git Tutorial

Nicola Chiapolini

Physik-Institut
University of Zurich

September 4, 2017

Based on talk by Emanuele Olivetti   https://github.com/emanuele/introduction_to_Git
This work is licensed under the *Creative Commons Attribution-ShareAlike 3.0 License*.

Motivation to use Version Control

Problem 1

"Help! my code worked yesterday, but I can't recall what I changed."

- ▶ track modifications
- ▶ access old version

Problem 2

"We would like to work together, but we don't know how!"

- ▶ concurrent editing
- ▶ merging
- ▶ development versions

Outline

Introduction

Single developer + local repository
    Demo/Exercise: single+local

Multiple developers + remote central repository
    Demo/Exercise: multi+remote/central

Behind the Scenes

Outline

Survey: Version Control

- ▶ Q1: Have you heard about *version control*?

- ▶ Q2: Do you use a version control software (cvs, svn, hg, bzr, git)?

- ▶ Q3: How much experience do you have with git?

Survey: Version Control

- Q1: Have you heard about *version control*?
- Q2: Do you use a version control software (cvs, svn, hg, bzr, git)?
- Q3: How much experience do you have with git?

Survey: Version Control

- Q1: Have you heard about *version control*?

- Q2: Do you use a version control software (cvs, svn, hg, bzr, git)?

- Q3: How much experience do you have with git?

Uses for git

"*Version control* is a system that records changes to a file or set of files over time so that you can recall specific versions later."
– https://git-scm.com/book

- ▶ checkpoints/backups/releases
- ▶ document developer effort
- ▶ collaboration across the globe

- ▶ for anything that's text
  - ▶ code
  - ▶ thesis/papers
  - ▶ system config files (vcsh)
- ▶ . . . and everything else ("gitify your life", git-annex)

Version Control: Local
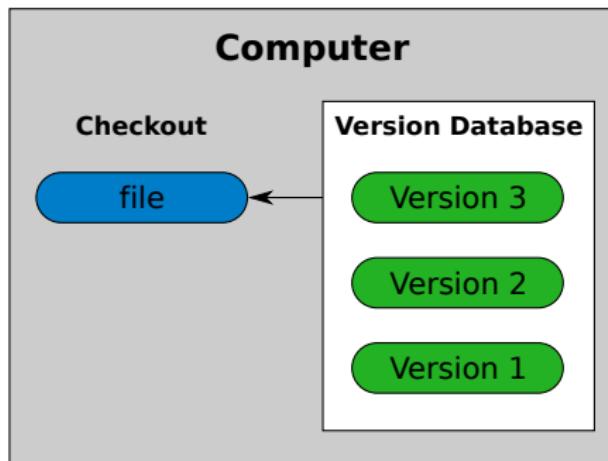
checkout working directory

version database repository

Version Control: Central
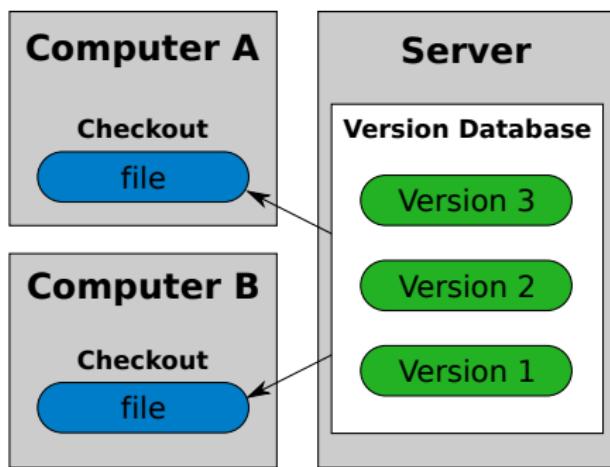
Version Control: Distributed
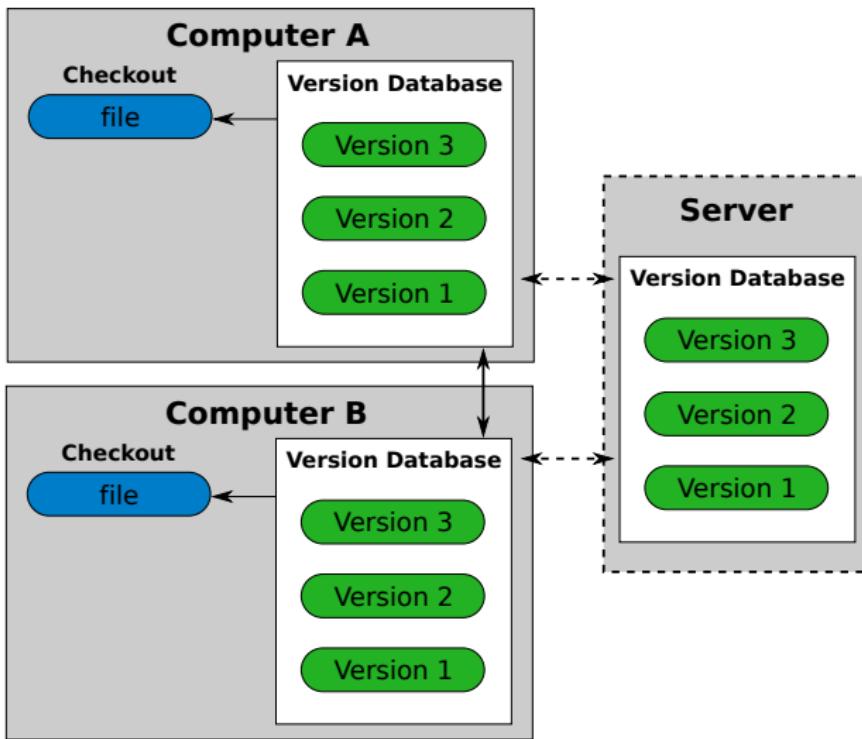
git: Help

```
usage: git [...]
           <command> [<args>]

These are common Git commands used in various situations:

[...]

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

git: Introduce yourself

```
git config --global user.name "Nicola Chiapolini"
```

```
git config --global user.email "nchiapol@physik.uzh.ch"
```

Outline

Introduction

Single developer + local repository
Demo/Exercise: single+local

Multiple developers + remote central repository
Demo/Exercise: multi+remote/central

Behind the Scenes

single+local: Init

```
git init
```

- Creates an empty git repository.
- Creates the git directory: `.git/`
- Your prompt may change.
  (If you added `$(__git_ps1)`)

- does not change your files

single+local: Init

<center>`git init`</center>

- ▶ Creates an empty git repository.
- ▶ Creates the git directory: `.git/`
- ▶ Your prompt may change.
  (If you added `$(__git_ps1)`)

- ▶ does not change your files

single+local: Add

```
git add file1 [file2 ...]
```

- ▶ Adds new files to be tracked by git
- ▶ Adds changes from working dir for next commit
- ▶ DOES NOT add info on file permissions other than *exec/noexec*
- ▶ DOES NOT add directories *per se*.
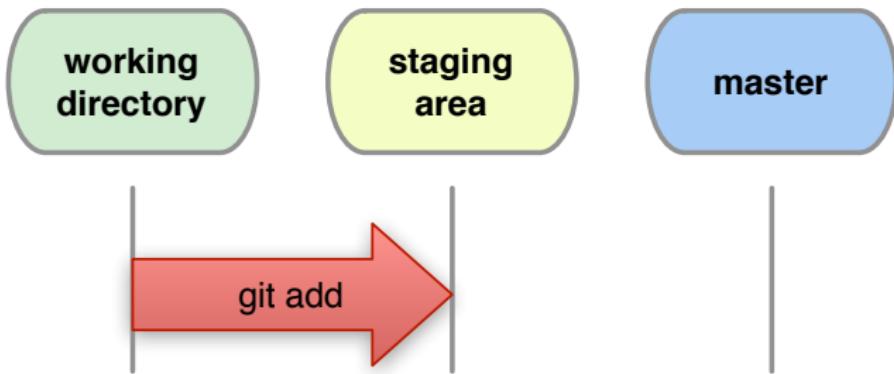
single+local: Commit

```
git commit [-m "Commit message."]
```

Records changes from the staging area to master.
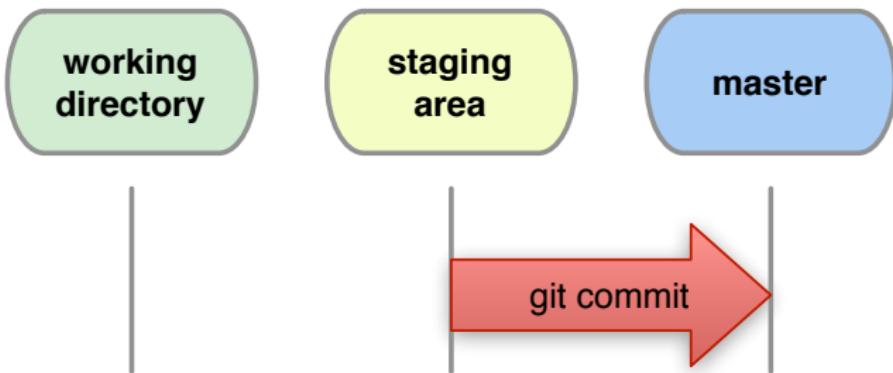
single+local: Direct Commit

```
git commit file1 file2 [-m "Commit message."]
```

Records all changes of `file1`, `file2` from working dir and staging area to master.
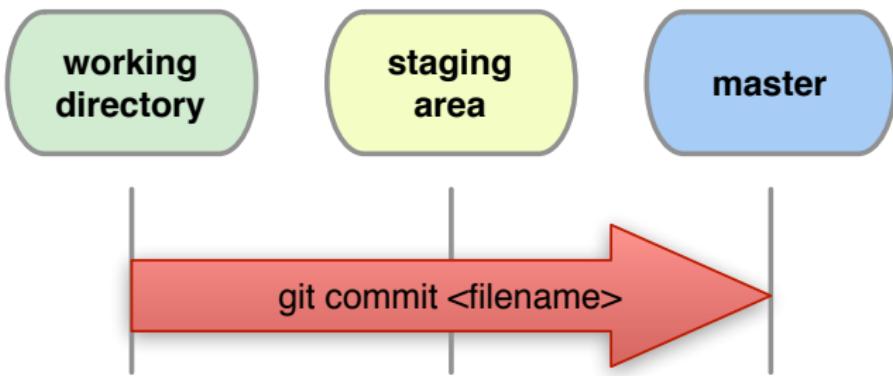

```
git commit -a[m "Commit message."]
```

Records all changes in working dir and staging area. *Be Careful!*

single+local: Diff

```
git diff [filename|...]
```

Shows changes between *working directory*
and *staging area*
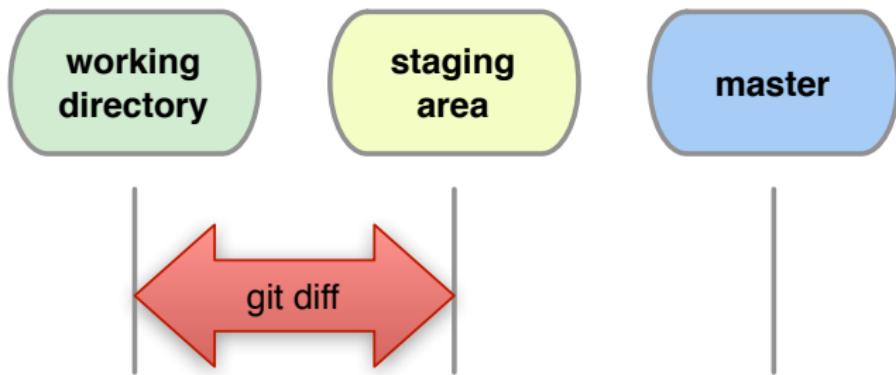
single+local: Diff Staged

How do I see what is staged?
`git diff --staged` shows differences
between staging area and last commit.
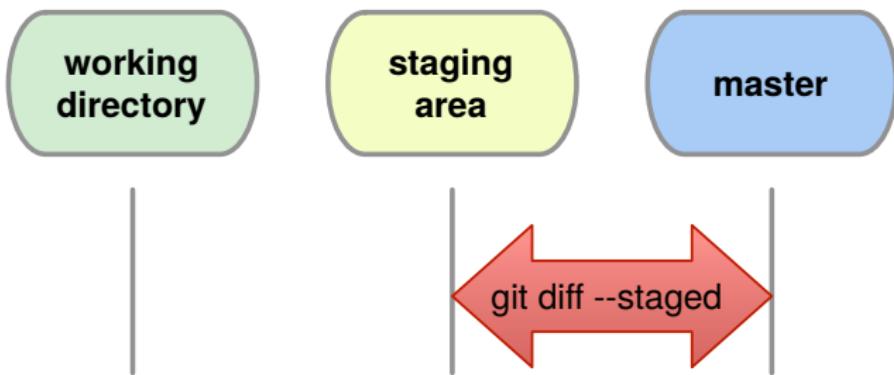
single+local: Commit History

```
git log [--oneline] [--patch] [file|branch]
```

Shows the history of a file or branch.
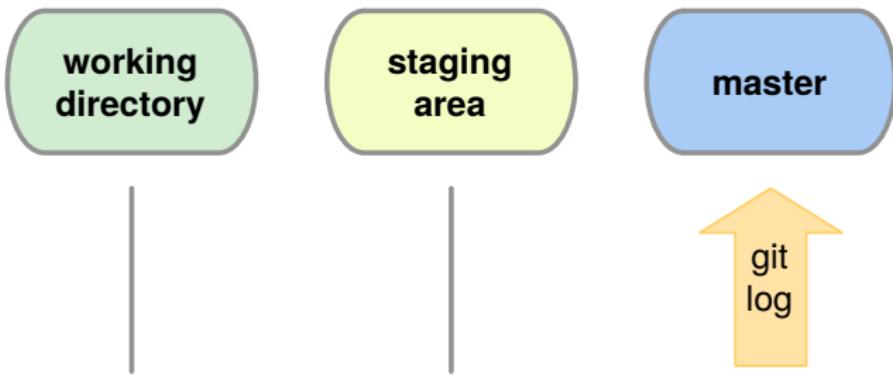
single+local: Old Changes

$$\texttt{git diff <commit A> <commit B>}$$
$$\texttt{git show <commit>}$$

Shows the changes stored in commits.

single+local: Graphic Logs

$$\texttt{qgit} \text{ (or } \texttt{gitg} \text{ or } \ldots \text{)}$$

GUI to browse the git repository.
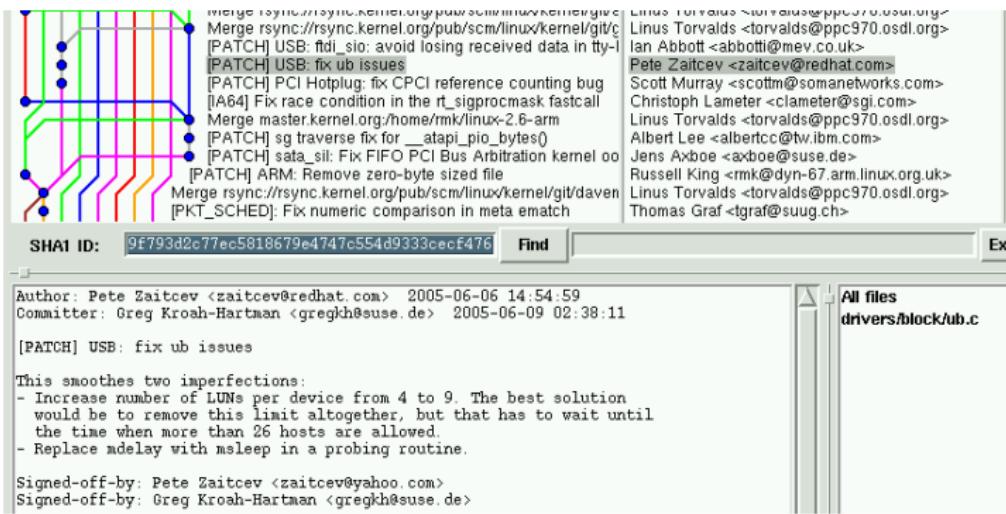
single+local: Changing Version

`git checkout <file|commit>`

single+local: Changing Version
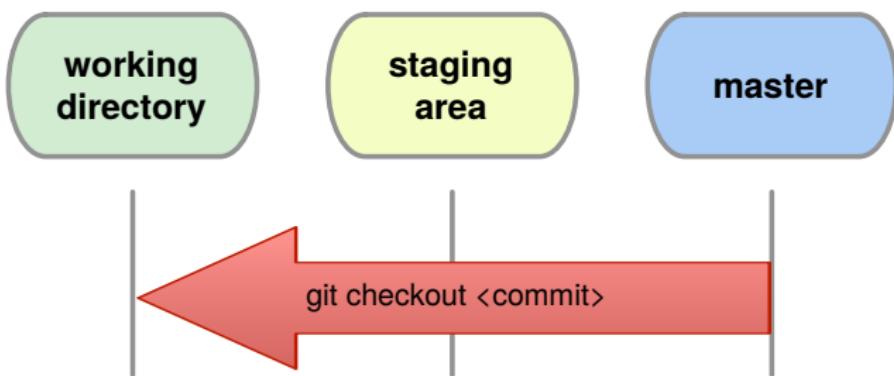
single+local: (Re)move.

Warning: whenever you want to *remove*, *move* or *rename* a tracked
file use git:

$$\texttt{git rm <filename>}$$

$$\texttt{git mv <oldname> <newname>}$$

Remember to `commit` these changes!

$$\texttt{git commit -m "File (re)moved."}$$

Outline

Introduction

Single developer + local repository
Demo/Exercise: single+local

Multiple developers + remote central repository
    Demo/Exercise: multi+remote/central

Behind the Scenes

Outline

multi+remote/central: Setup

multi+remote/central: Clone

```
git clone <URL>
```

Creates two local copies of the whole remote repository.

Remote (Server)

Version Database

Hint
git remote -v   shows **name** and URL of the remote repository.

multi+remote/central: Clone

<center>

`git clone <URL>`

</center>
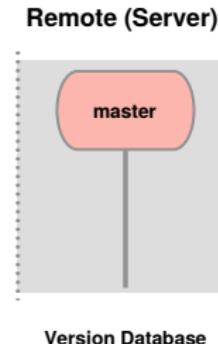
Creates two local copies of the whole remote repository.

> **Hint**
> `git remote -v`   shows **name** and `URL` of the remote repository.
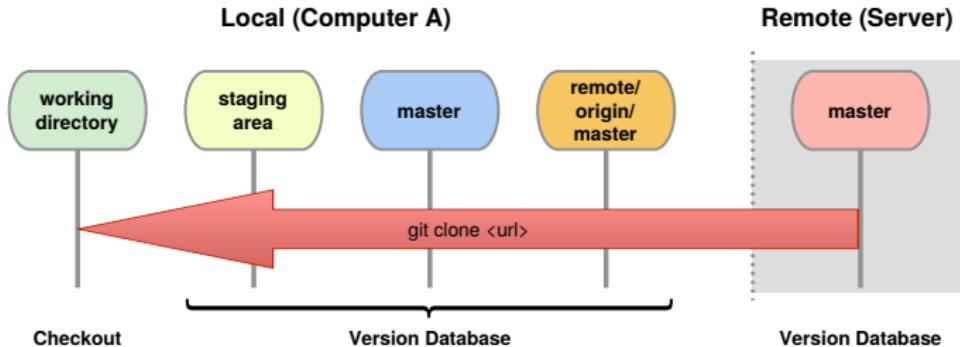
multi+remote/central: Commands
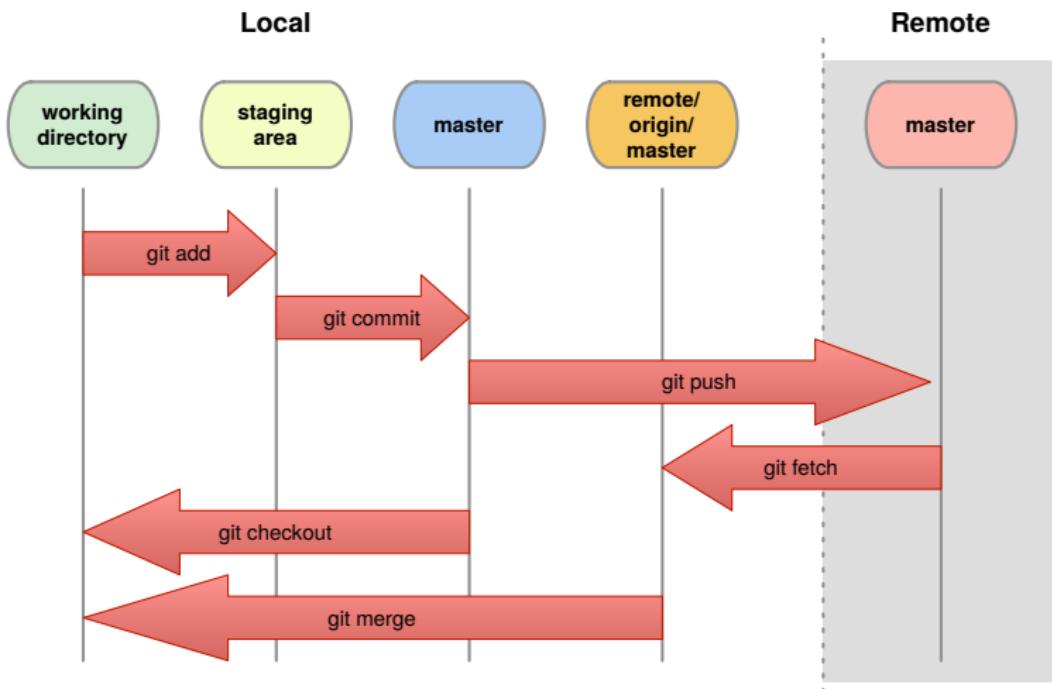
multi+remote/central: Fetch

```
git fetch
```

▶ Updates origin master from remote master
▶ local master, staging area and working dir not changed
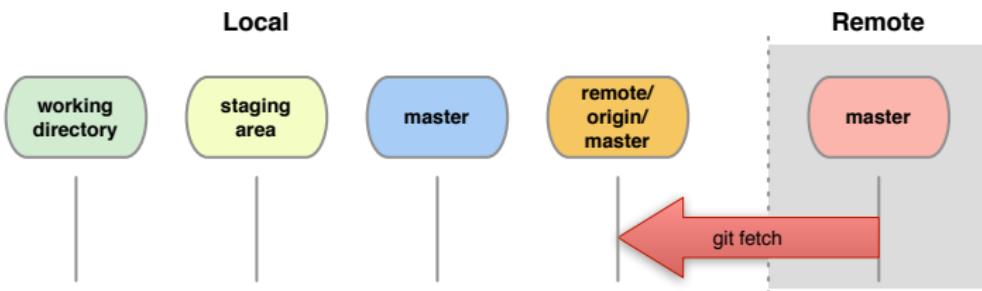
multi+remote/central: Merge

```
git merge
```

- combines changes from both sources
- Warning: can generate *conflicts*!
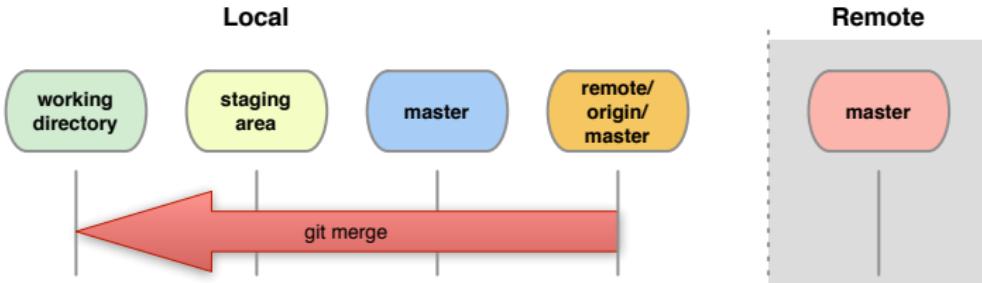

```
git fetch + git merge = git pull
```

multi+remote/central: Conflicts

<div align="center" style="color:red">Conflict!</div>

```
...
<<<<<<< yours:sample.txt
Conflict resolution is hard;
let's go shopping.
=======
Git makes conflict resolution easy.
>>>>>>> theirs:sample.txt
...
```

multi+remote/central: Resolving Conflicts

1. See where conflicts are:

   ```
   git diff
   ```

2. Edit conflicting lines.

3. Add changes to the staging area:

   ```
   git add file1 [...]
   ```

4. Commit changes:

   ```
   git commit -m "Conflicts solved."
   ```

multi+remote/central: Push

<div align="center">

`git push`

</div>

- ► Updates *remote master*.
- ► Requires `fetch+merge` first.
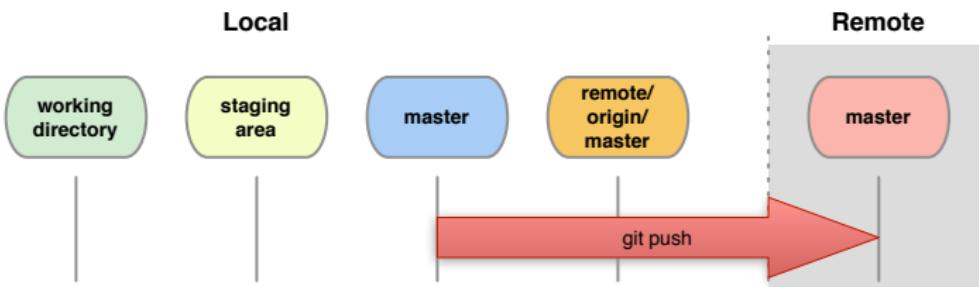
Outline

Lessons Learned

- pushing to a central server can be promlematic
  $\rightarrow$ a setup where everybody pulls can help here
- be careful, what you commit
  (no git add *)

Reference: Setting up a central remote repository.

access to repository via `ssh`

On *remote* server create **bare**+**shared** repository:

- ▶ `mkdir newproject`
- ▶ set up proper *group* permissions: `chmod g+rw`**`s`** `newproject`
- ▶ `cd newproject`
- ▶ `git` `--bare` `init` `--shared=group`

Everybody clones:

`git clone ssh://remote.example.com/path/newproject`

Outline

Behind the Scenes: Setup

```
git init; git add [...]; git commit -m "A: init"
```

a

working dir   staging area   master

Behind the Scenes: Setup

```
git init; git add [...]; git commit -m "A: init"
```

Behind the Scenes: Setup

```
git commit -am "B"
```

Behind the Scenes: Setup

```
git commit -am "C"
```
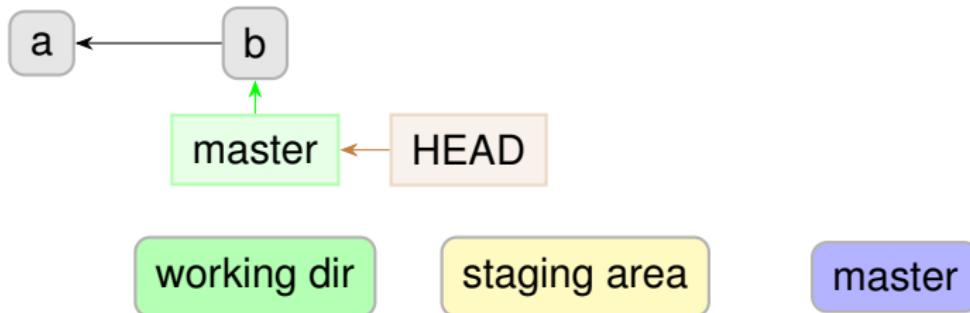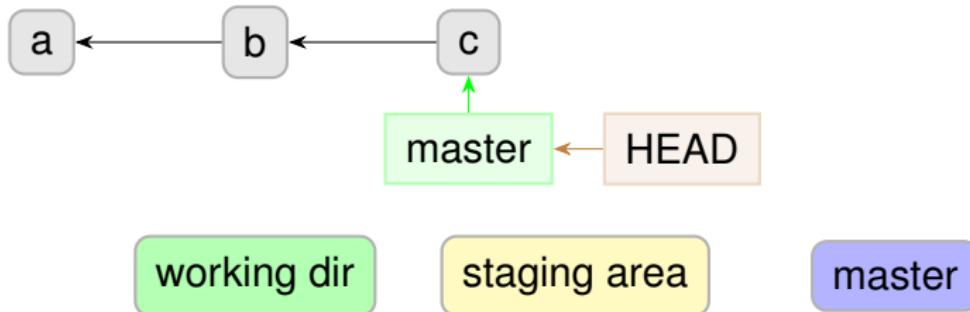
Behind the Scenes: Branches

`git branch devel`
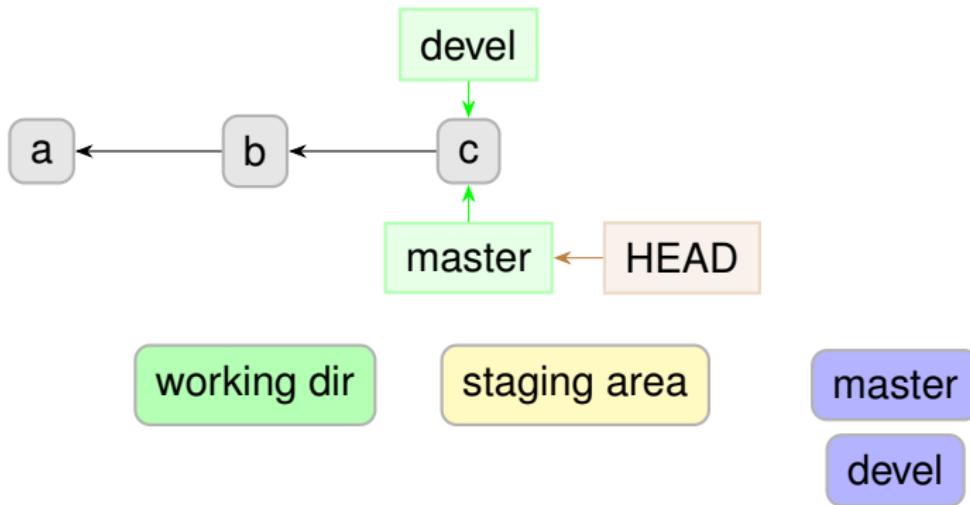
Behind the Scenes: Branches

`git checkout devel`
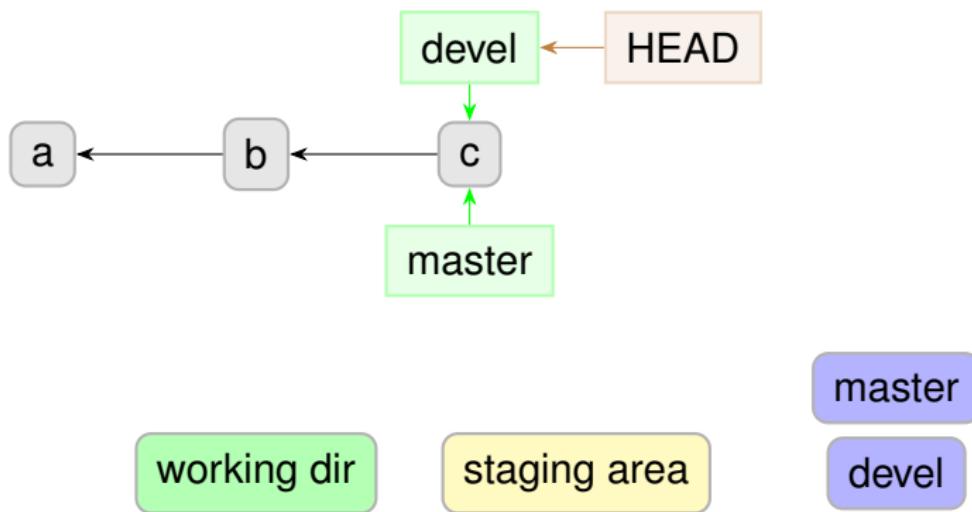
Behind the Scenes: Branches

```
git commit -am "D"
```

# Behind the Scenes: Branches

```
git commit -am "E"
```

Behind the Scenes: Branches

`git checkout master`
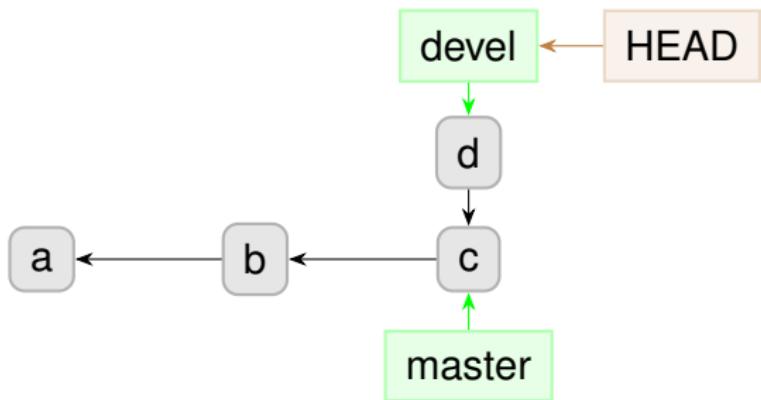
working dir   staging area   master

                                                     devel
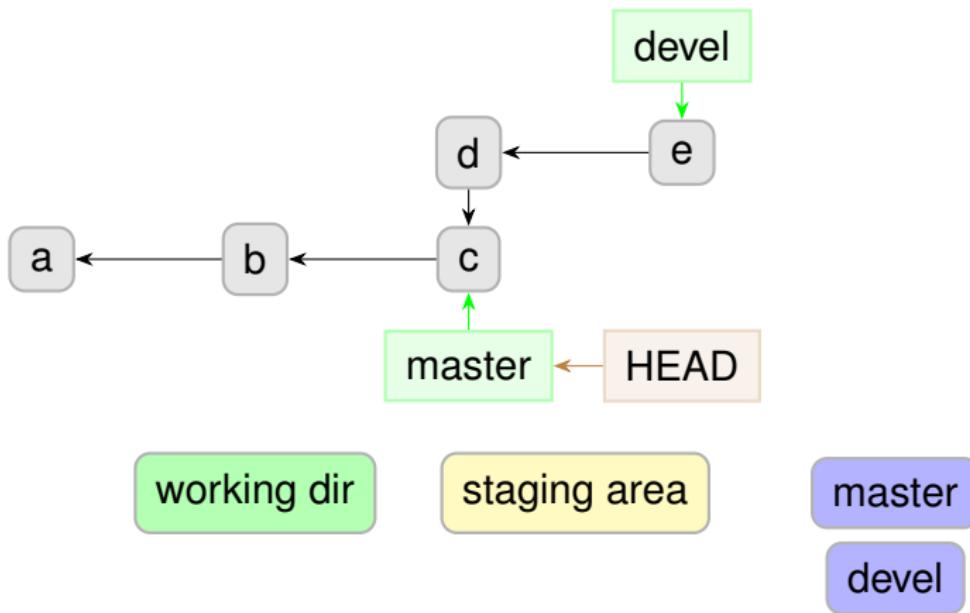
Behind the Scenes: Branches

```
git commit -am "F"
```

Behind the Scenes: Branches

`git merge devel`

# Behind the Scenes: Setup

```
git commit -am "C"
```
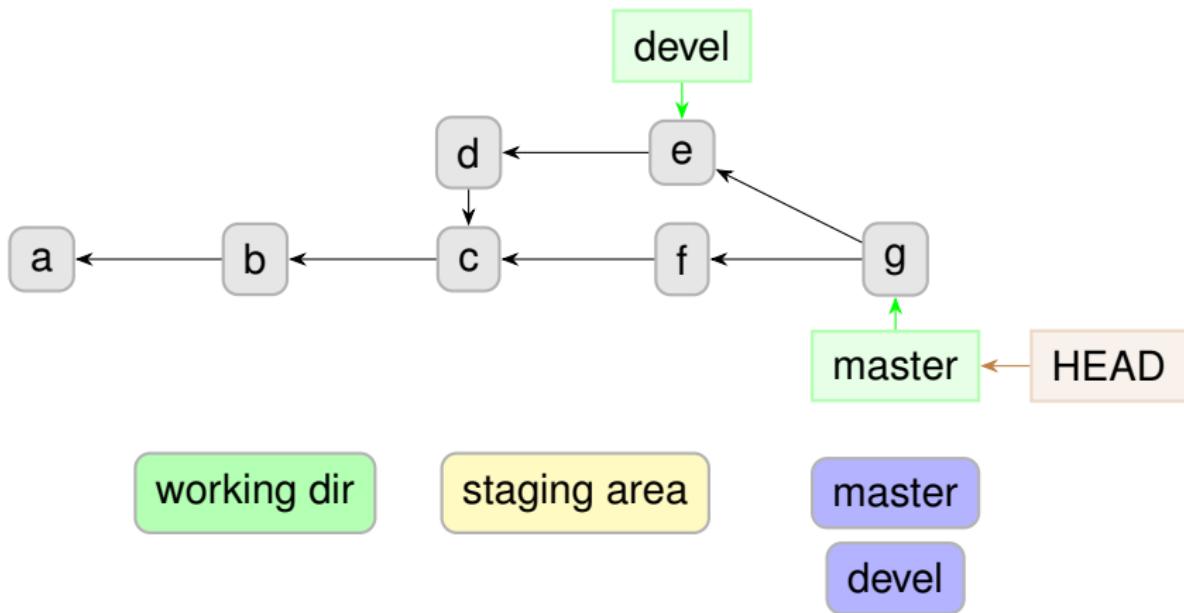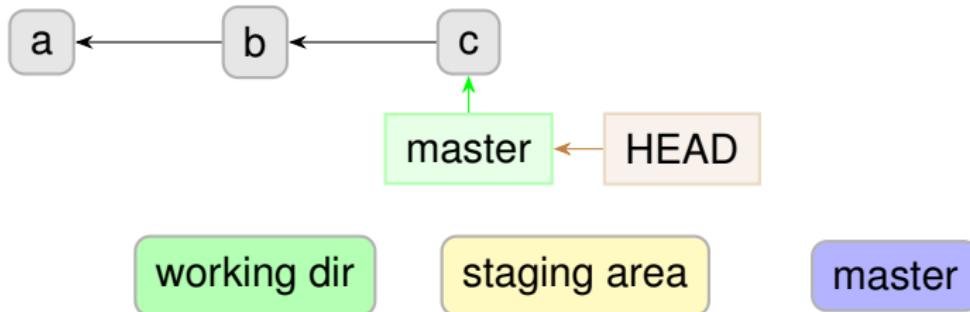
Behind the Scenes: Tags

```
git tag [-m "my message"] v1.0
```
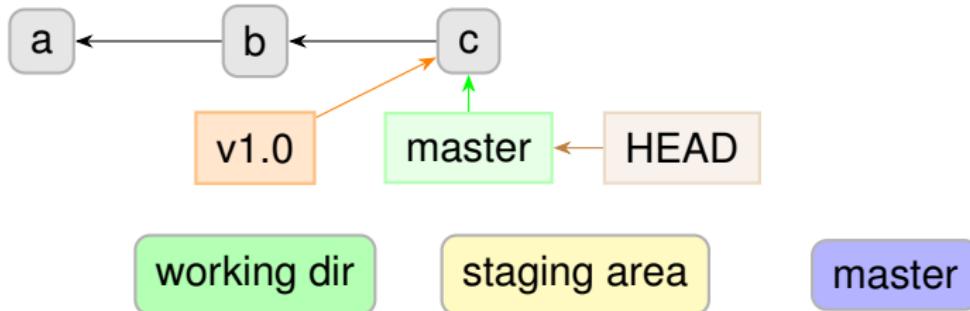
Behind the Scenes: Tags
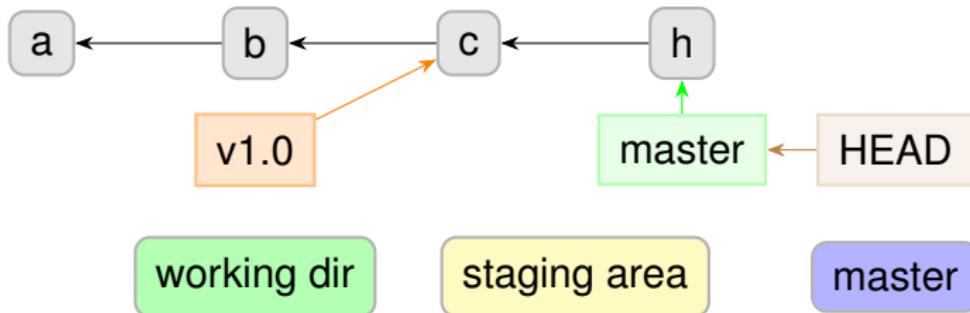
`git commit -am "H"`

Behind the Scenes: Tags

$$\texttt{git commit -am "H"}$$

to share tags: `git push origin <tag>` or `git push --tags`
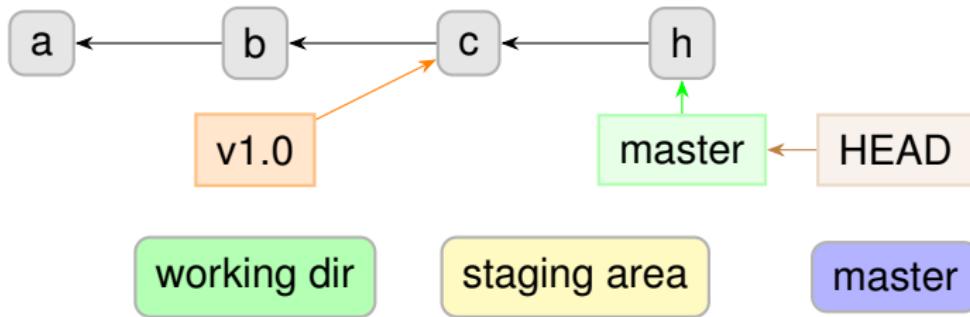
# Behind the Scenes: Detached HEAD
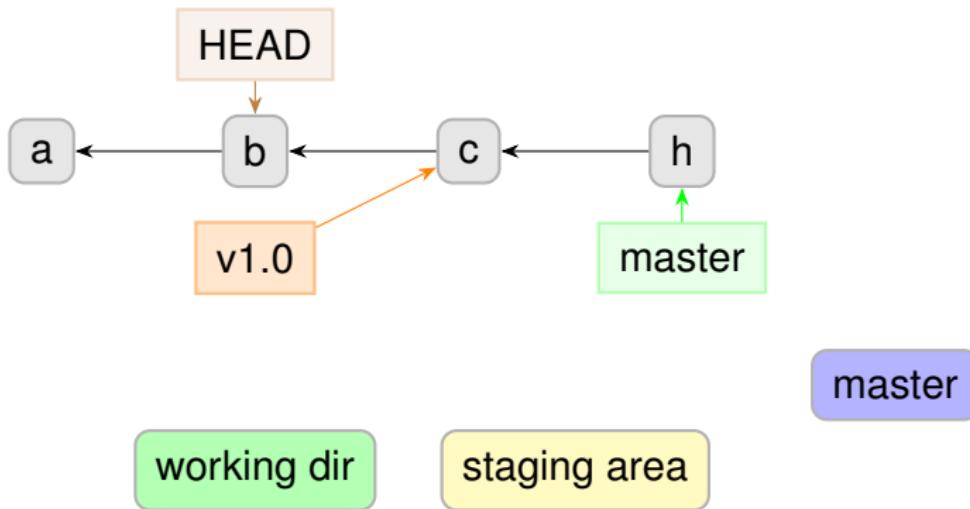
`git checkout b`

# Behind the Scenes: Detached HEAD

`git commit -am "J"`
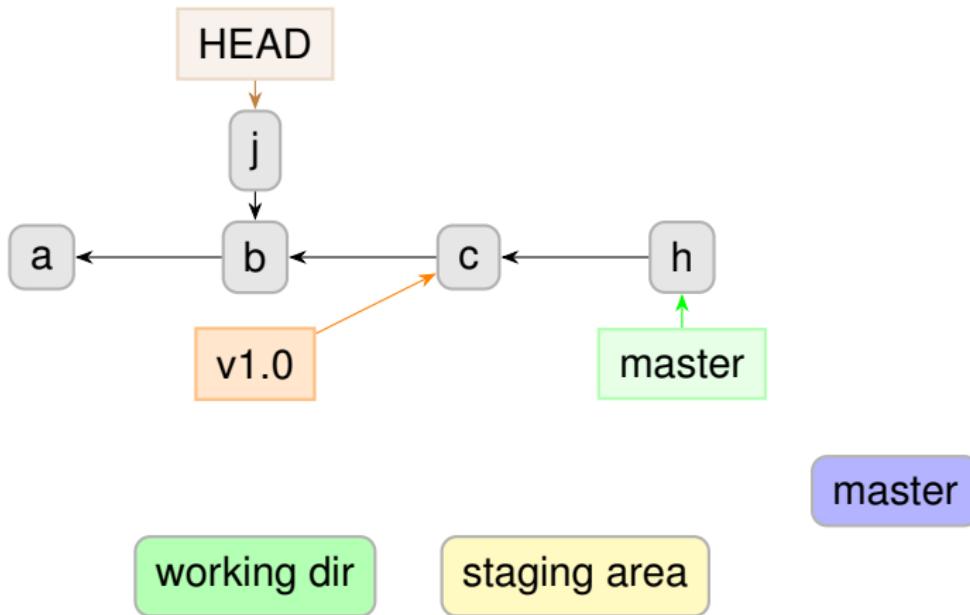
# Behind the Scenes: Detached HEAD


```
git commit -am "K"
```

Behind the Scenes: Detached HEAD

`git checkout master`

Behind the Scenes: Detached HEAD
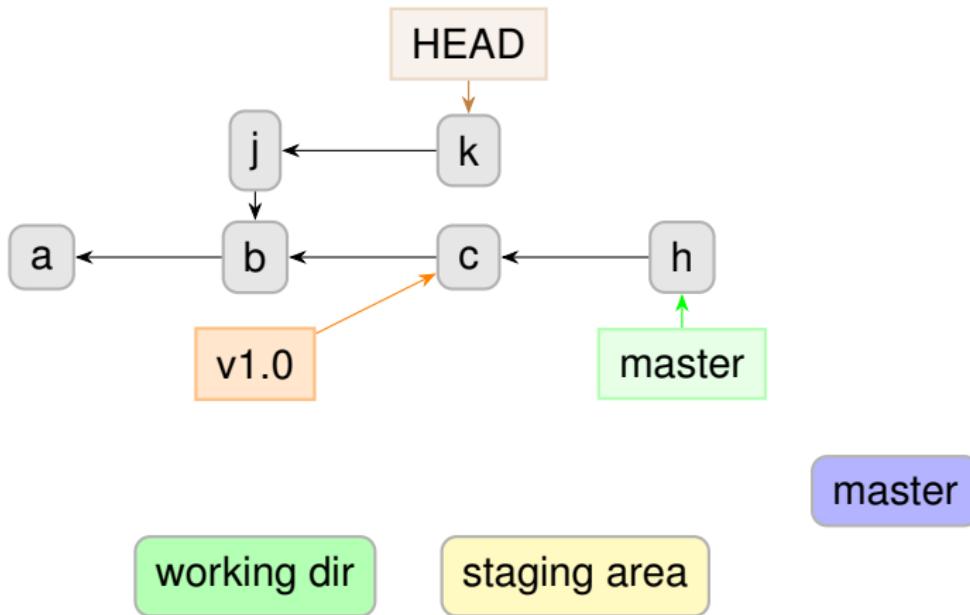
Behind the Scenes: Detached HEAD

`git checkout -b devel`

Behind the Scenes: Detached HEAD

`git checkout master`

Questions?

Understanding how git works:

- ▶ git foundations, by Matthew Brett:
  http://matthew-brett.github.io/pydagogue/foundation.html
- ▶ The git parable, by Tom Preston-Werner: http://tom.preston-werner.com/2009/05/19/the-git-parable.html

Excellent guides:

- ▶ "Pro Git" book: https://git-scm.com/book/en/v2 (FREE)
- ▶ git magic:
  http://www-cs-students.stanford.edu/~blynn/gitmagic/

Behind the Scenes: Rebase

`git checkout devel`
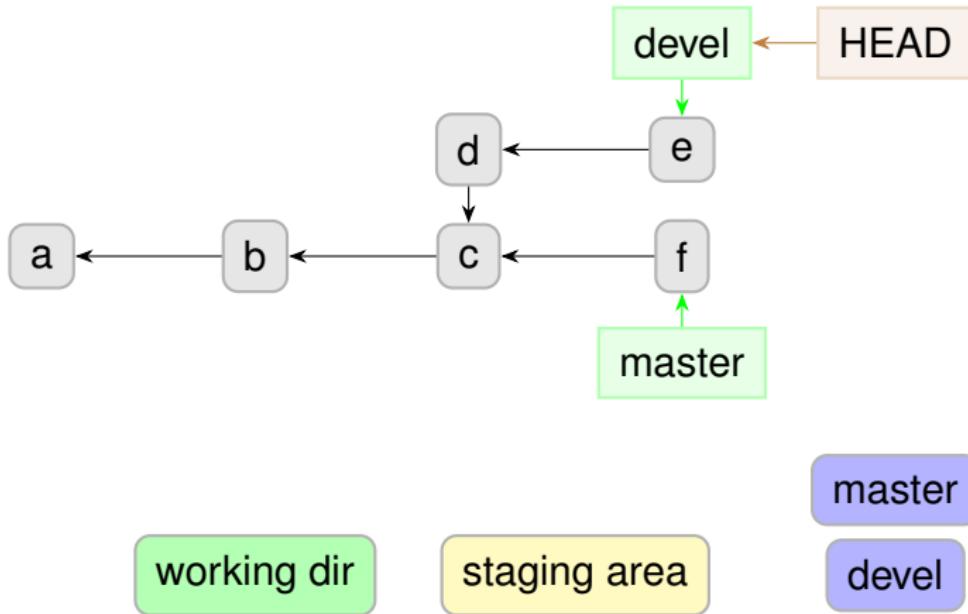
Behind the Scenes: Rebase

`git rebase master`