves current state of directory
- `git reset`: resets HEAD to specific state (removes commits)
- `git revert`: reverts existing commits by creating new commit
- `git blame`: see who fucked up (ruining friendships since 2005)

Submodules

A git repo can have other git repos as submodules

- Add submodule: `git submodule add <repo-url>`
- Commit add: `git commit -am 'Add submodule ...'`
- Push changes: `git push origin master`

Use a repo with submodules

- Get repo: `git clone <repo-url>`
- Init: `git submodule init`
- or: `git clone --recurse-submodules <repo-url>`
- update: `git submodule update --remote <module-repo-name>`

Best Practice

General

- commit frequently, push often
- if there is a problem, keep calm (and use a search engine)
- use (a lot of) branches
- push only (semi) tested code
- use rebase for cleanups

Commit messages

Read: `https://cbea.ms/git-commit/`

- Subject should be 50 characters
- Capitalize Subject and use imperative mood
- wrap body at 72 chars
- explain the What and Why in body

branch strategies

Different strategies:

- only push tested code to main
- have development branch to merge features
- every feature/issue/bug should have its own branch
- these branches should be short lived
- don't work on branches with lots of outside changes

Gitlab/Github

- learn git first
- use issues to communicate
- leverage Pull/Merge requests to add features
- master the concepts not the platform

Link collection

- https://git-scm.com/docs
- https://git-scm.com/book/en/v2
- https://cbea.ms/git-commit/
- https://learngitbranching.js.org/
- https://stackoverflow.com/
- https://docs.github.com/en/get-started
- https://docs.gitlab.com/ee/topics/use_gitlab.html