

git Tutorial

Nicola Chiapolini

Physik-Institut
University of Zurich

June 6, 2016

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

Introduction

Single developer + local repository

Demo/Exercise: single+local

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

Survey: Version Control

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

Uses for git

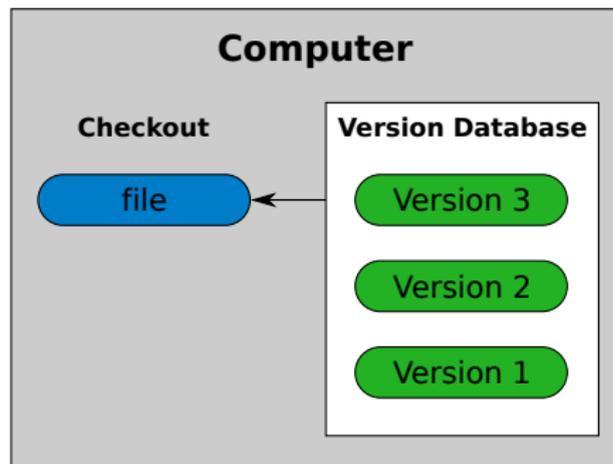
“*Version control* is a system that records changes to a file or set of files over time so that you can recall specific versions later.”

– <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](#))

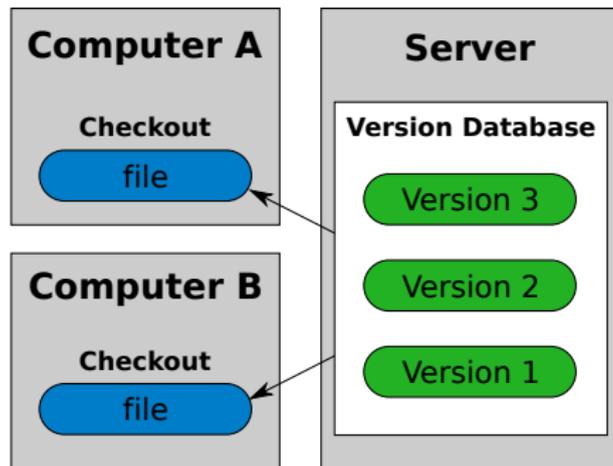
Version Control: Local



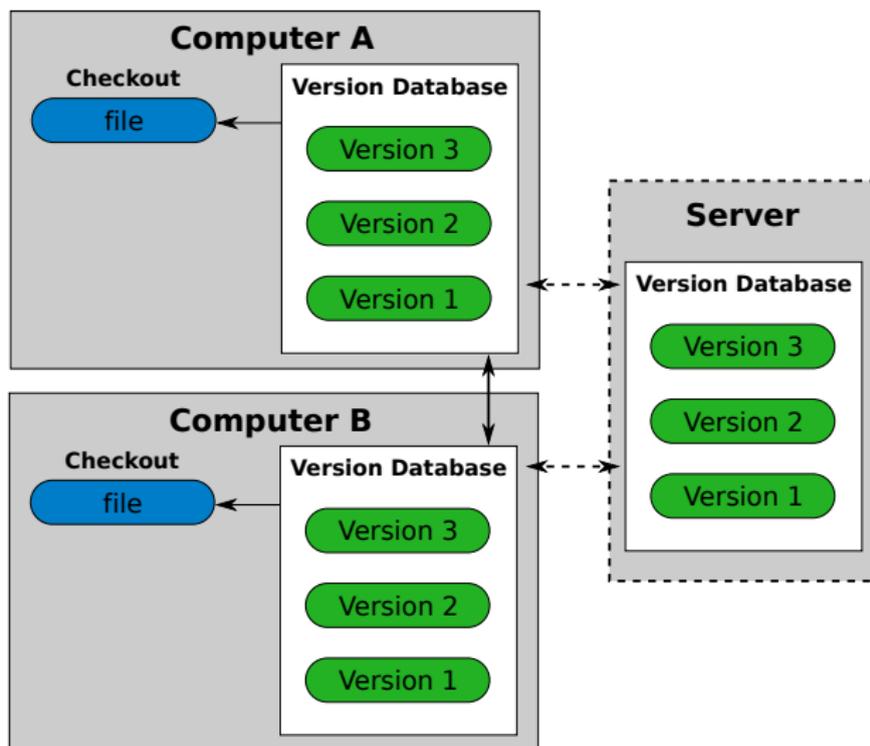
checkout working directory

version database repository

Version Control: Central



Version Control: Distributed



git: Help

```
usage: git [OPTIONS] COMMAND [ARGS]
```

The most commonly used git commands are:

```
add          Add file contents to the index
commit       Record changes to the repository
diff         Show changes between commits, commit and working tree, etc
...
```

```
git help <command>
```

```
git help <concept>
```

```
git status
```

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

```
git init
```

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

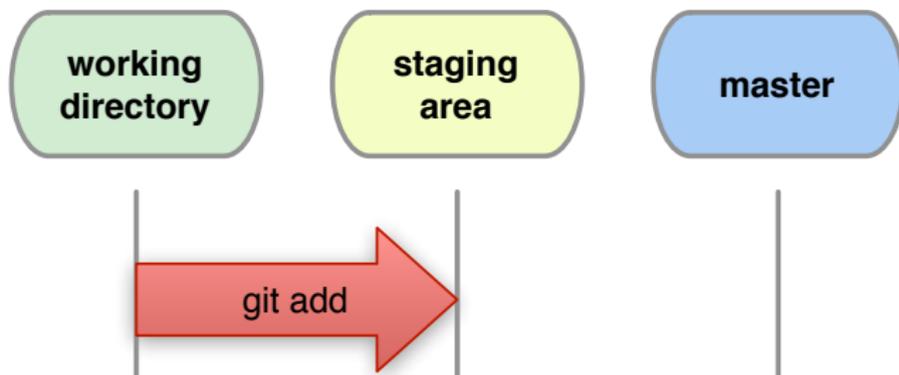


- ▶ does not change your files

single+local: Add

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

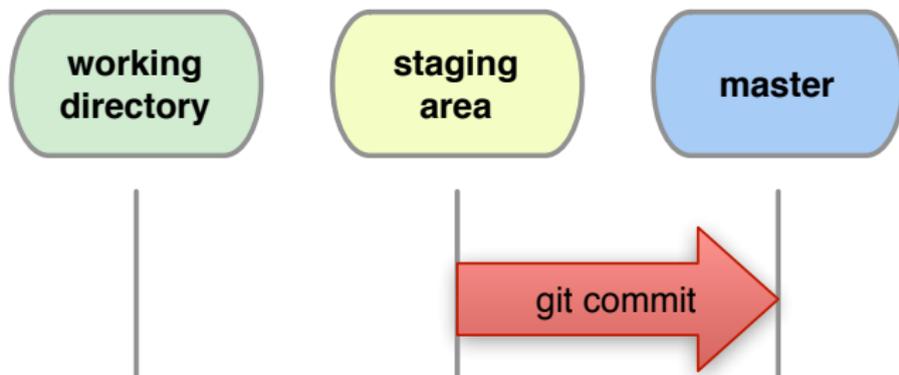
- ▶ Adds new files to be tracked by git
- ▶ Adds changes from working dir for next commit
- ▶ DOES NOT add info on file permissions other than *exec/noexec*
- ▶ DOES NOT add directories *per se*.



single+local: Commit

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

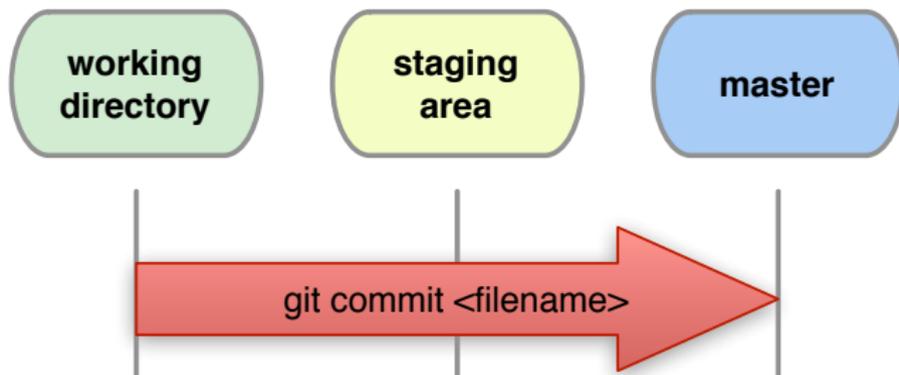
Records changes from the staging area to master.



single+local: Direct Commit

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

Records all changes of `file1`, `file2` to master.



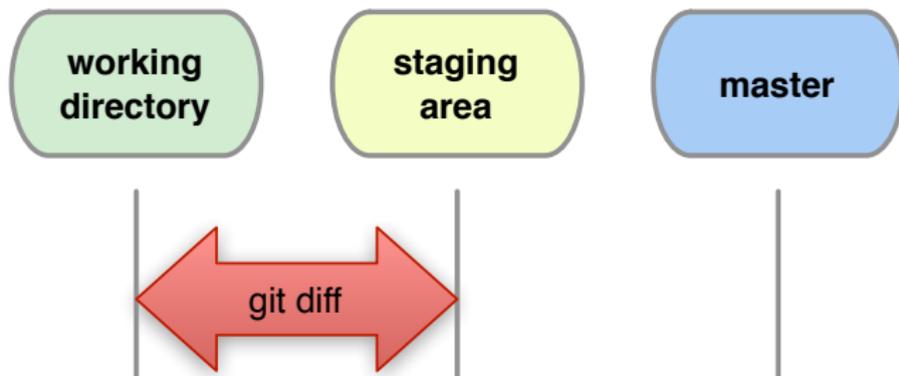
```
git commit -a[m "Commit message."]
```

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

single+local: Diff

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

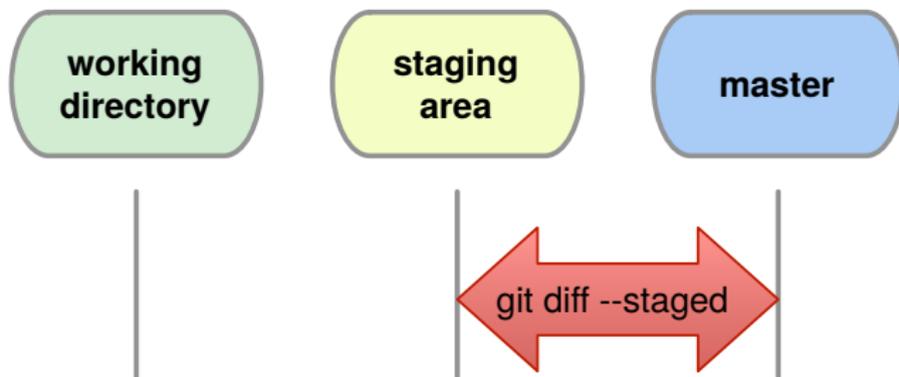
Shows changes between *working directory* and *staging area*



single+local: Diff Staged

How do I see what is staged?

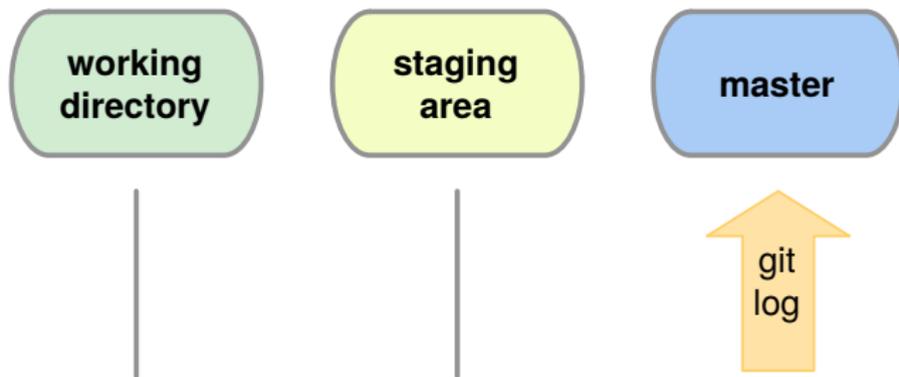
`git diff --staged` shows differences between staging area and last commit.



single+local: Commit History

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

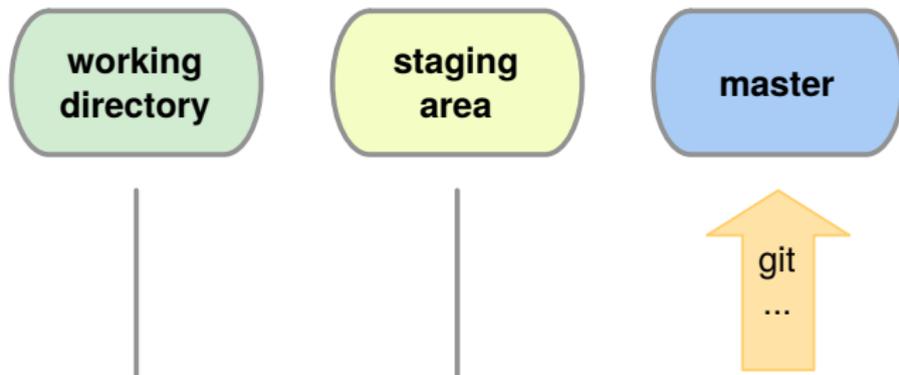
Shows the history of a file or branch.



single+local: Old Changes

```
git diff <commit A> <commit B>  
git show <commit>
```

Shows the changes stored in commits.



single+local: Graphic Logs

qgit (or gitg or ...)

GUI to browse the git repository.

merge rsync://rsync.kernel.org/pub/scm/linux/kernel/git/c...
Merge rsync://rsync.kernel.org/pub/scm/linux/kernel/git/c...
[PATCH] USB: ftdi_sio: avoid losing received data in tty-
[PATCH] USB: fix ub issues
[PATCH] PCI Hotplug: fix CPCI reference counting bug
[A64] Fix race condition in the rt_sigprocmask fastcall
Merge master.kernel.org:/home/mrk/linux-2.6-arm
[PATCH] sg traverse fix for __atapi_pio_bytes()
[PATCH] sata_sil: Fix FIFO PCI Bus Arbitration kernel oo
[PATCH] ARM: Remove zero-byte sized file
Merge rsync://rsync.kernel.org/pub/scm/linux/kernel/git/daven...
[PKT_SCHED]: Fix numeric comparison in meta ematch

SHA1 ID: Find Ex

Author: Pete Zaitcev <zaitcev@redhat.com> 2005-06-06 14:54:59
Committer: Greg Kroah-Hartman <gregkh@suse.de> 2005-06-09 02:38:11

[PATCH] USB: fix ub issues

This smoothes two imperfections:

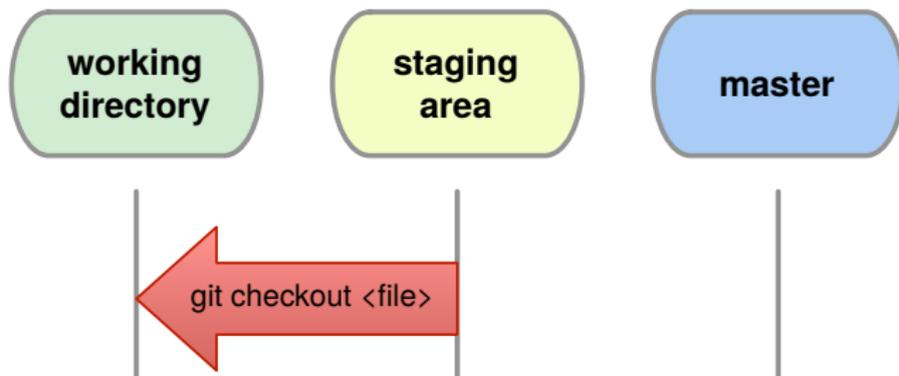
- Increase number of LUNs per device from 4 to 9. The best solution would be to remove this limit altogether, but that has to wait until the time when more than 26 hosts are allowed.
- Replace mdelay with msleep in a probing routine.

Signed-off-by: Pete Zaitcev <zaitcev@yahoo.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

All files
drivers/block/ub.c

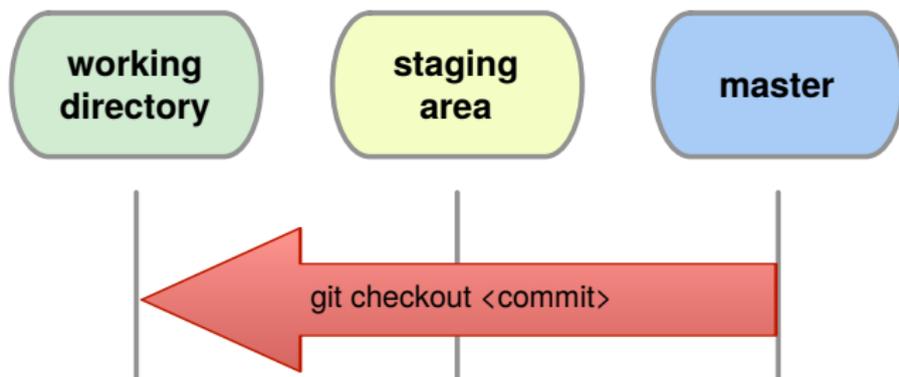
single+local: Changing Version

```
git checkout <file|commit>
```



single+local: Changing Version

```
git checkout <file|commit>
```



single+local: (Re)move.

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

```
git rm <filename>
```

```
git mv <oldname> <newname>
```

Remember to `commit` these changes!

```
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

Introduction

Single developer + local repository

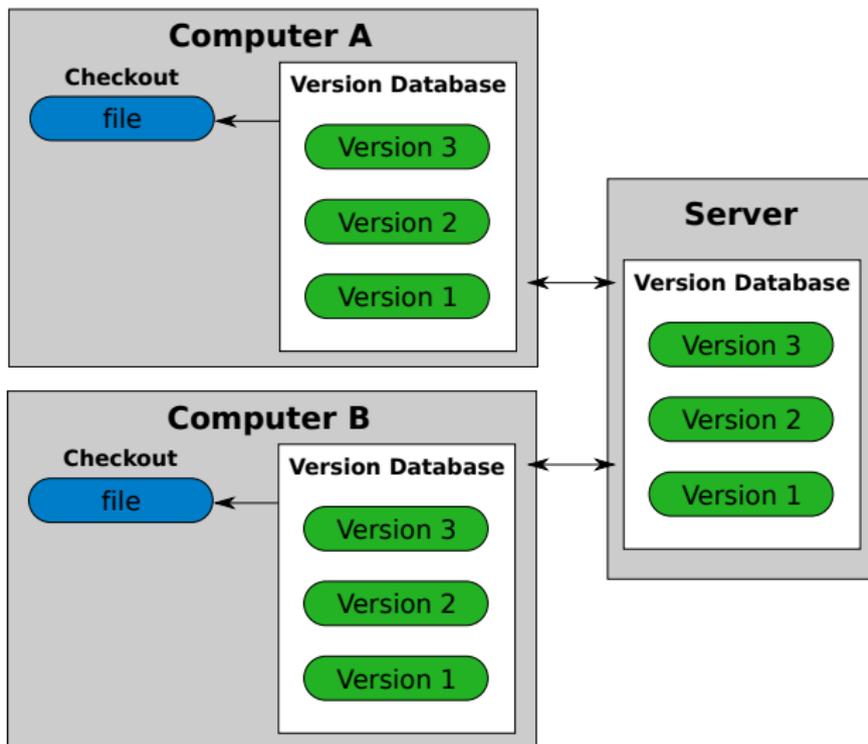
Demo/Exercise: single+local

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

multi+remote/central: Setup

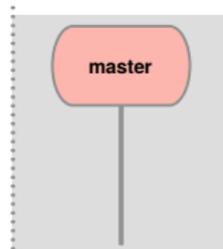


multi+remote/central: Clone

```
git clone <URL>
```

Creates **two** local copies of the **whole** remote repository.

Remote (Server)



Version Database

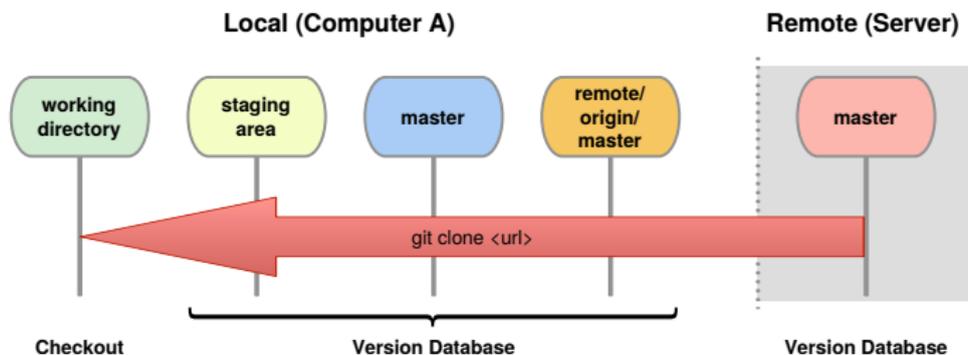
Hint

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

multi+remote/central: Clone

```
git clone <URL>
```

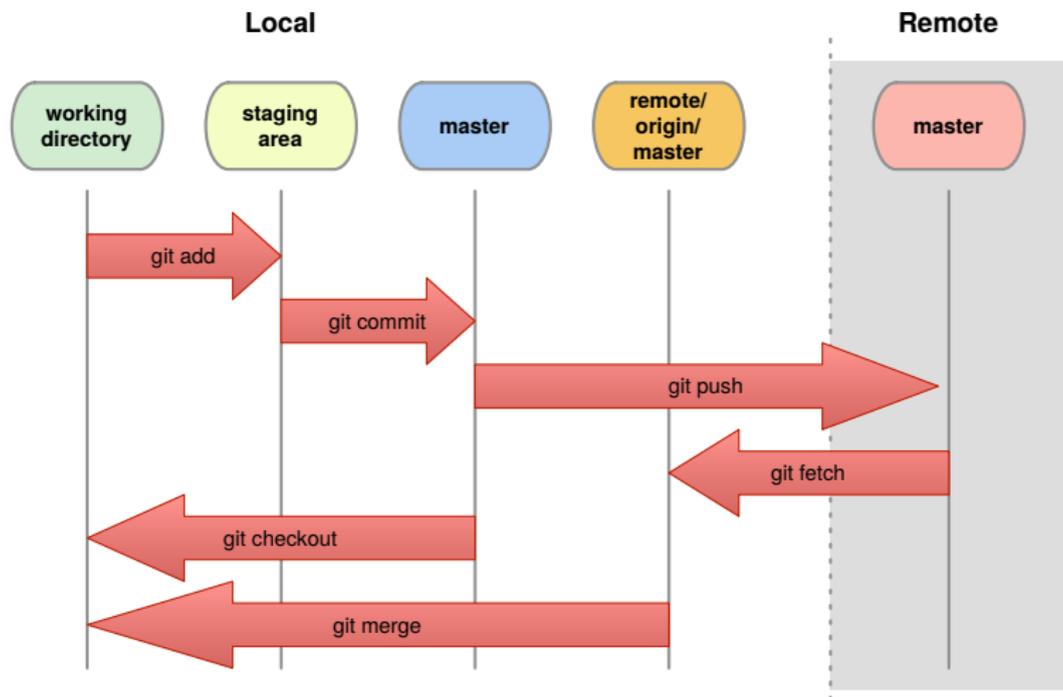
Creates **two** local copies of the **whole** remote repository.



Hint

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

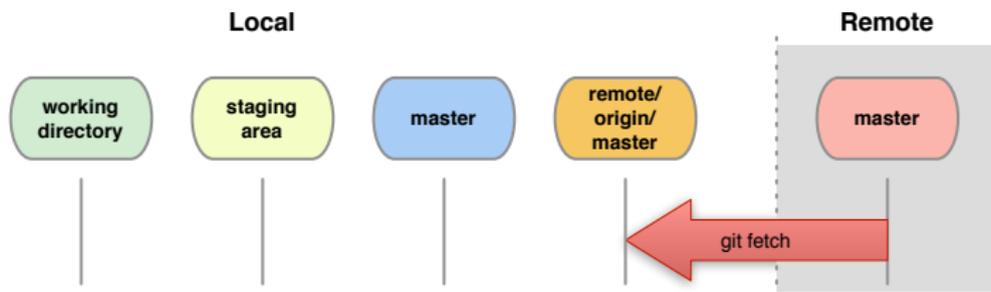
multi+remote/central: Commands



multi+remote/central: Fetch

```
git fetch
```

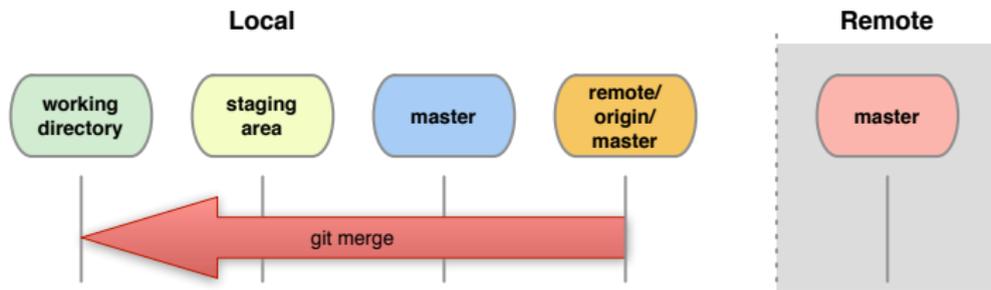
- ▶ Updates origin master from remote master
- ▶ local master, staging area and working dir not changed



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

Conflict!

```
...
<<<<<<< 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 [...]
```

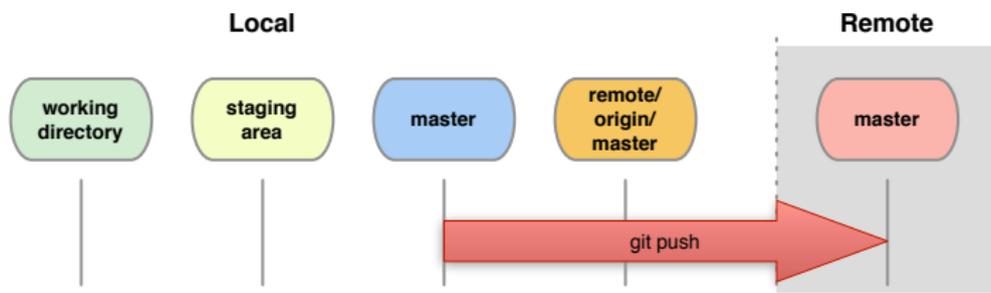
4. Commit changes:

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

multi+remote/central: Push

git push

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



Outline

Introduction

Single developer + local repository

Demo/Exercise: single+local

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

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+rws newproject`
- ▶ `cd newproject`
- ▶ `git --bare init --shared=group`

Everybody clones:

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

Outline

Introduction

Single developer + local repository

Demo/Exercise: single+local

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

Behind the Scenes: Setup

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

a

working dir

staging area

master

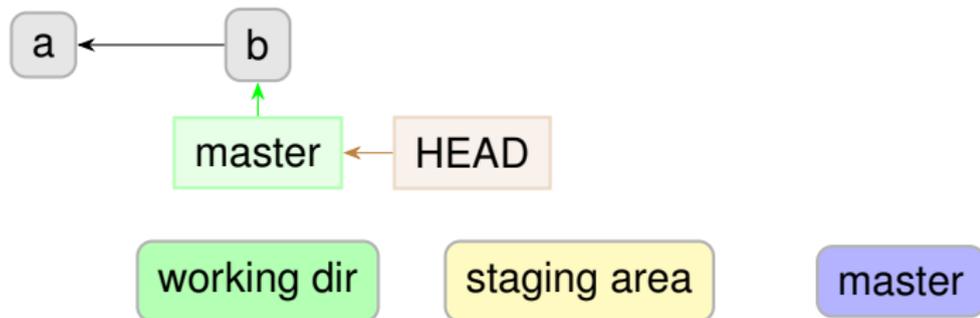
Behind the Scenes: Setup

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



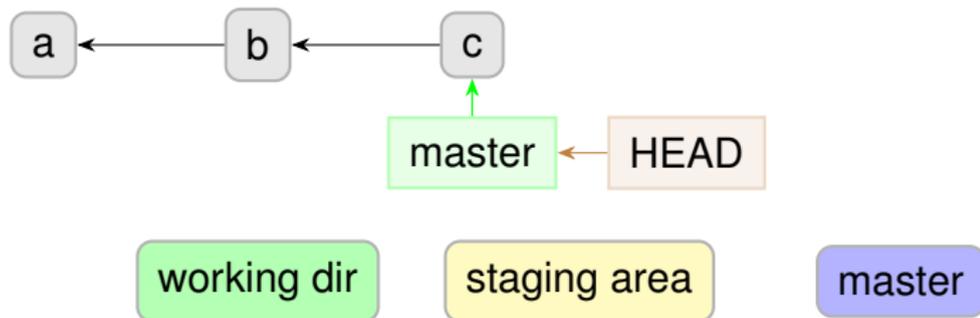
Behind the Scenes: Setup

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



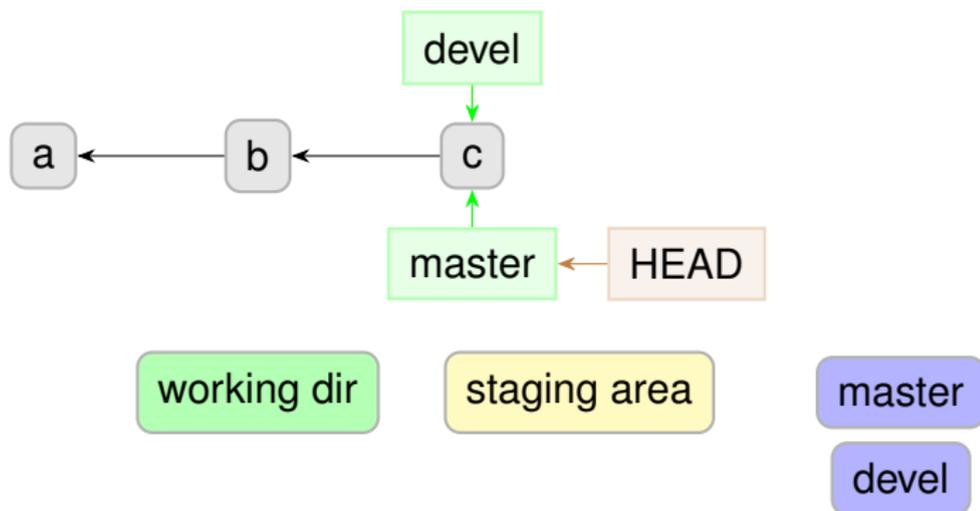
Behind the Scenes: Setup

```
git commit -am "C"
```



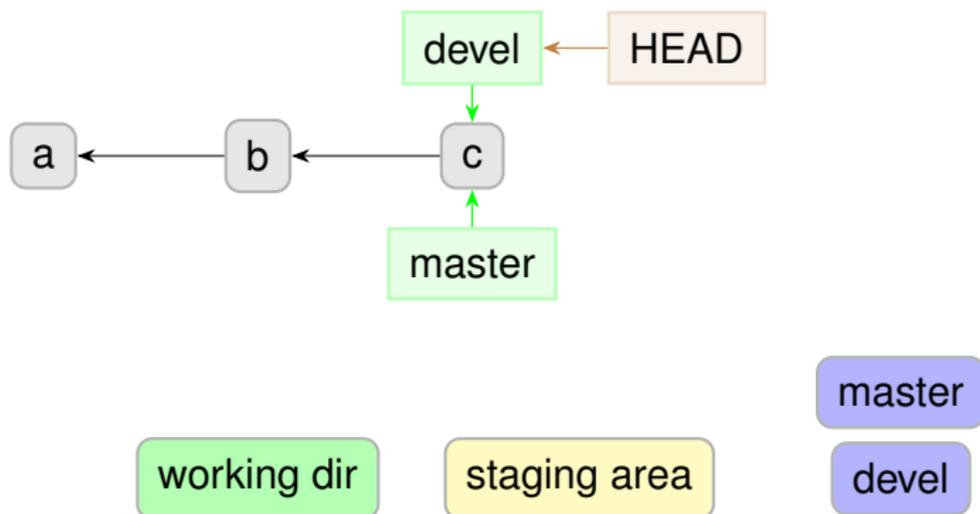
Behind the Scenes: Branches

git branch devel



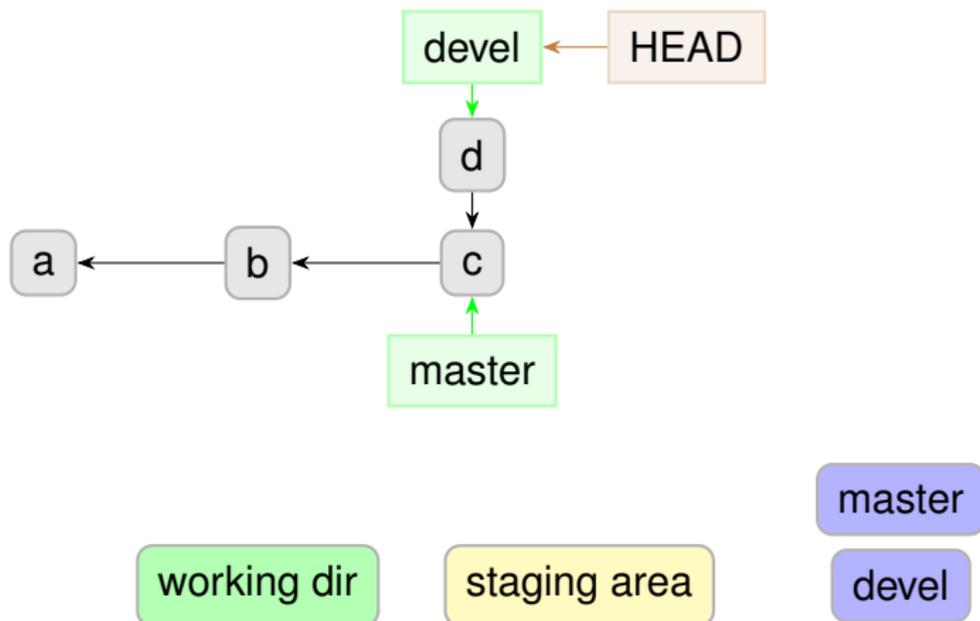
Behind the Scenes: Branches

git checkout devel



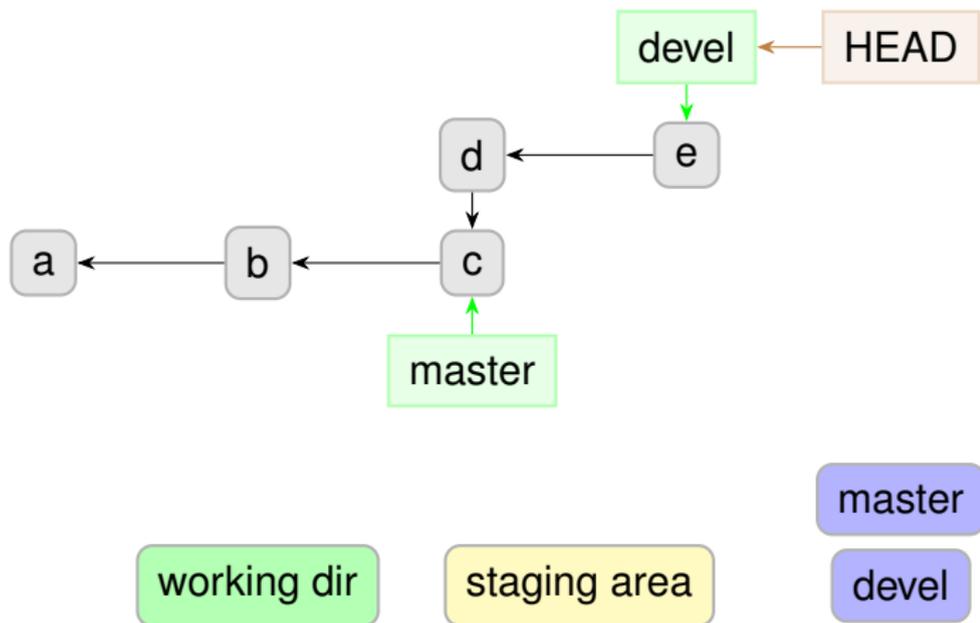
Behind the Scenes: Branches

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



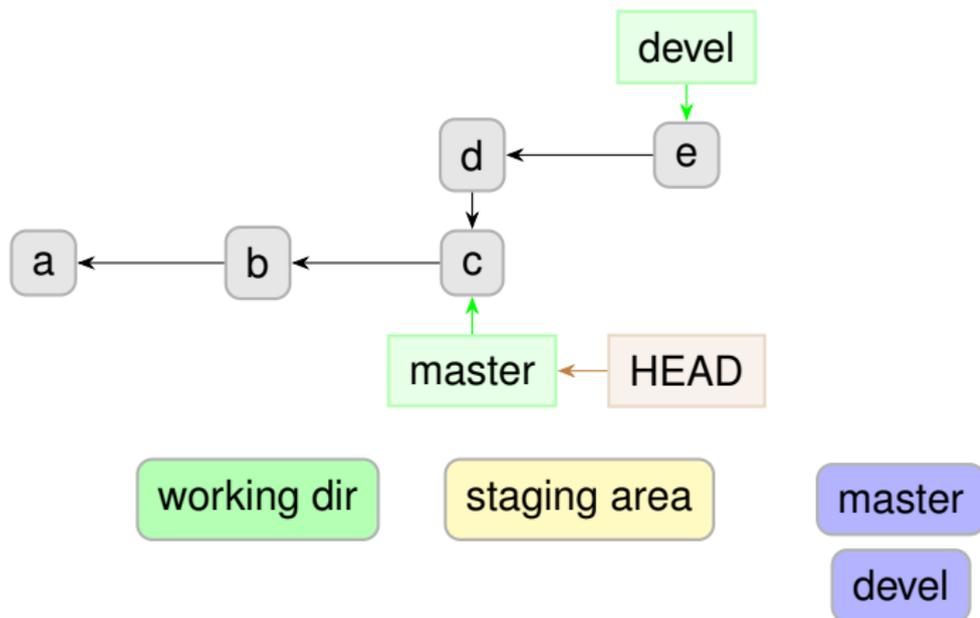
Behind the Scenes: Branches

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



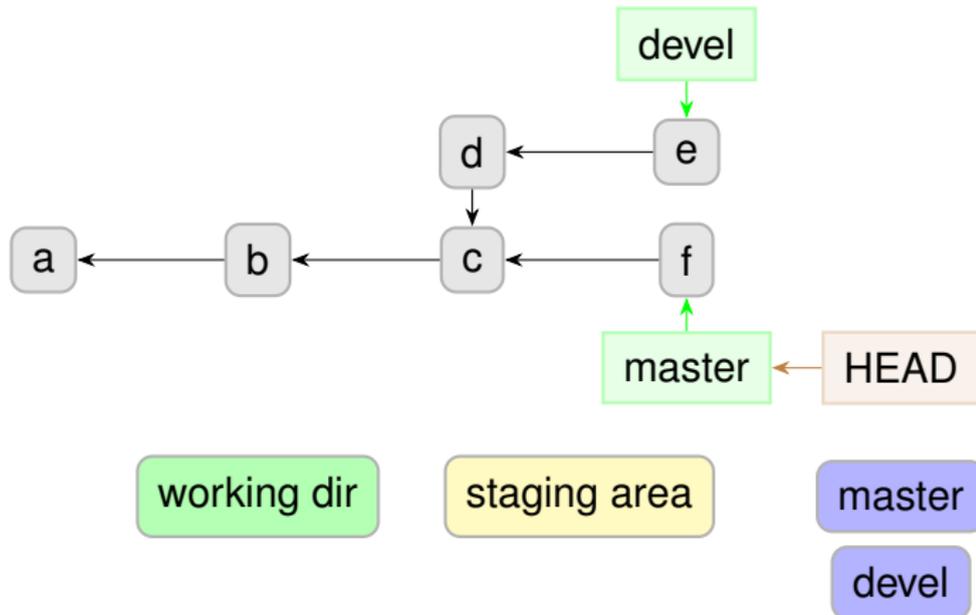
Behind the Scenes: Branches

git checkout master



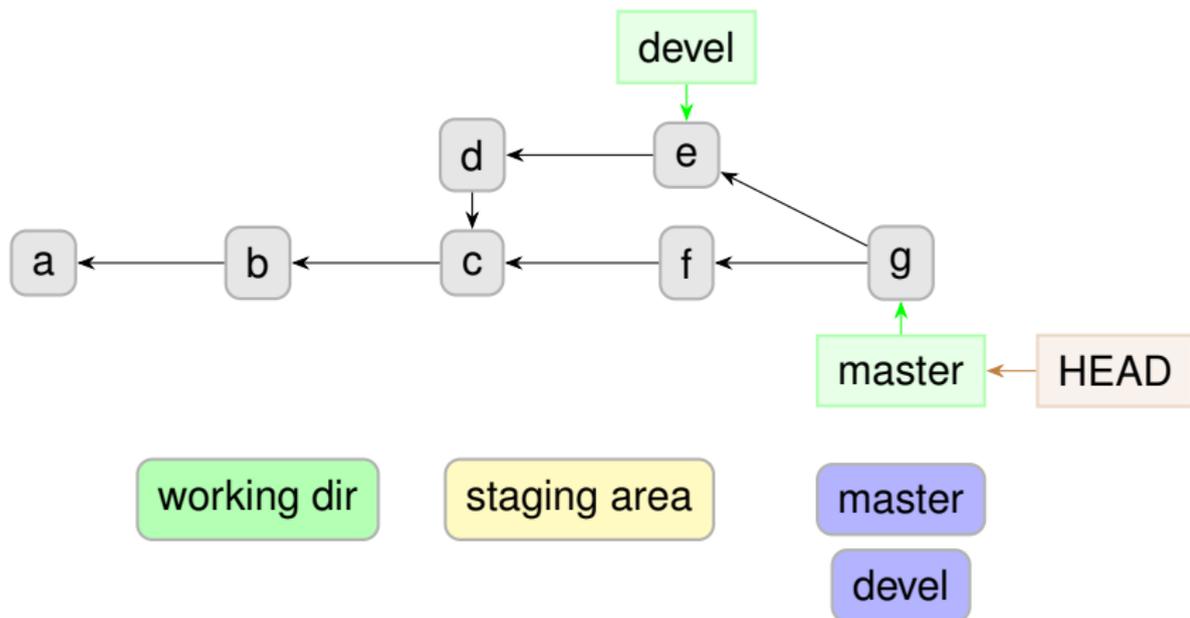
Behind the Scenes: Branches

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



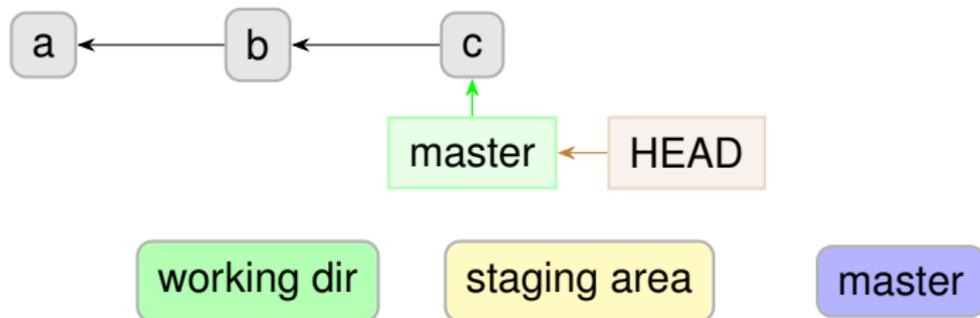
Behind the Scenes: Branches

git merge devel



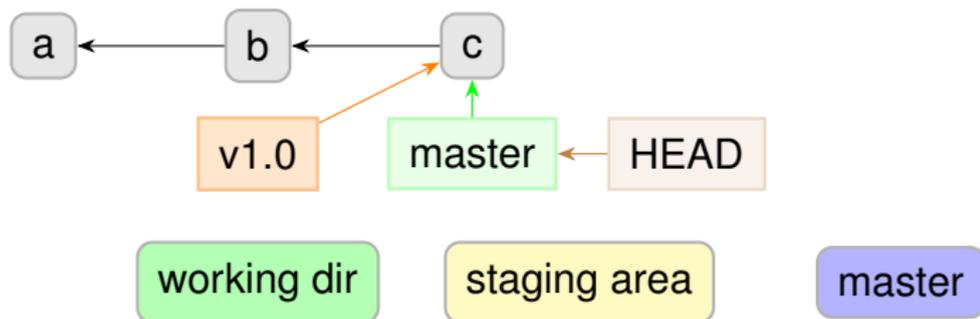
Behind the Scenes: Setup

```
git commit -am "C"
```



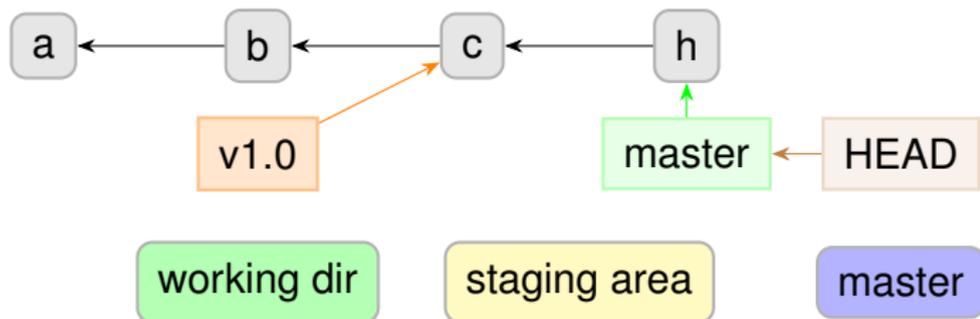
Behind the Scenes: Tags

```
git tag [-m "my message"] v1.0
```



Behind the Scenes: Tags

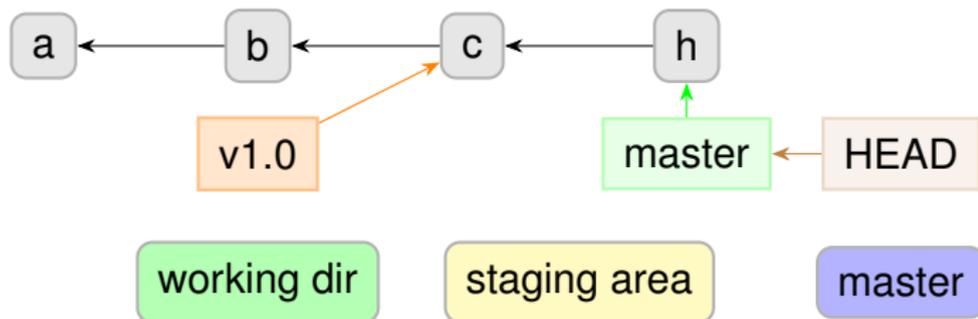
```
git commit -am "H"
```



Behind the Scenes: Tags

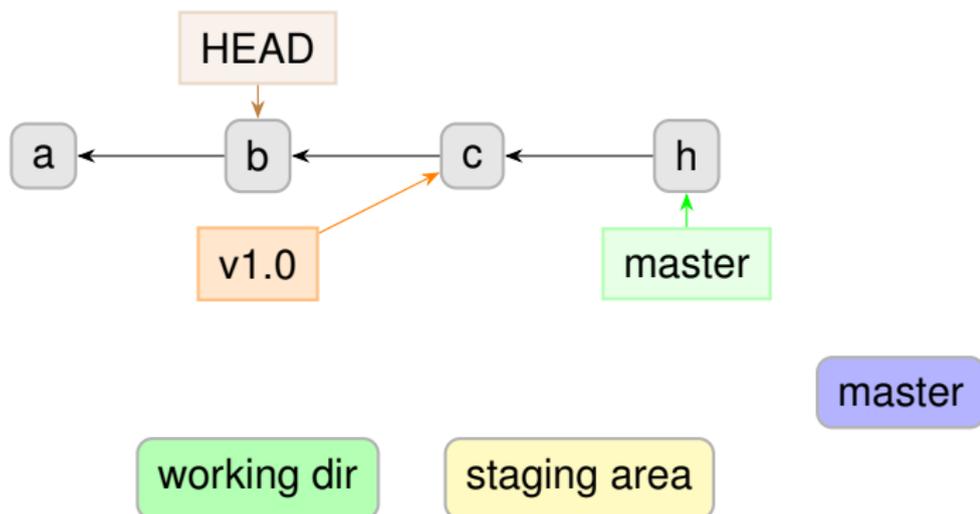
```
git commit -am "H"
```

to **share** tags: `git push origin <tag>` or `git push --tags`



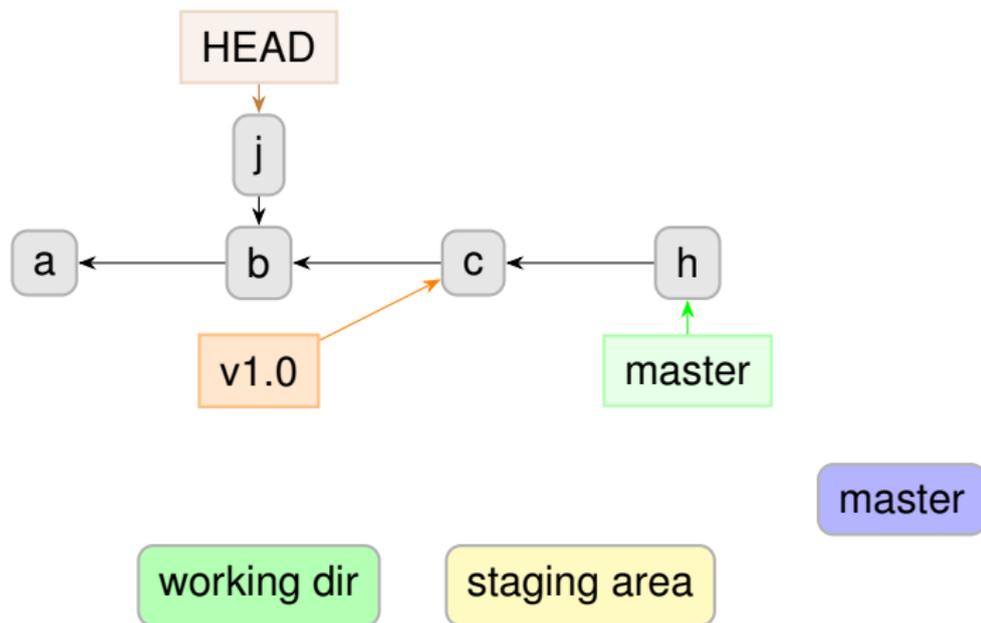
Behind the Scenes: Detached HEAD

```
git checkout b
```



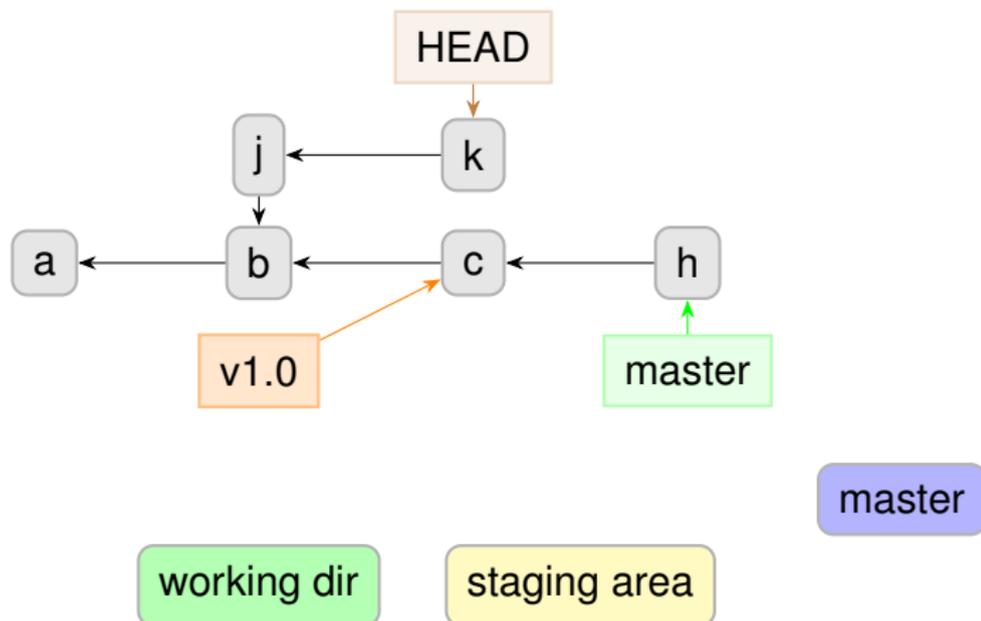
Behind the Scenes: Detached HEAD

```
git commit -am "J"
```



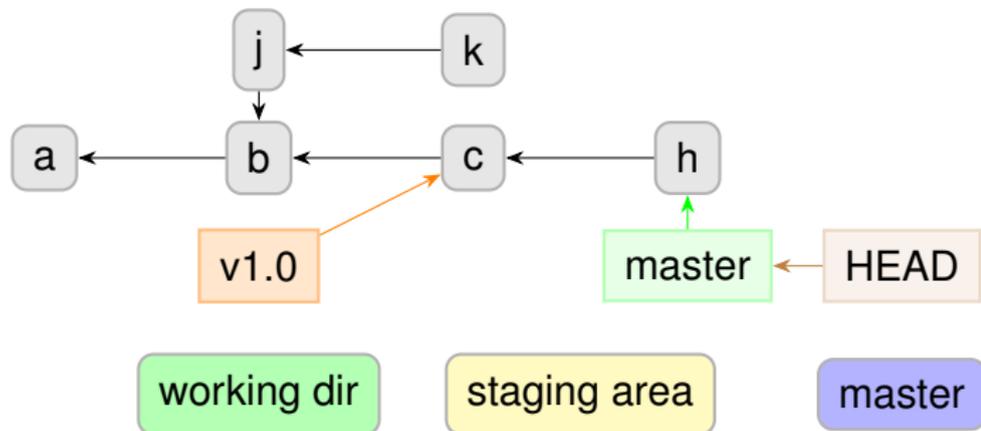
Behind the Scenes: Detached HEAD

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



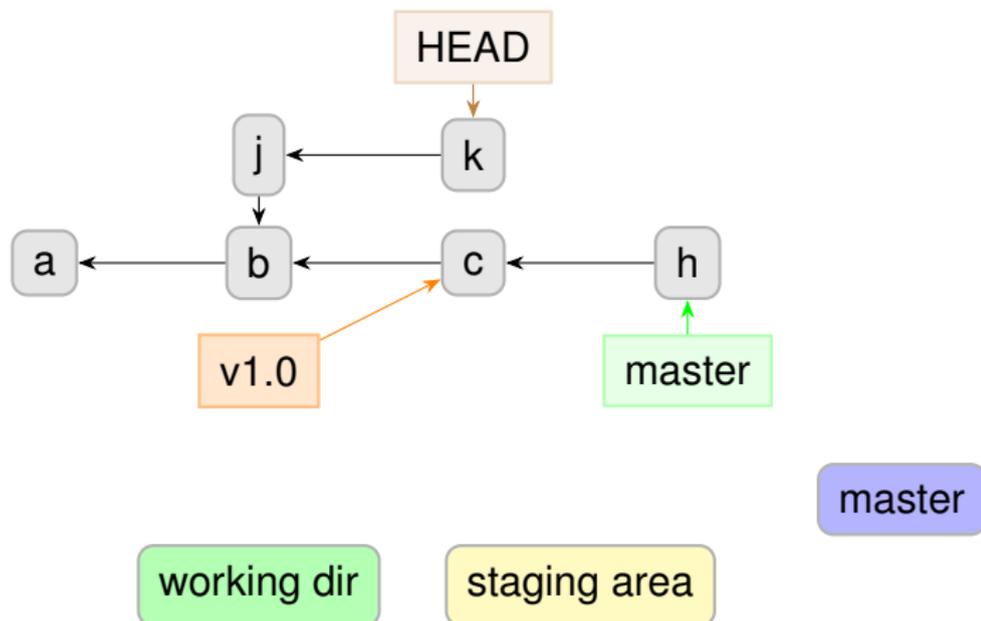
Behind the Scenes: Detached HEAD

```
git checkout master
```



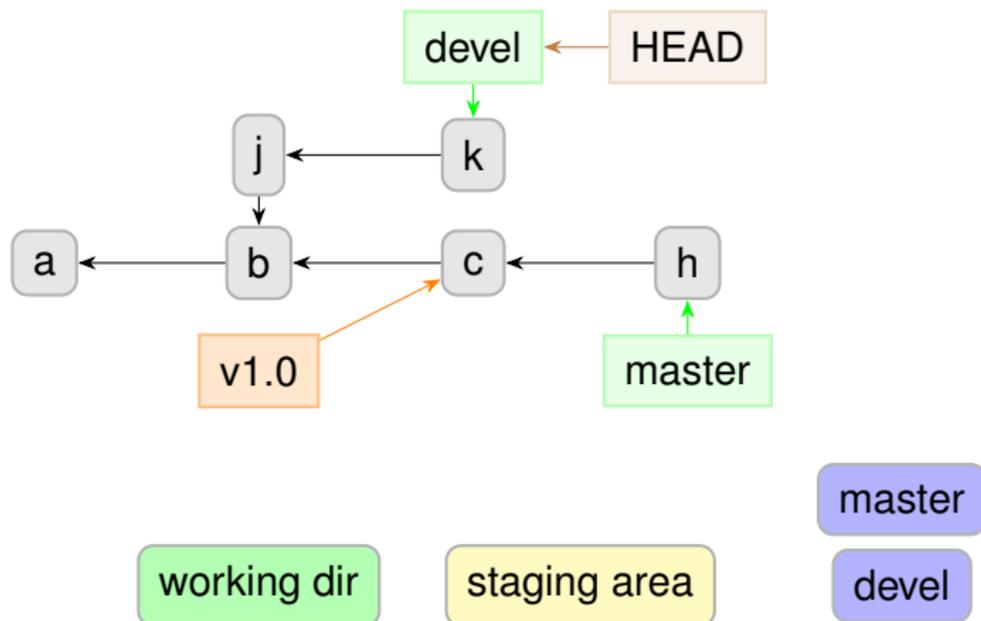
Behind the Scenes: Detached HEAD

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



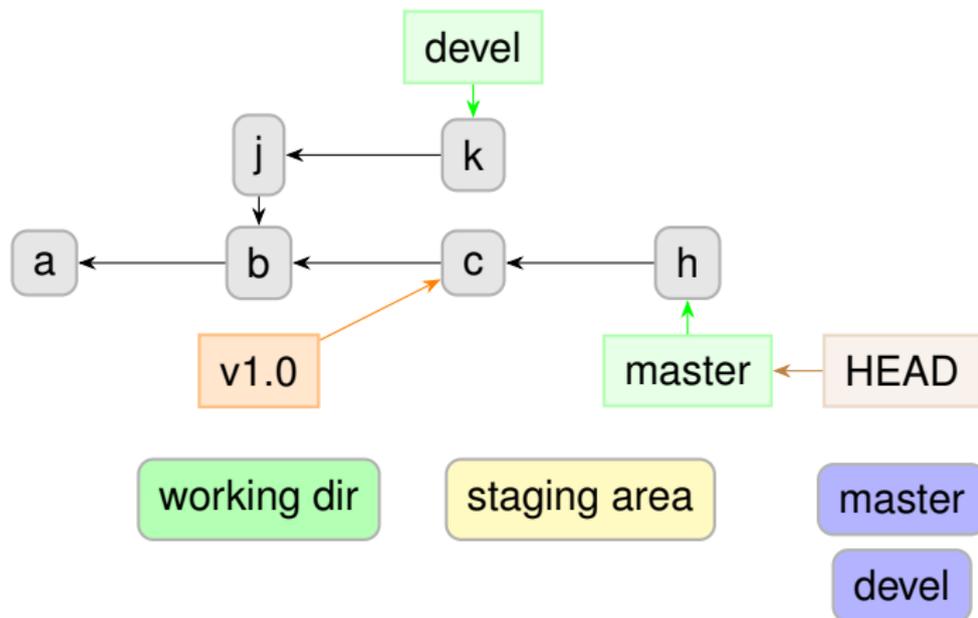
Behind the Scenes: Detached HEAD

```
git checkout -b devel
```



Behind the Scenes: Detached HEAD

git checkout master



Questions?

Understanding how git works:

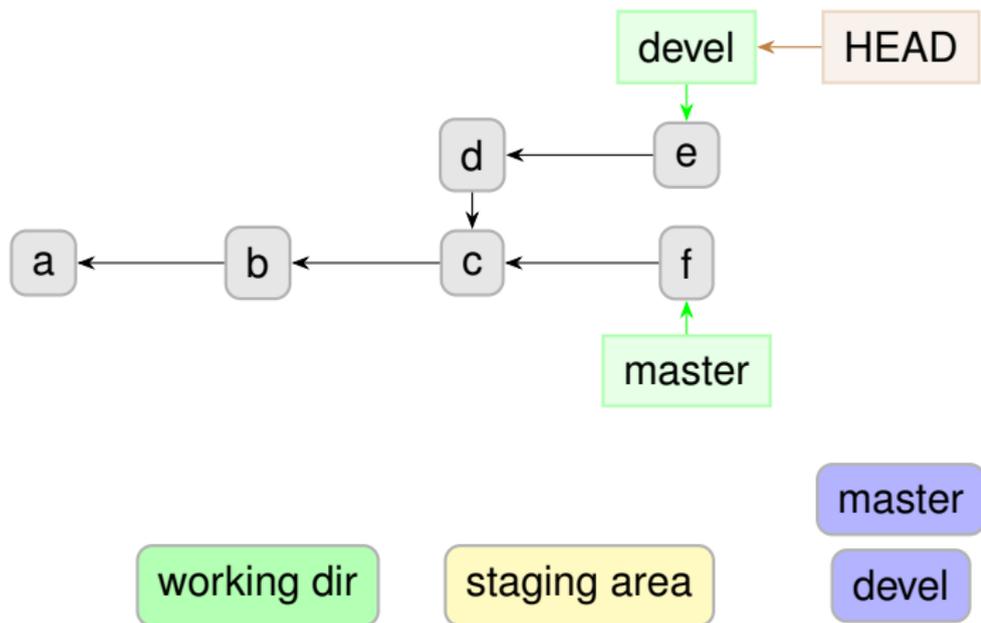
- ▶ git foundations, by Matthew Brett:
<http://matthew-brett.github.io/pydagogue/foundation.html>
- ▶ The git parable, by Tom Preston-Werner: <http://tom.preston-werner.com/2009/05/19/the-git-parable.html>

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`

