git Tutorial

Nicola Chiapolini

Physik-Institut
University of Zurich

2020-06-22

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

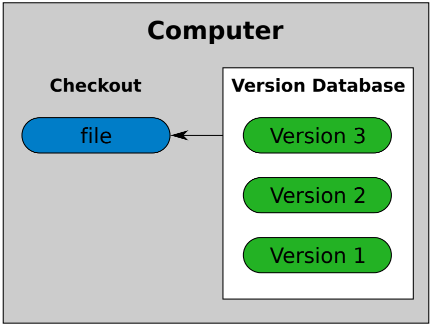
Outline

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, etckeeper)

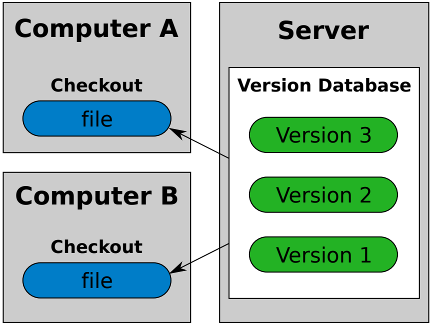
Version Control: Local



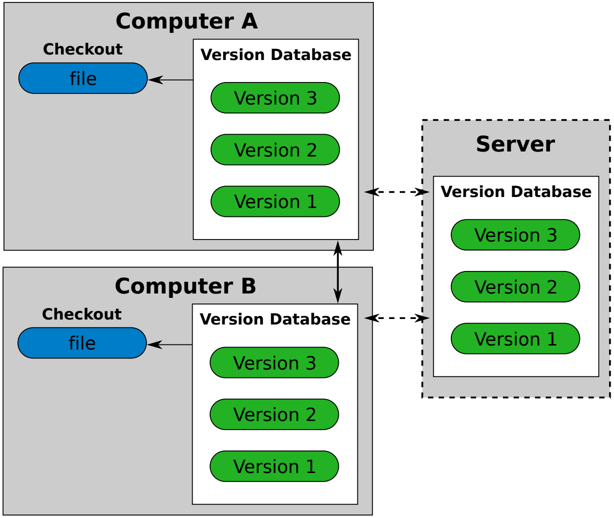
checkout	working directory
version database	repository

There is always only one version of a file present in the working directory. Version Control allows you to change that file to different versions stored in the 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.
```

The Glossary

`git help glossary`
explains many terms that might be puzzling to new users.

git: Introduce yourself

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

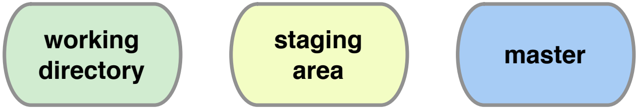
```
git config --global user.email "nchiapol@physik.uzh.ch"
```

Outline

single+local: Init

```
git init
```

- Creates an empty git repository with one branch
  - a branch stores a line of development (see next section)
  - default branch is called `master`
- Creates the git directory: `.git/`
- Your prompt may change.
  (If you added `$(__git_ps1)`)

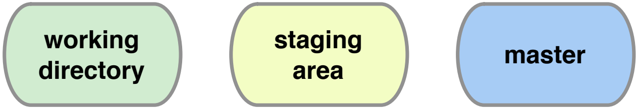


- does not change your files

single+local: Init

```
git init
```

- Creates an empty git repository with one branch
  - a branch stores a line of development (see next section)
  - default branch is called `master`
- Creates the git directory: `.git/`
- Your prompt may change.
  (If you added `$(__git_ps1)`)

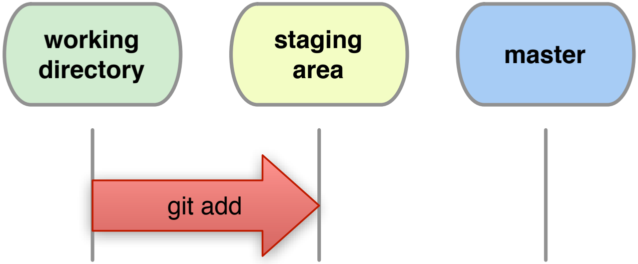


- does not change your files

single+local: Add

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

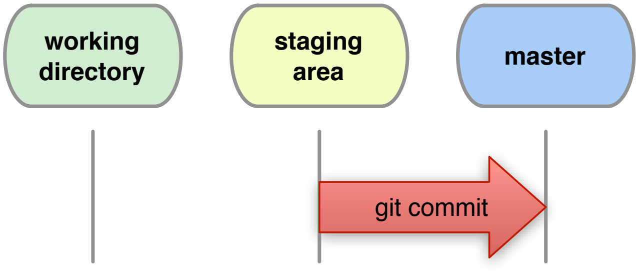
- Adds new files to be tracked by git
- Adds changes from working dir for next commit (Confusion!)
- DOES NOT add info on file permissions other than *exec/noexec*
- DOES NOT add directories *per se*.



single+local: Commit

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

Records changes from the staging area to master.

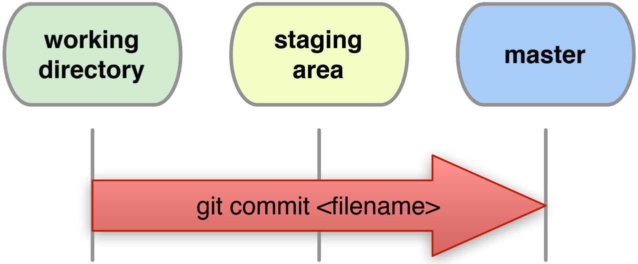


Config Tip: `git config [-global] core.editor "/usr/bin/kate"`

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.



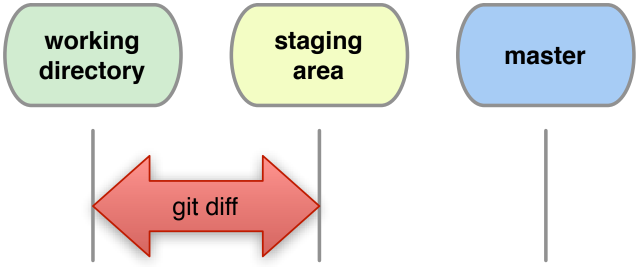
```
git commit -a[m "Commit message."]
```

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

single+local: Diff

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

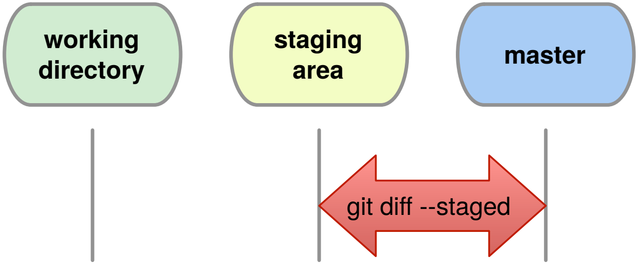
Shows changes between *working directory* and *staging area*



single+local: Diff Staged

How do I see what is staged?

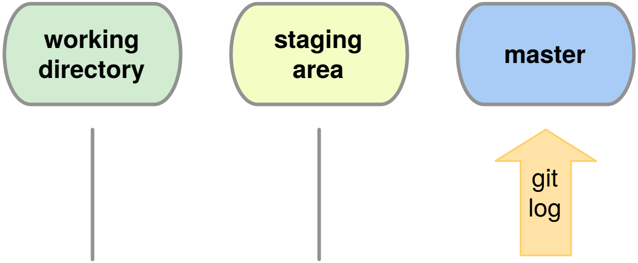
`git diff --staged` shows differences between staging area and last commit.



single+local: Commit History

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

Shows the history of a file or branch.

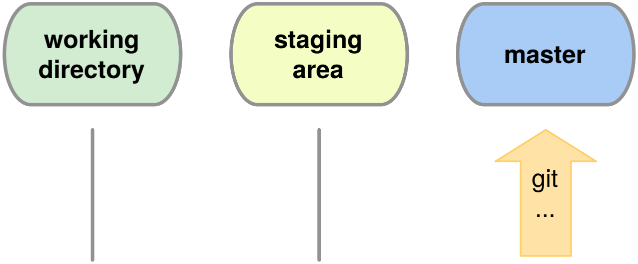


Config Tip: `git config [-global] log.date "iso"`

single+local: Old Changes

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

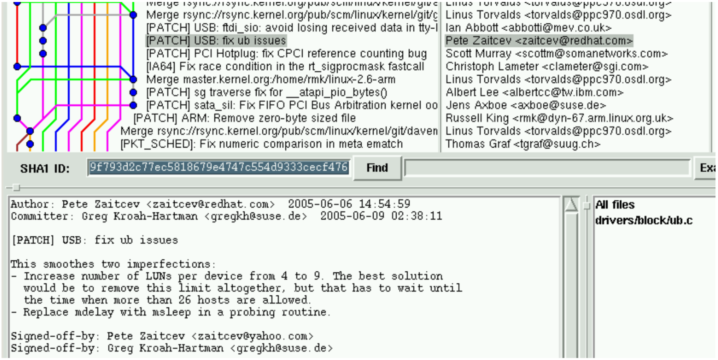
Shows the changes stored in commits.



single+local: Graphic Logs

`qgit` (or `gitg` or . . .)

GUI to browse the git repository.



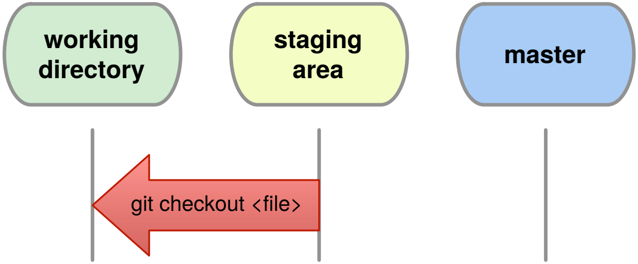
```
Merge rsync://rsync.kernel.org/pub/scm/linux/kernel/git/c   Linus Torvalds <torvalds@ppc970.osdl.org>
[PATCH] USB: ftdi_sio: avoid losing received data in tty-l  Ian Abbott <abbotti@mev.co.uk>
[PATCH] USB: fix ub issues                                  Pete Zaitcev <zaitcev@redhat.com>
[PATCH] PCI Hotplug: fix CPCI reference counting bug        Scott Murray <scottm@somanetworks.com>
[IA64] Fix race condition in the rt_sigprocmask fastcall    Christoph Lameter <clameter@sgi.com>
Merge master.kernel.org:/home/rmk/linux-2.6-arm             Linus Torvalds <torvalds@ppc970.osdl.org>
[PATCH] sg traverse fix for __atapi_pio_bytes()             Albert Lee <albertcc@tw.ibm.com>
[PATCH] sata_sil: Fix FIFO PCI Bus Arbitration kernel oo    Jens Axboe <axboe@suse.de>
[PATCH] ARM: Remove zero-byte sized file                    Russell King <rmk@dyn-67.arm.linux.org.uk>
Merge rsync://rsync.kernel.org/pub/scm/linux/kernel/git/daven Linus Torvalds <torvalds@ppc970.osdl.org>
[PKT_SCHED]: Fix numeric comparison in meta ematch          Thomas Graf <tgraf@suug.ch>
```

SHA1 ID: 9f793d2c77ec5818679e4747c554d9333cecf476

```
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>
```

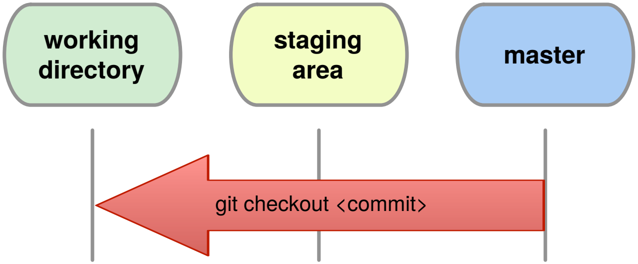
single+local: Changing Version

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



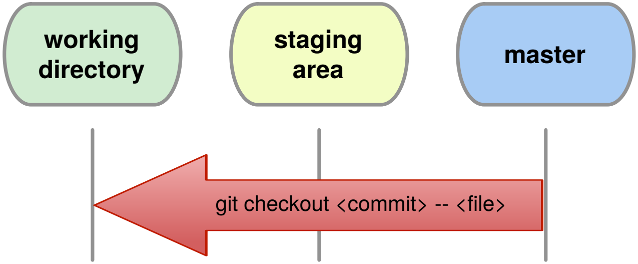
single+local: Changing Version

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



single+local: Changing Version

```
git checkout <commit> -- <file>
```



Warning: The old file is immediately staged for the next 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

Outline

Branches: Active Lines of Development

- So far: linear history stored in `master` branch
- could work on several branches in parallel
- seperate version of each file in each branch

```
* 9bce (HEAD -> master) bugfix
| * af63 (binary) optimize
| * b1f4 add documentation
* | 4c48 increase search space
| * d458 use binary search
|/
* 1209 brute force search
* f96f initial commit
```

Why

- Develop new features without breaking the running version
- Test different ideas starting from the same base
- Synchronise with a remote server (see next section)

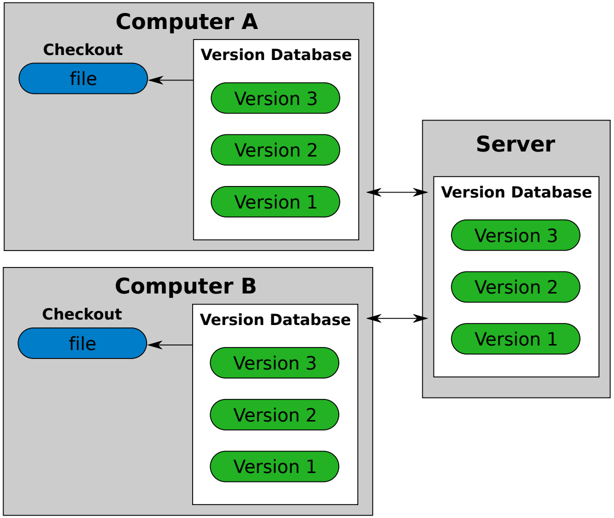
Branches: Common Commands

Create a new branch	`git branch <branch-name>`
Switch to a different branch	`git checkout <branch-name>`
Create + switch in one go	`git checkout -b <branch-name>`
List branches	`git branch [--list] [-a]`
Integrate changes	`git merge <branch-name>` includes all changes from `branch-name` into the currently checked-out branch
Delete a branch	`git branch -d <branch-name>`

Note: Normal git commands only affect the branch currently checked out.

Outline

multi+remote/central: Setup

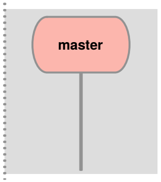


multi+remote/central: Clone

```
git clone <URL>
```

Creates two local copies of the whole remote branch.

Remote (Server)



Version Database

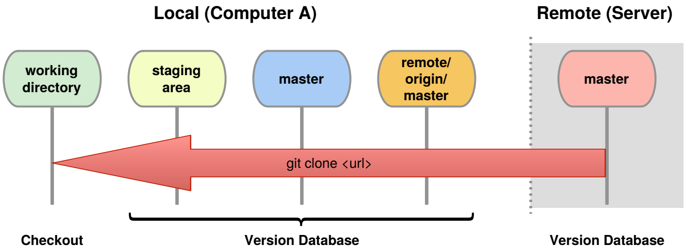
Hint

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

multi+remote/central: Clone

```
git clone <URL>
```

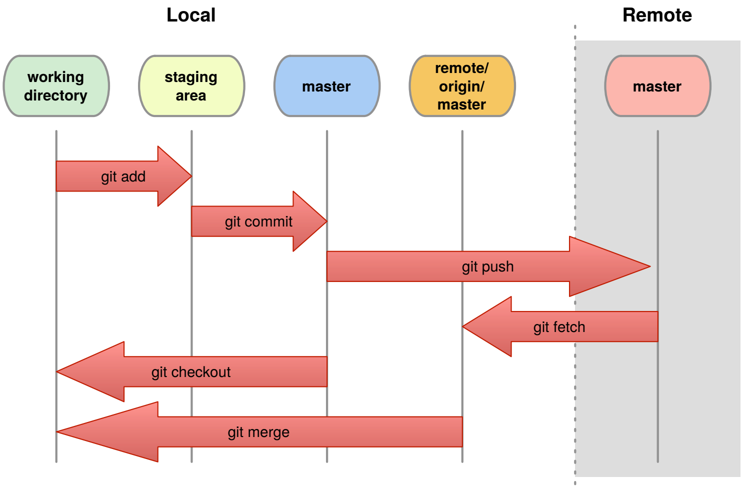
Creates two local copies of the whole remote branch.



Hint

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

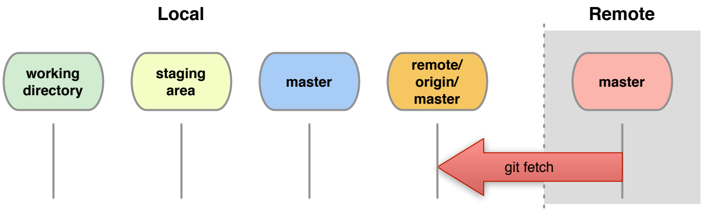
multi+remote/central: Commands



multi+remote/central: Fetch

```
git fetch
```

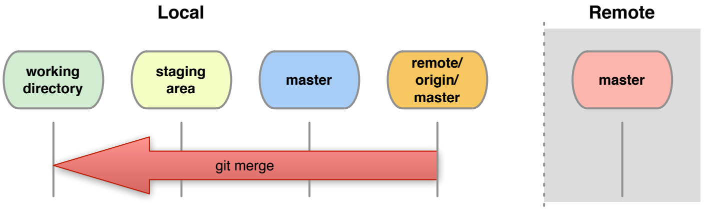
- Updates origin master from remote master
- local master, staging area and working dir not changed



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
...
```

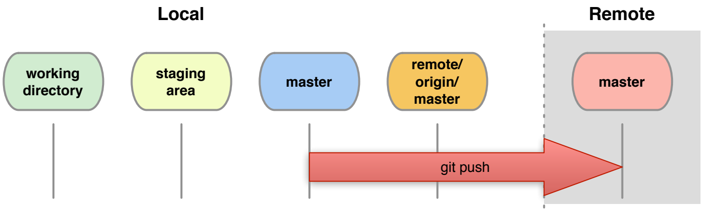
multi+remote/central: Resolving Conflicts

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

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

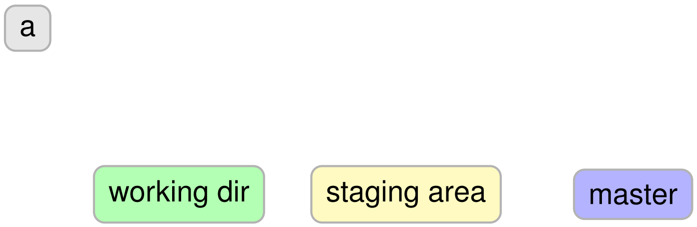
Everybody clones:

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

Outline

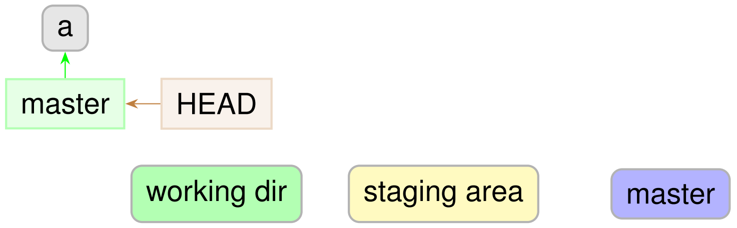
Behind the Scenes: Setup

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



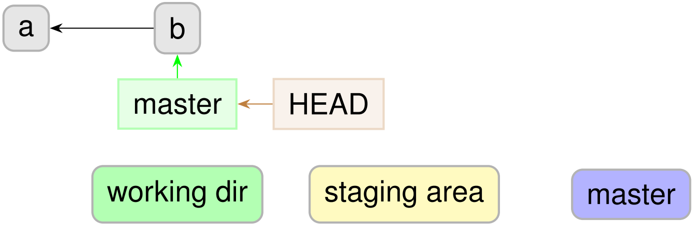
Behind the Scenes: Setup

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



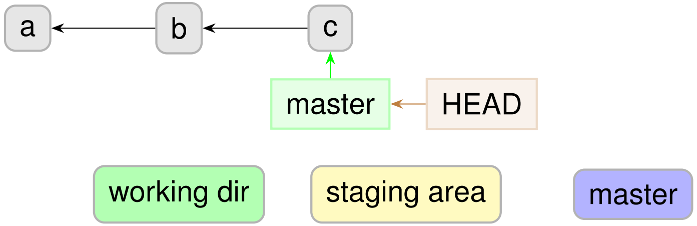
Behind the Scenes: Setup

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



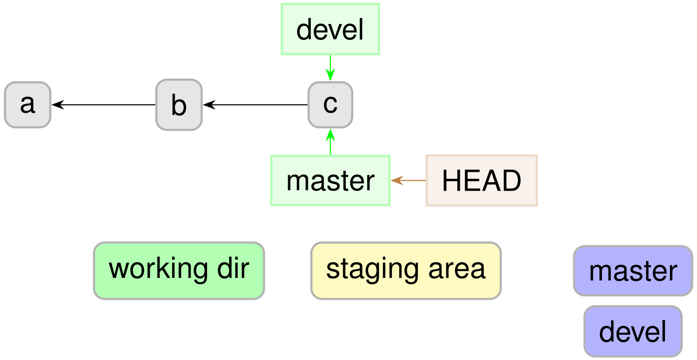
Behind the Scenes: Setup

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



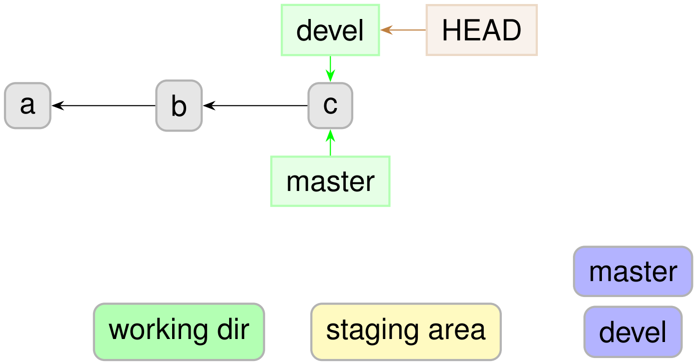
Behind the Scenes: Branches

```
git branch devel
```



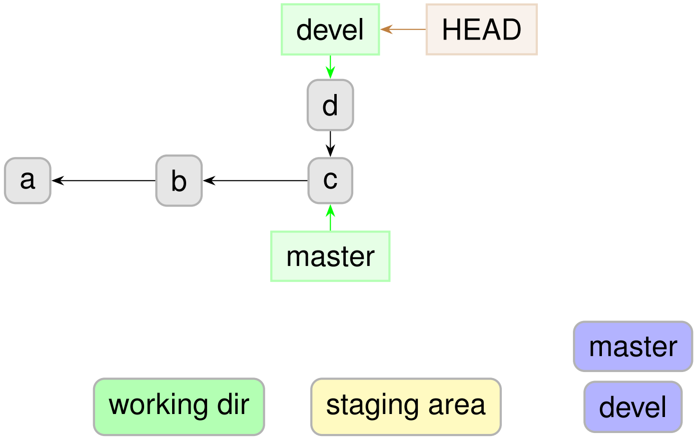
Behind the Scenes: Branches

```
git checkout devel
```



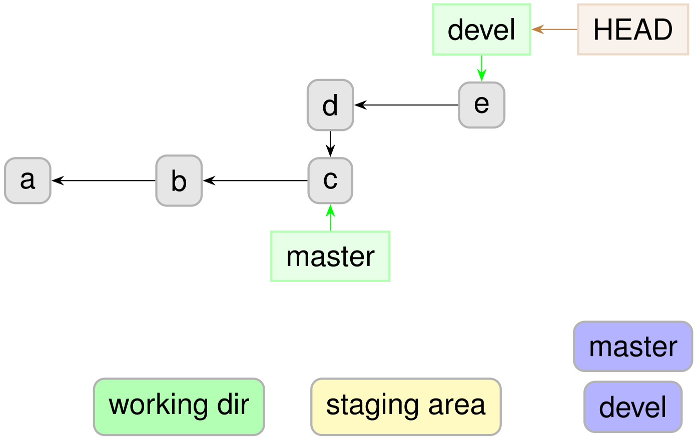
Behind the Scenes: Branches

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



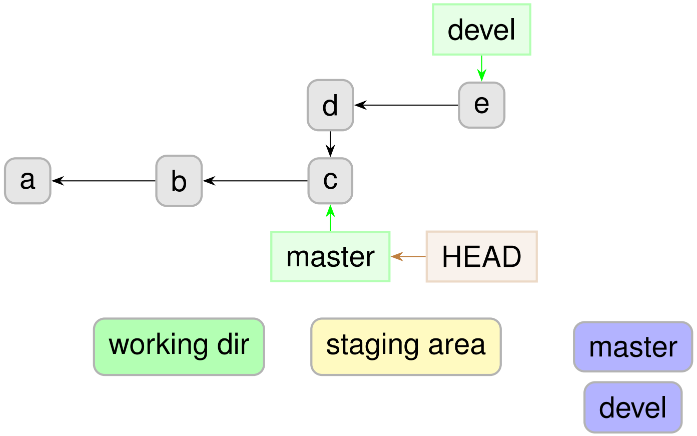
Behind the Scenes: Branches

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



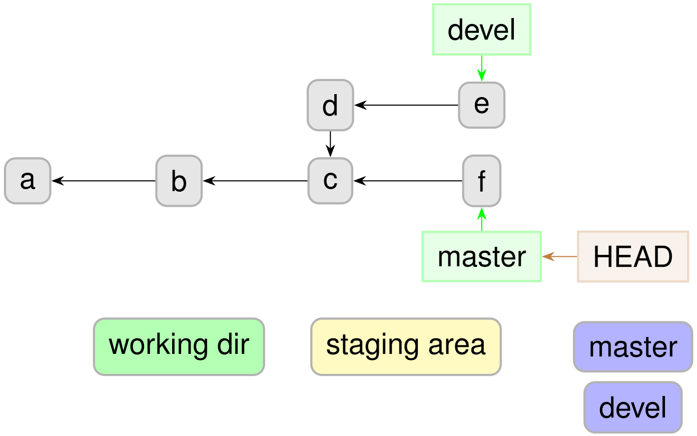
Behind the Scenes: Branches

```
git checkout master
```



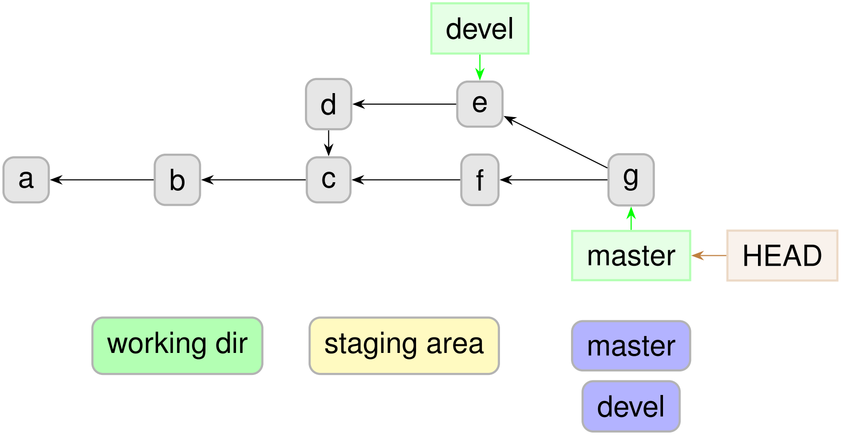
Behind the Scenes: Branches

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



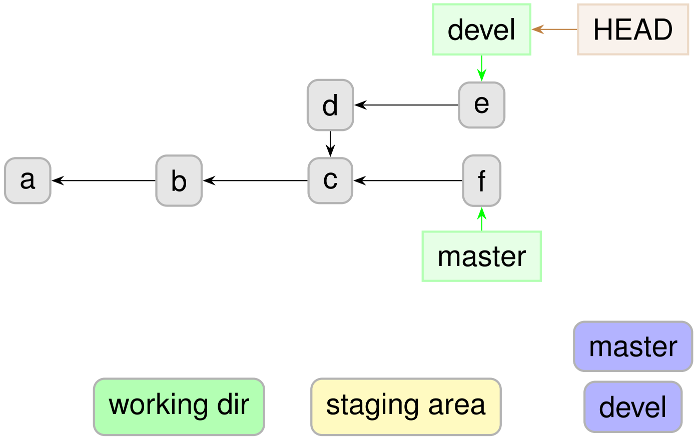
Behind the Scenes: Branches

```
git merge devel
```



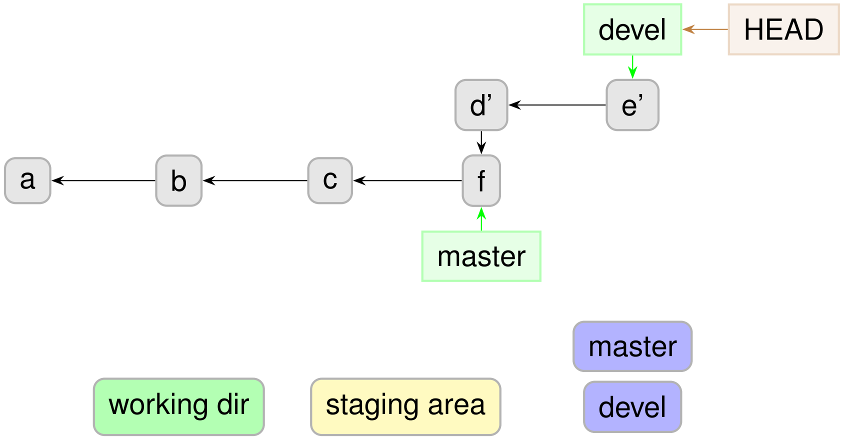
Behind the Scenes: Rebase

```
git checkout devel
```



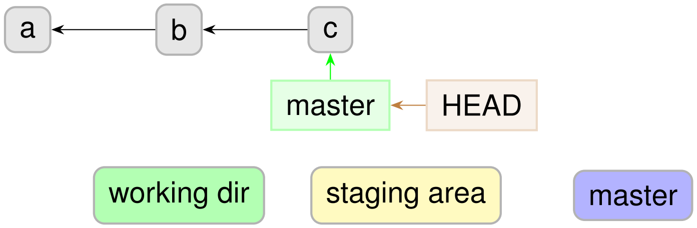
Behind the Scenes: Rebase

```
git rebase master
```



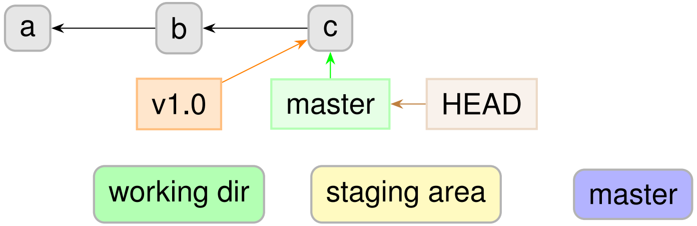
Behind the Scenes: Setup

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



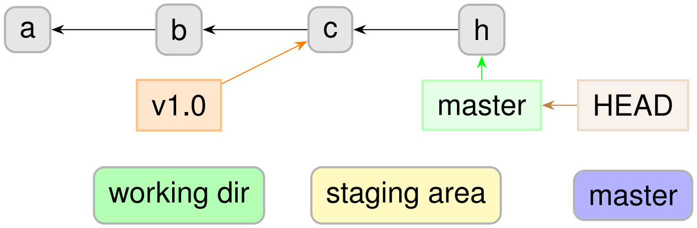
Behind the Scenes: Tags

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



Behind the Scenes: Tags

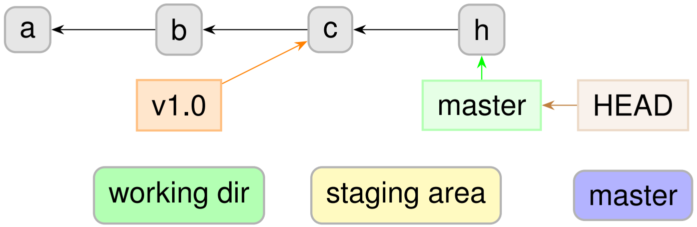
```
git commit -am "H"
```



Behind the Scenes: Tags

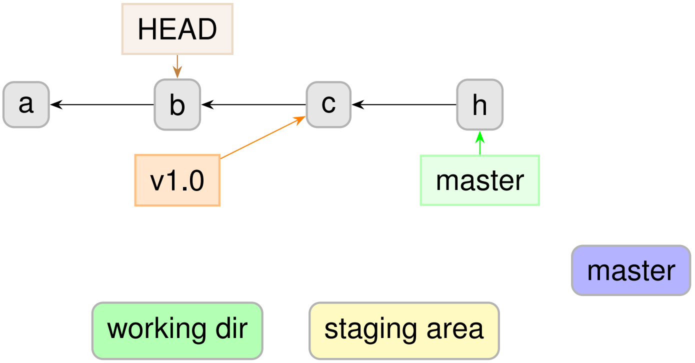
```
git commit -am "H"
```

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



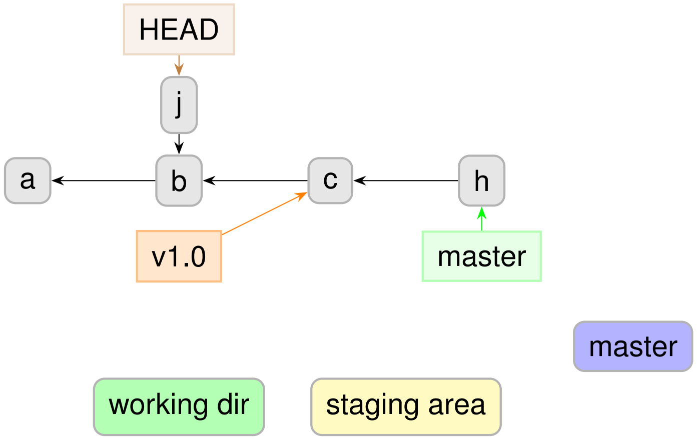
Behind the Scenes: Detached HEAD

```
git checkout b
```



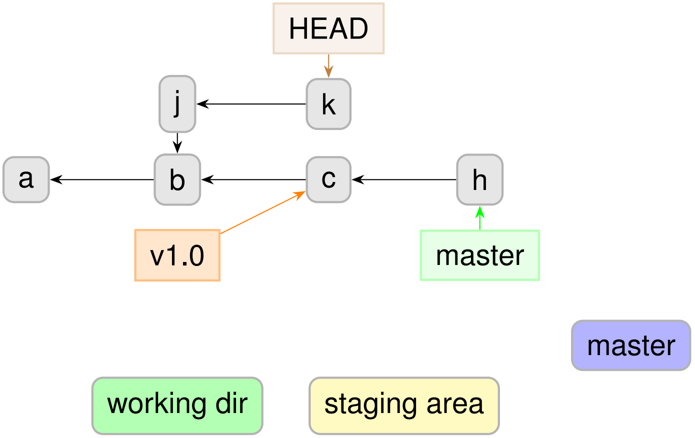
Behind the Scenes: Detached HEAD

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



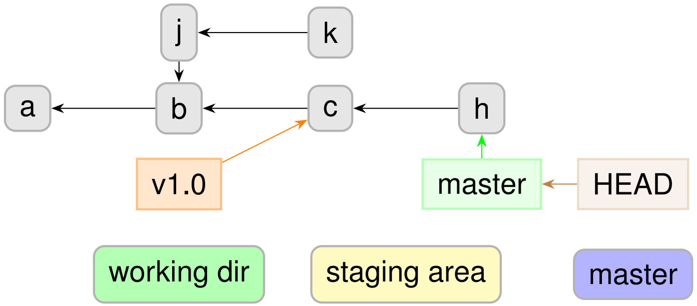
Behind the Scenes: Detached HEAD

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



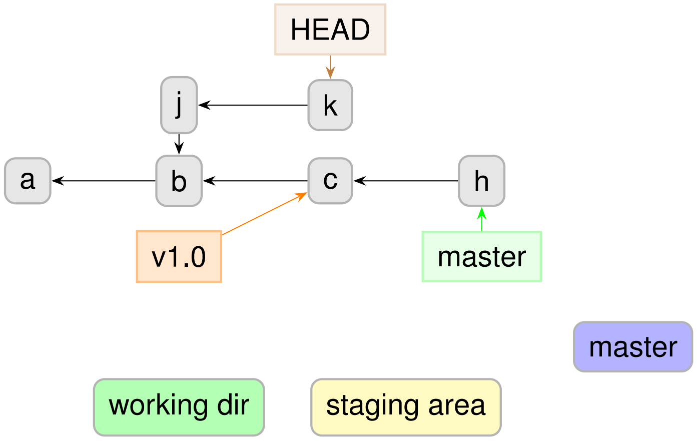
Behind the Scenes: Detached HEAD

```
git checkout master
```



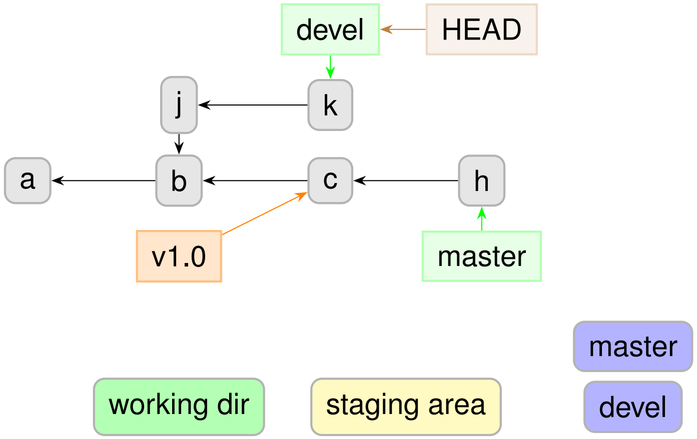
Behind the Scenes: Detached HEAD

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



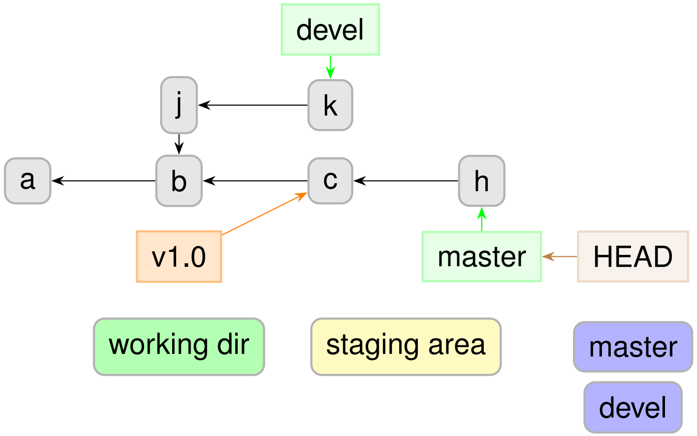
Behind the Scenes: Detached HEAD

```
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: https://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/