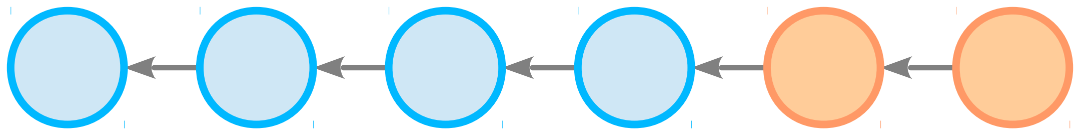
git
the incomplete introduction

René Schwietzke, Xceptance, 2013

v 0.96

Please don't...



Please don't ask for any mapping of SVN to Git commands. Git is different and you cannot simply use the same approach and commands.

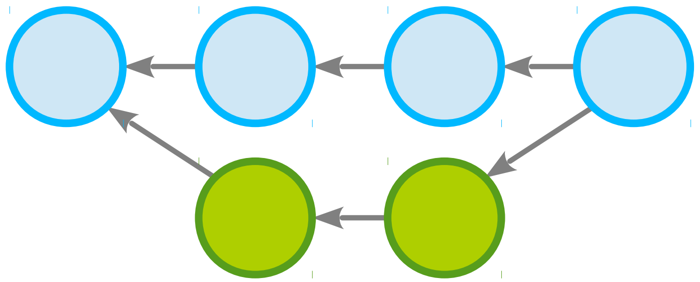
Of course you can, but people will hate you forever.

Some (de)motivation first

- Most of things are borrowed
- Command line only first
- Yes, you can mess things up
- Don't be scared!
- Be scared and respect the tool!
- Not every term will be introduced
- Google is your friend
- Go http://git-scm.com/ and use the book!
- Share this deck, it is CC-BY-SA licensed

Why are we so... excited?

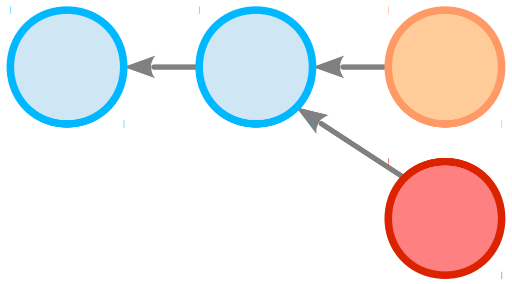
- Subversion is ok – Git is better
- Commit often, very often...
- Everything is **local**
- Everything is **distributed**
- Cheap branching
- Data assurance
- Fast
- Everything has a history... P.S That is going to be important later on



The swiss army knife – you just have to know how to use it.

What is that GIT/git/Git thing?

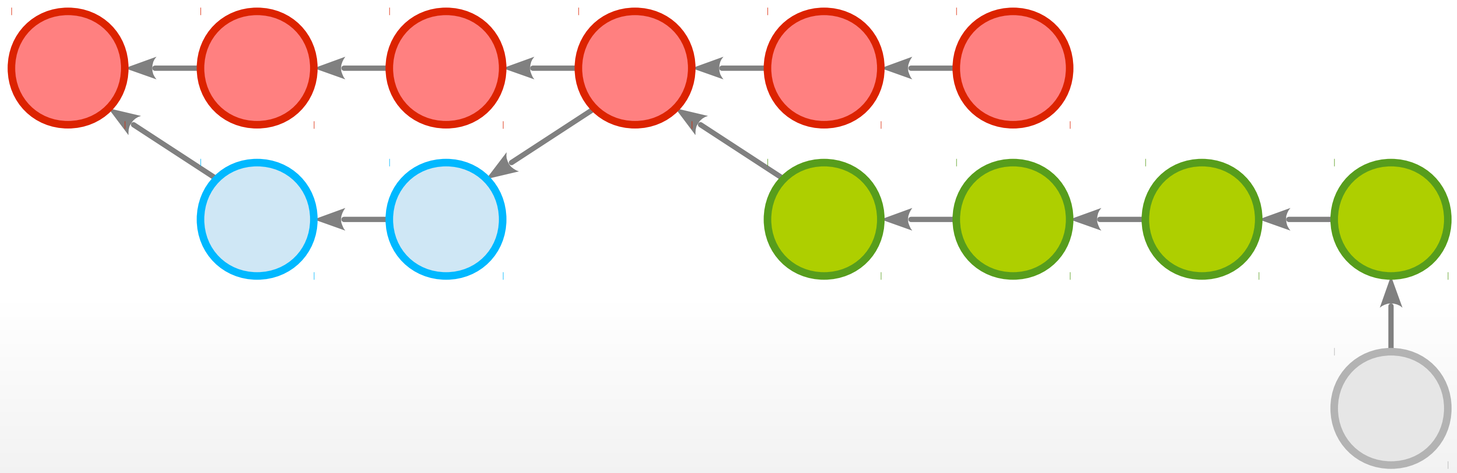
- Distributed Version Control System (DVCS)
- Open Source
- Started by Linus T. to aid kernel development
- Used by a lot of projects
- Git is not GitHub!



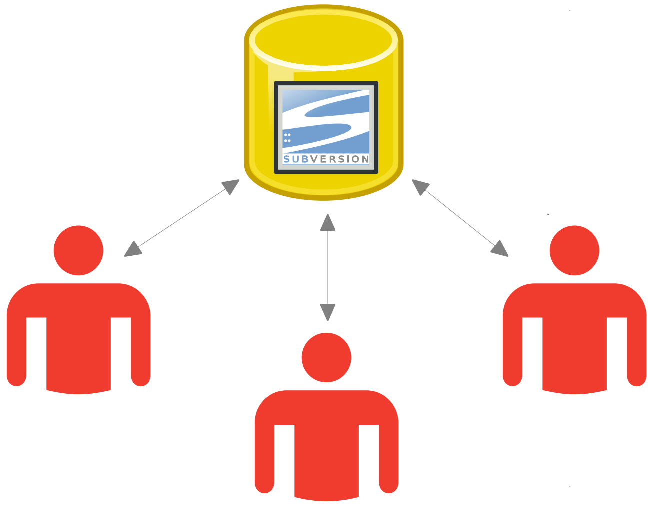
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git takes care of...

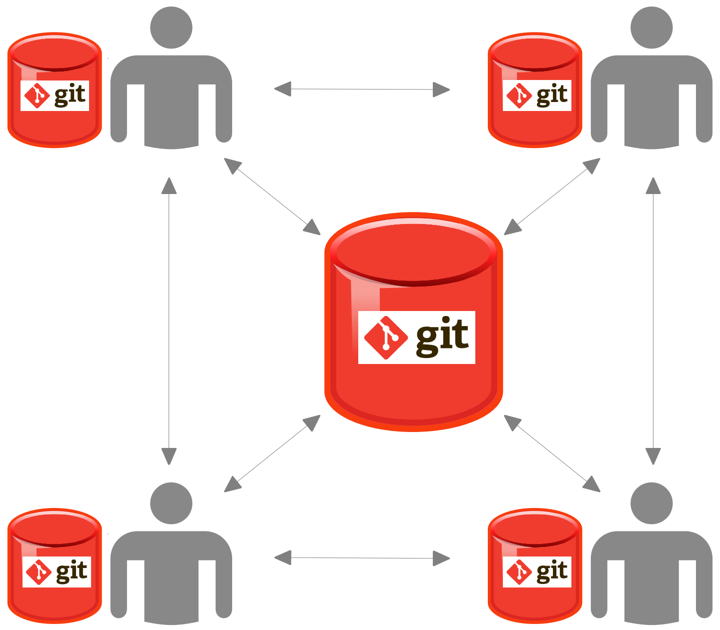
- Content: by protecting each commit with SHA-1
- History: by keeping relations
- Merges: by using the history
- Speed: by using snapshots and not diffs



Distributed vs. Centralized



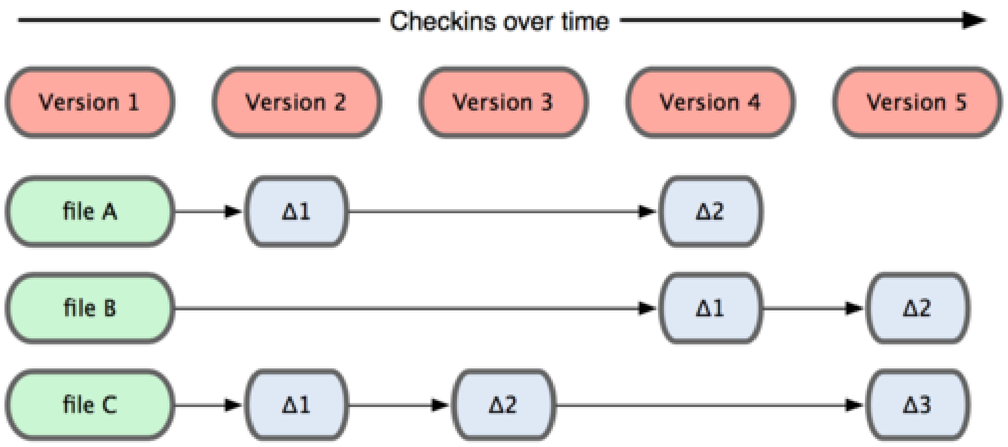
Centralized

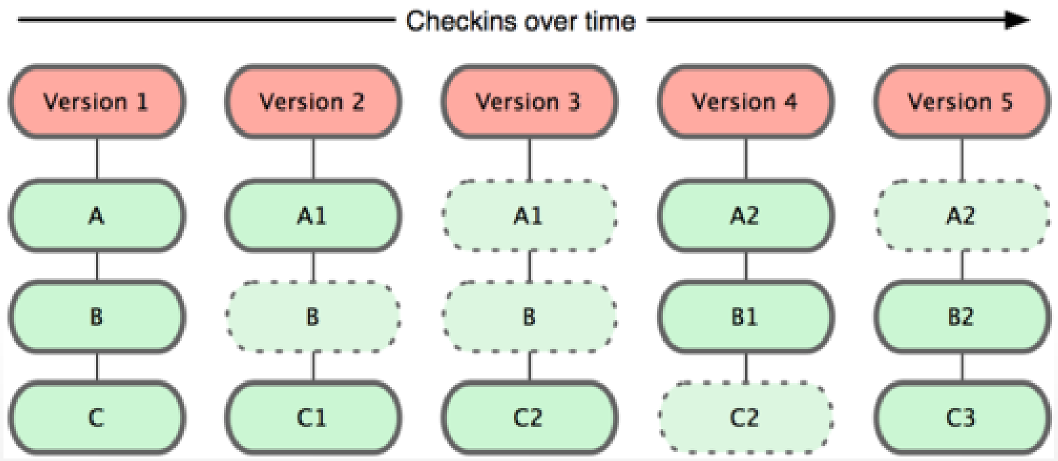


Distributed

How things are stored

- Snapshots and not file differences





Lifetime phases of data

- **Unmodified** – you did nothing
- **Modified** – you changed something
- **Staged** – you prepared for a commit
- **Committed** – you made changes permanent
- **Pushed** – you shared your changes

You have to tell Git that you want a file to be included in the next commit. Git is opt-in!

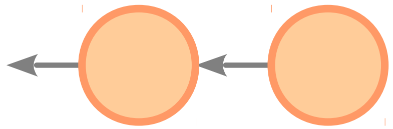
The thingy were Git keeps your soon to be committed data is called the staging area.

What is a commit?

- A commit is a snapshot of files
- Keeps references to its parents
- Has a unique SHA-1 over the content

```
Local ~$ git show --pretty=raw 28d88a4
commit 28d88a44d6640a7ab89aa9b0db0597e36b50c831
tree 55335193a182bc9e2850d00700ff482896f95c0f
parent bcbb62be834ca0abd4b81952e612bf64f57bedb2
author Kati Smith <kati@foobar.net> 1379449534 +0200
committer Kati Smith <kati@foobar.net> 1379449534 +0200

    Fallback code added

Local ~$ git show --pretty=fuller 0da605d
commit 0da605df6974eaeef472355936b9fc15912a6211
Merge: a6c4011 28d88a4
...
```

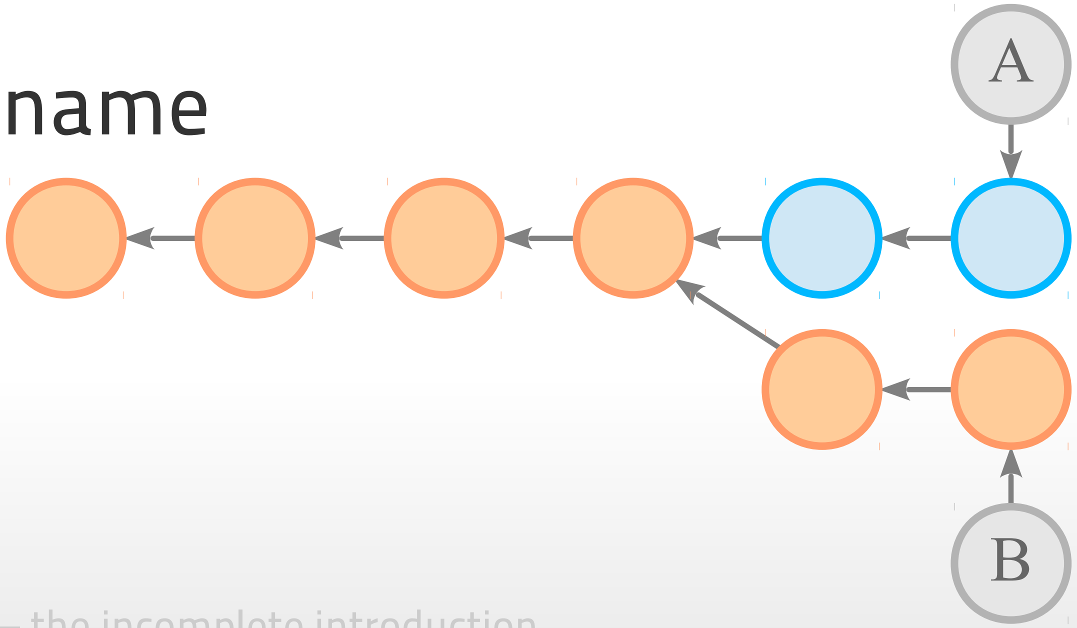
Data assurance and commits



- Git does not have a counter
- Git has data assurance
  - Changes in the past will change anything from that point in time
- A commit knows its predecessors
- A commit has a unique SHA-1
- Any change to a commit changes the SHA-1
- Any new SHA-1 is something new despite the same content (see rebase)

What is a branch?

- A branch is just a pointer to a commit that is the branch head
- A branch contains everything that can be reached by following the parent relations
- A branch is like a parallel universe
- Each branch has a name



Get your hands gitty

We play with git

Get your hands gitty

- Local repo first
- Learn the basics
- Clone a remote repo
- Some magic around shared code

```
Local ~$ git --version
git version 1.8.3.4
```

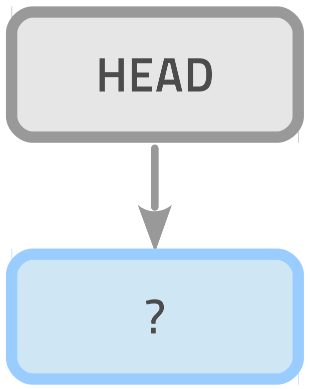
git init

- Creates a new repository right here
- Never ever play with **.git** subdirectory

```
Local ~$ mkdir mytest
Local ~$ cd mytest
Local ~mytest$ git init
Initialized empty Git repository in ~mytest/.git/
Local ~mytest$ ls -la
total 0
drwxrwxr-x  3 tfontane tfontane   60 Jul 27 20:56 .
drwxrwxr-x  7 tfontane tfontane  200 Jul 27 20:56 .git
```

git init

git



File Sys

git config

- Tell something about you
- Git identifies you but does not authenticate!
- Stash/GitLab/GitHub do the authentication

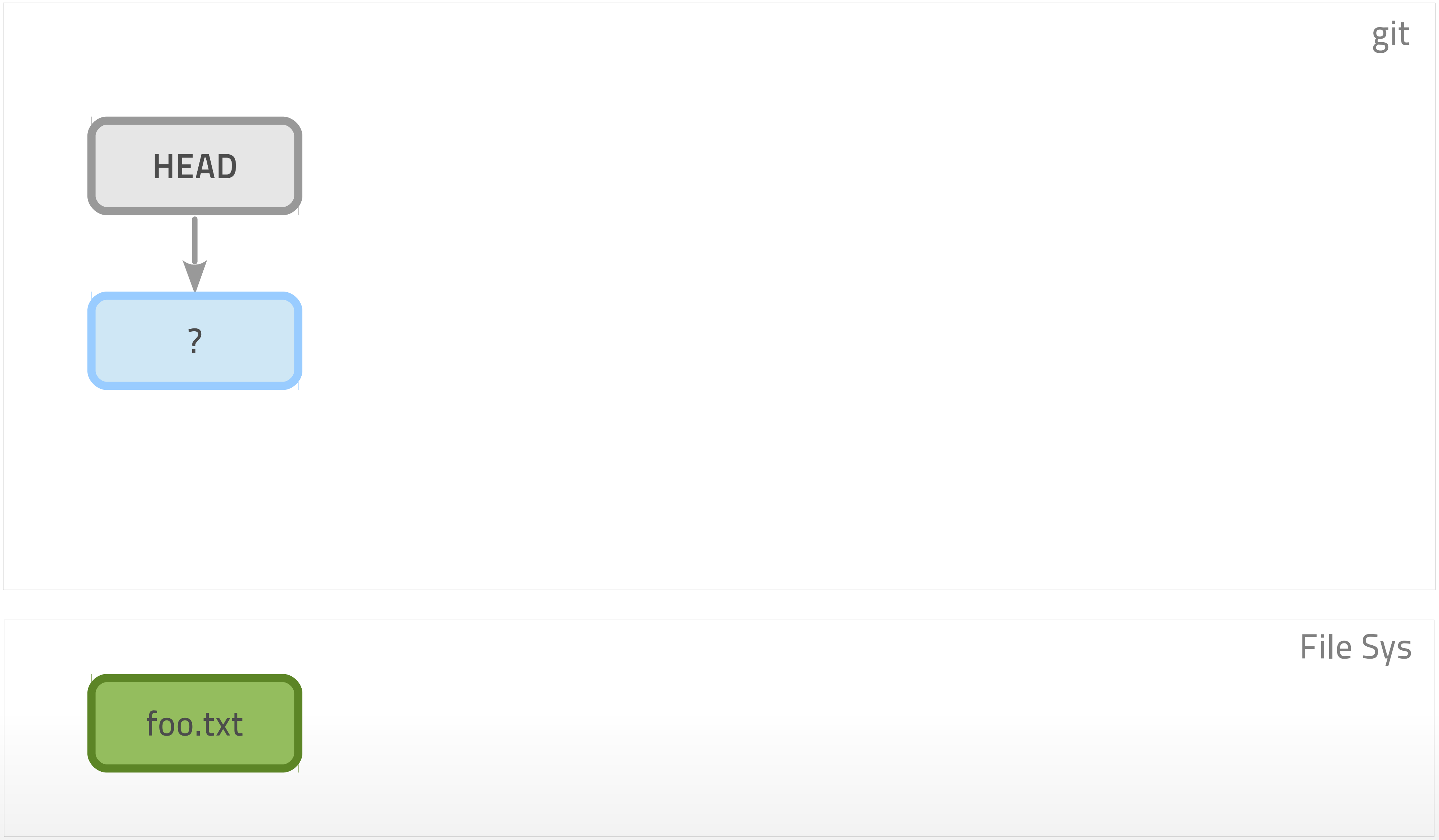
```
Local ~$ git config user.name "Rene Schwietzke"
Local ~$ git config user.email "emailme@foobar.com"

Local ~$ git config --global user.name "Rene Schwietzke"
Local ~$ git config --global user.email "emailme@foobar.com"
```

git status - add a little

- Create a file and fill it
- Check the status
- git status is your friend!

```
Local ~mytest$ vi foo.txt
Local ~mytest$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   foo.txt
nothing added to commit but untracked files present (use "git add" to
track)
```

Modified

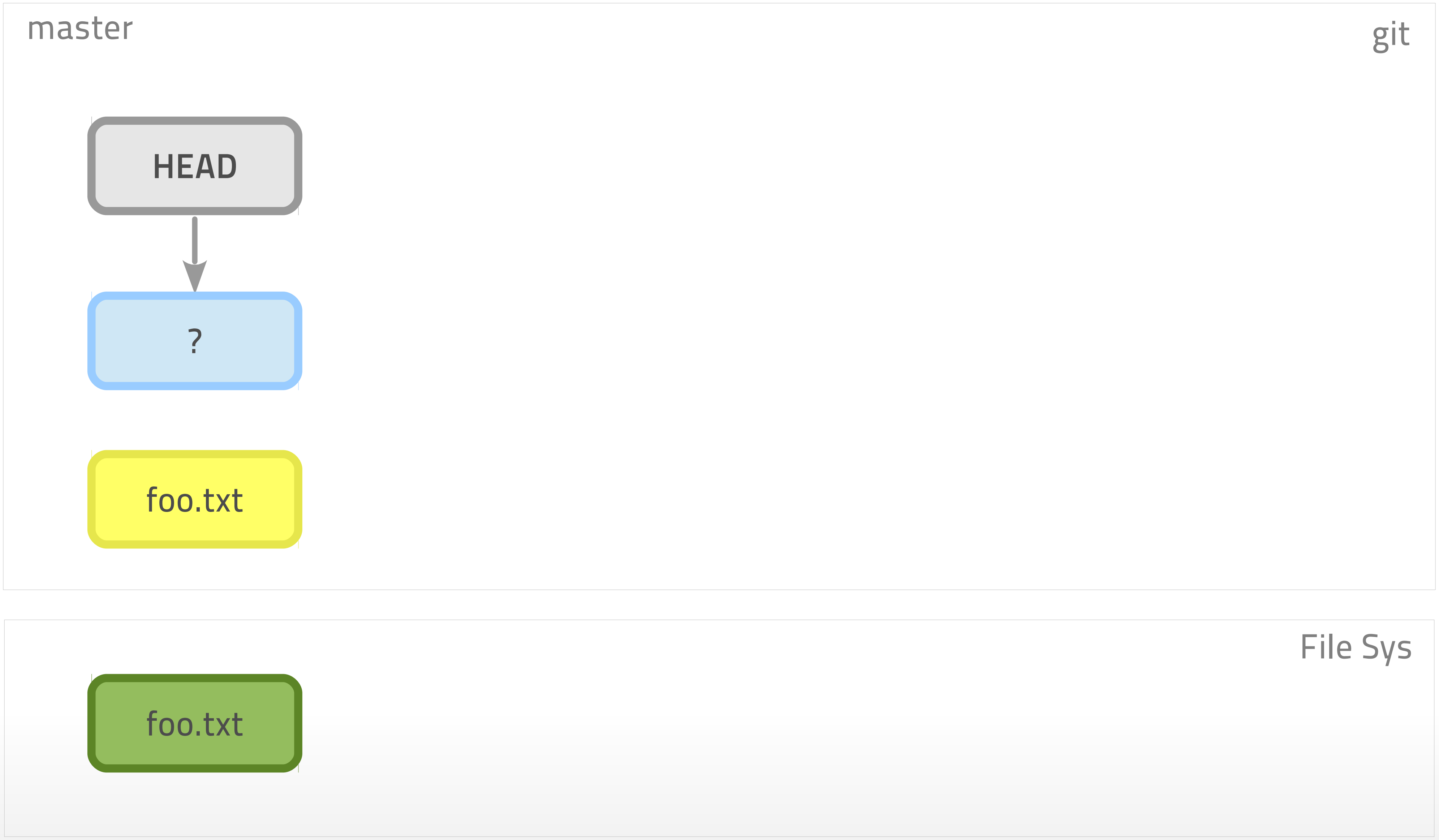


git add / git status

- Stage the file to make it known to git
- Git will snapshot the file
- Check the state

```
Local ~mytest$ git add foo.txt
Local ~mytest$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   foo.txt
#
Local ~mytest$ git add . # all recursive
Local ~mytest$ git add * # all in current dir
```

Staged

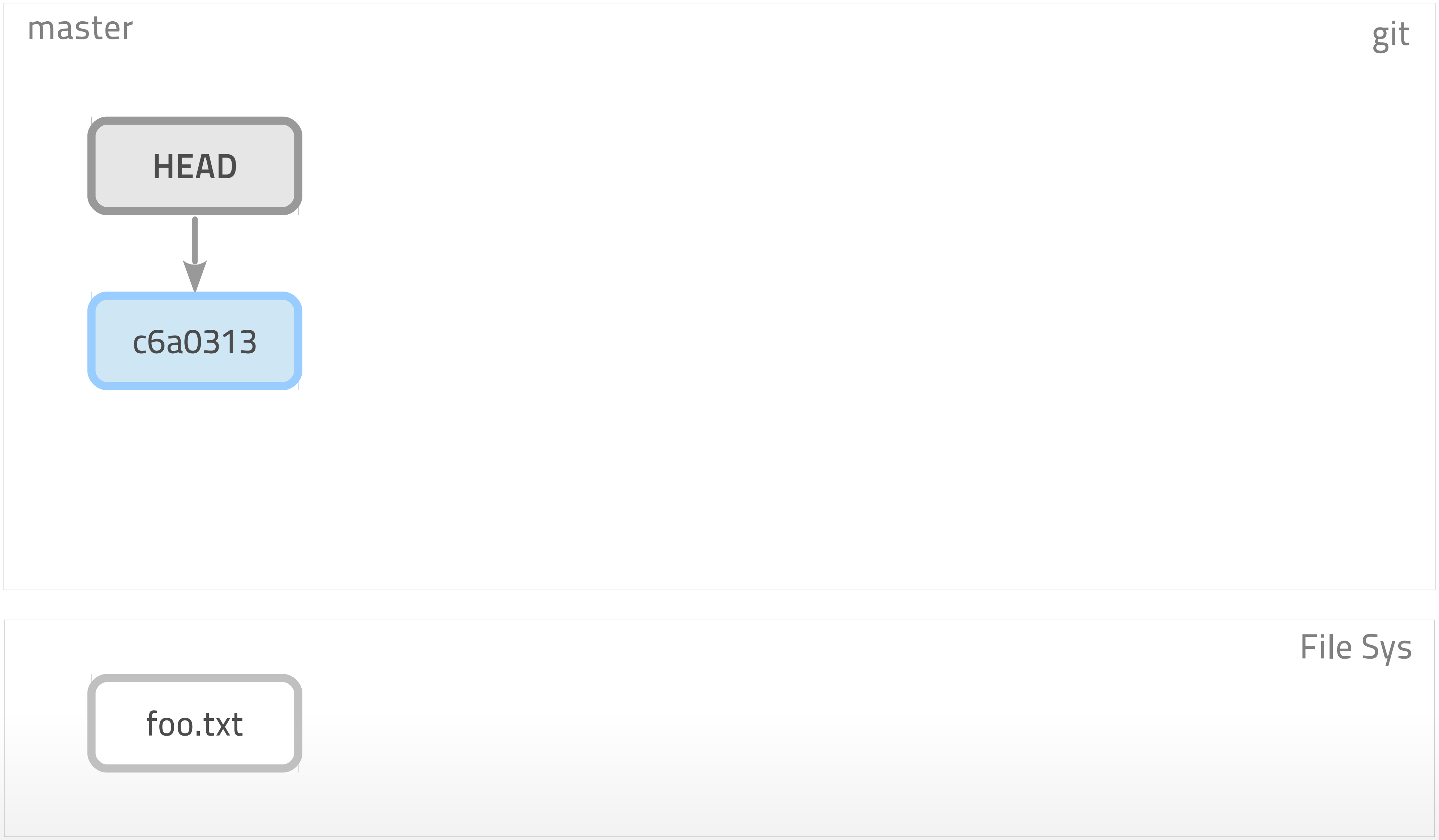


git commit – engrave it in stone

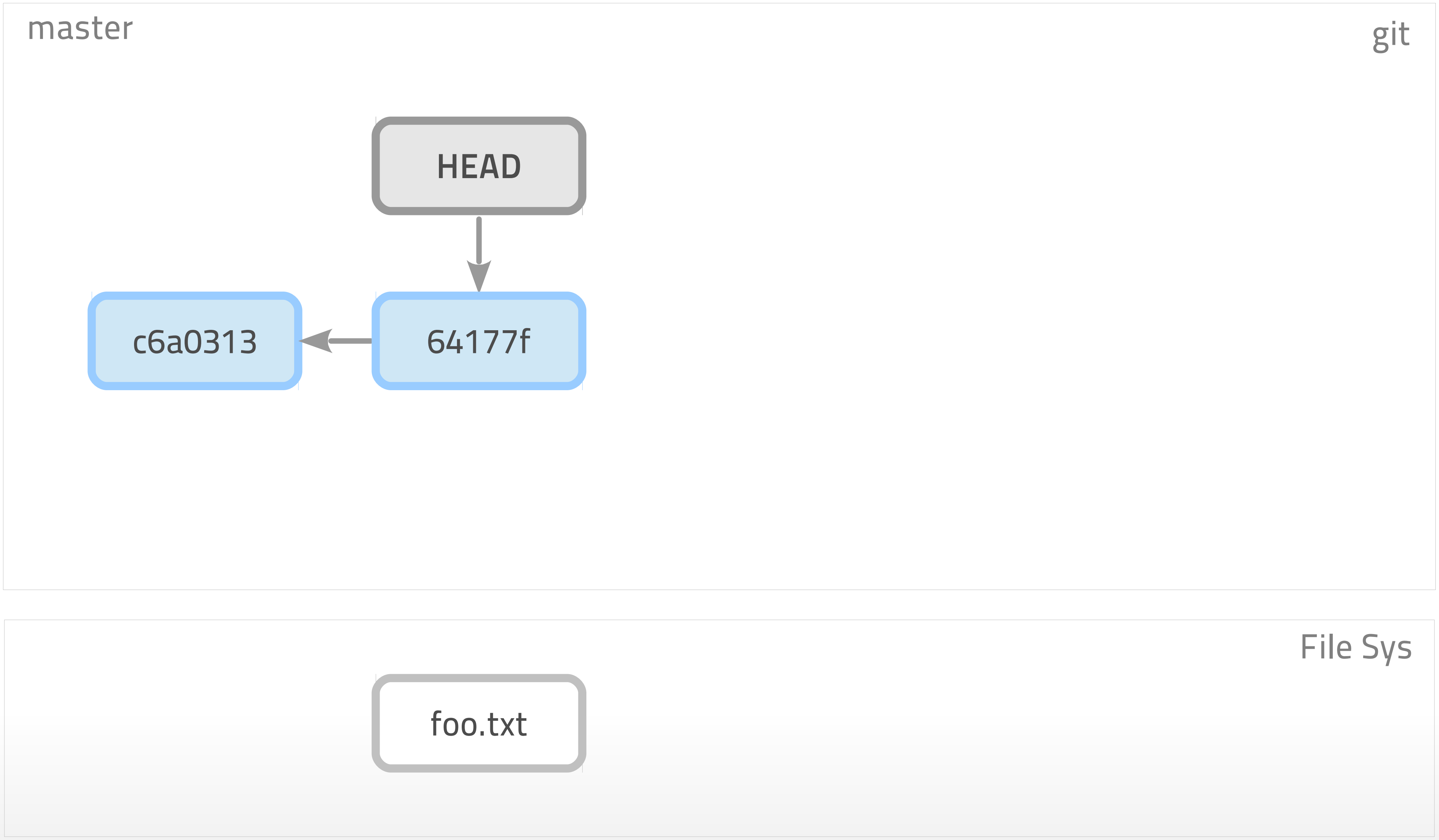
- Make your changes permanent
- Commits only what you staged
- No commit comment – no fun!

```
Local ~mytest$ git commit -m "Just started"
[master (root-commit) c6a0313] Just started
 1 file changed, 2 insertions(+)
 create mode 100644 foo.txt
Local ~mytest$ git status
# On branch master
nothing to commit, working directory clean
Local ~mytest$
```

Committed



Play it again Sam!

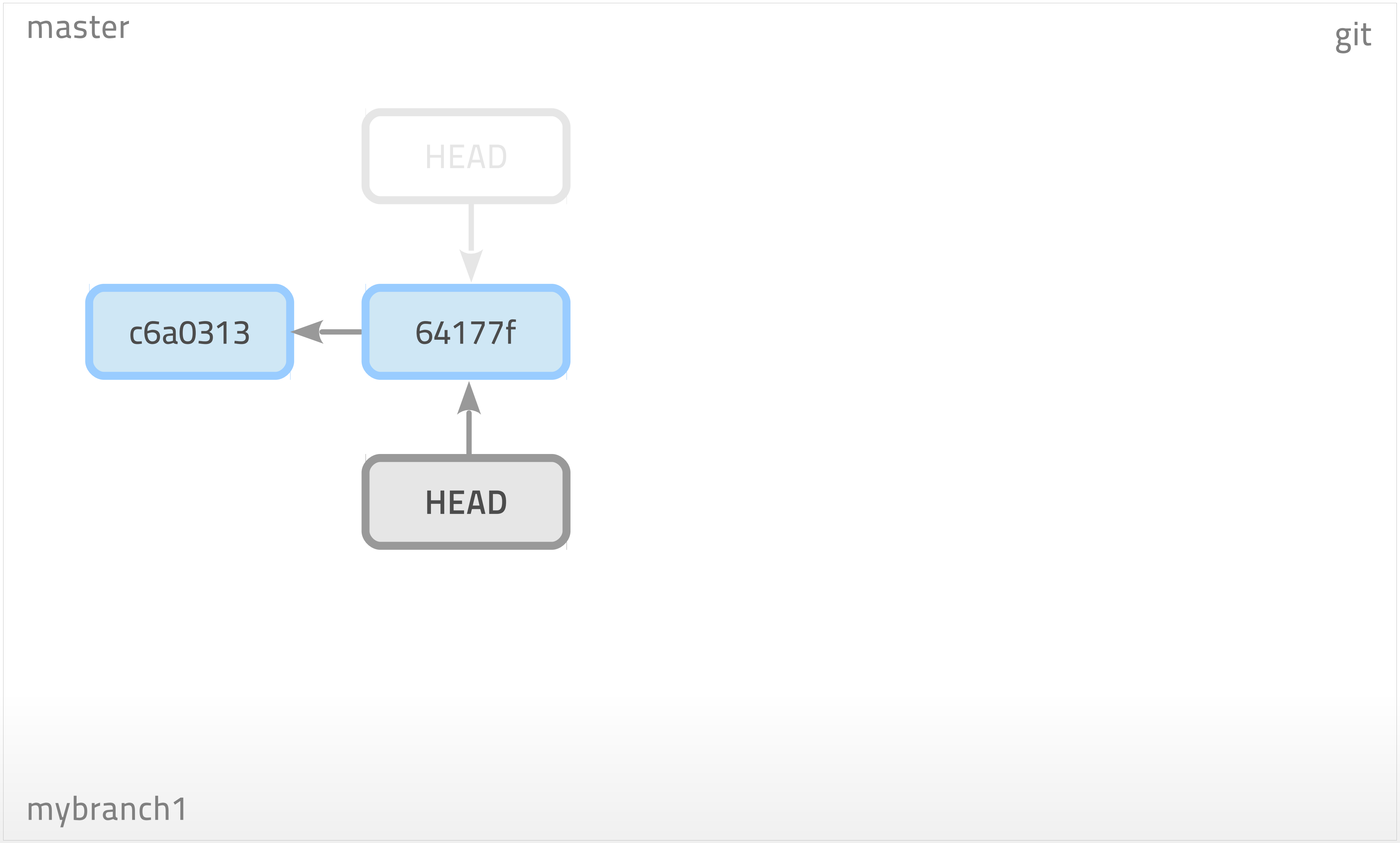


git branch / git checkout

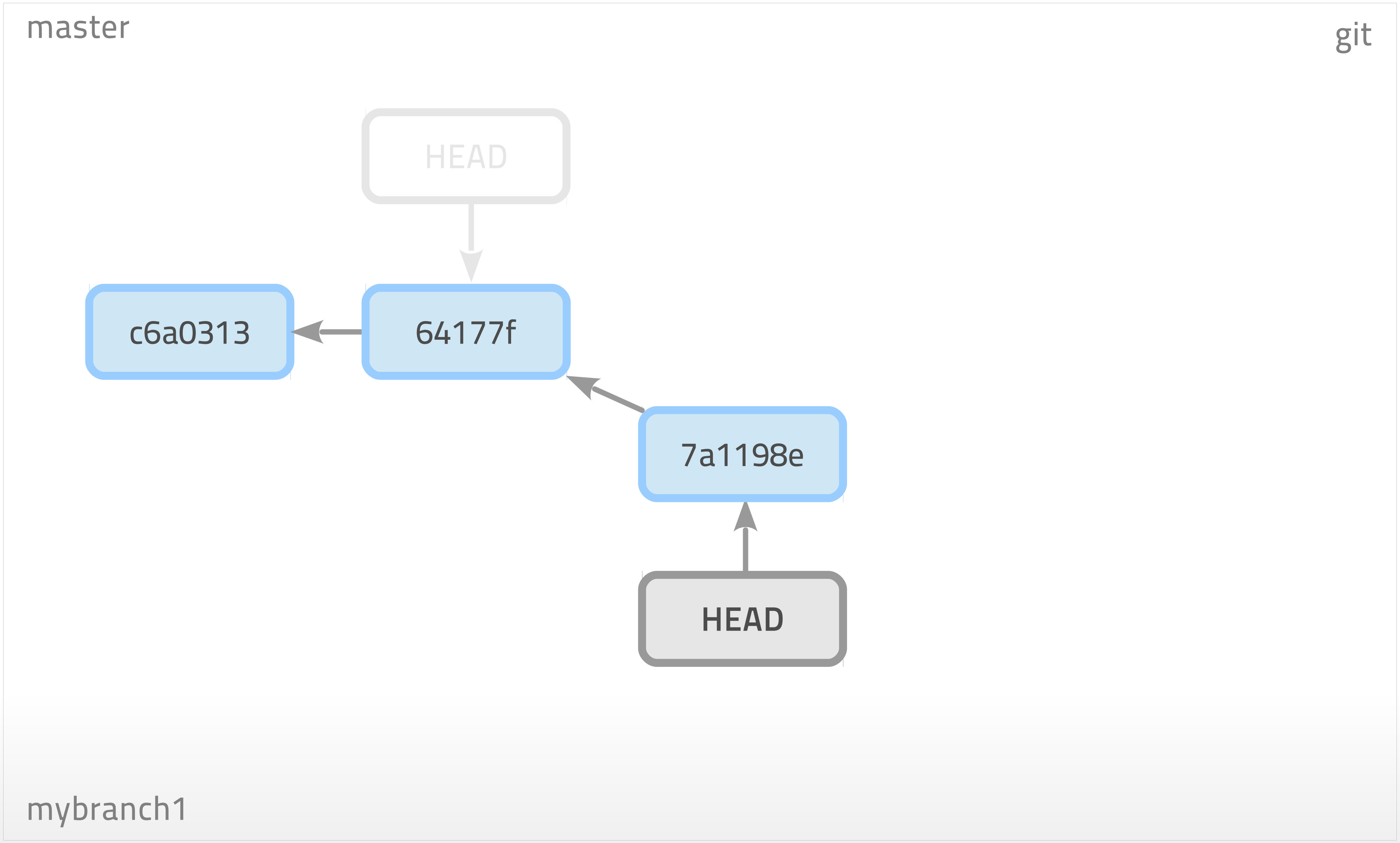
- Cheap and fast
- Your playground to try things, finish things, and share things, all that without disturbing others
- Use a branch, Luke!

```
Local ~mytest(master)$ git branch mybranch1; git checkout mybranch1
Local ~mytest(master)$ git checkout -b mybranch1
Switched to a new branch 'mybranch1'
Local ~mytest(mybranch1)$ git status
# On branch mybranch1
nothing to commit, working directory clean
Local ~mytest(mybranch1)$ git branch
  master
* mybranch1
Local ~mytest(mybranch1)$ git checkout master
Switched to branch 'master'
Local ~mytest(master)$ git checkout mybranch1
Switched to branch 'mybranch1'
```

Branch View



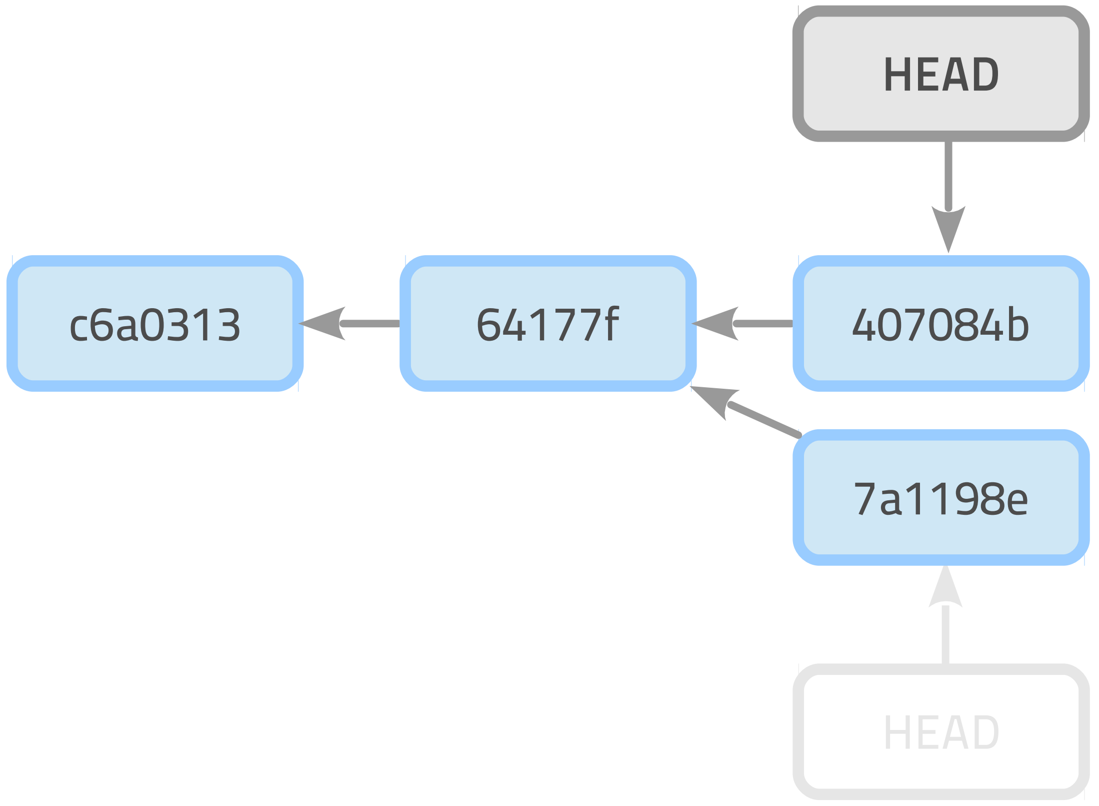
Modify / Add / Commit



Back to master and new file

master

git



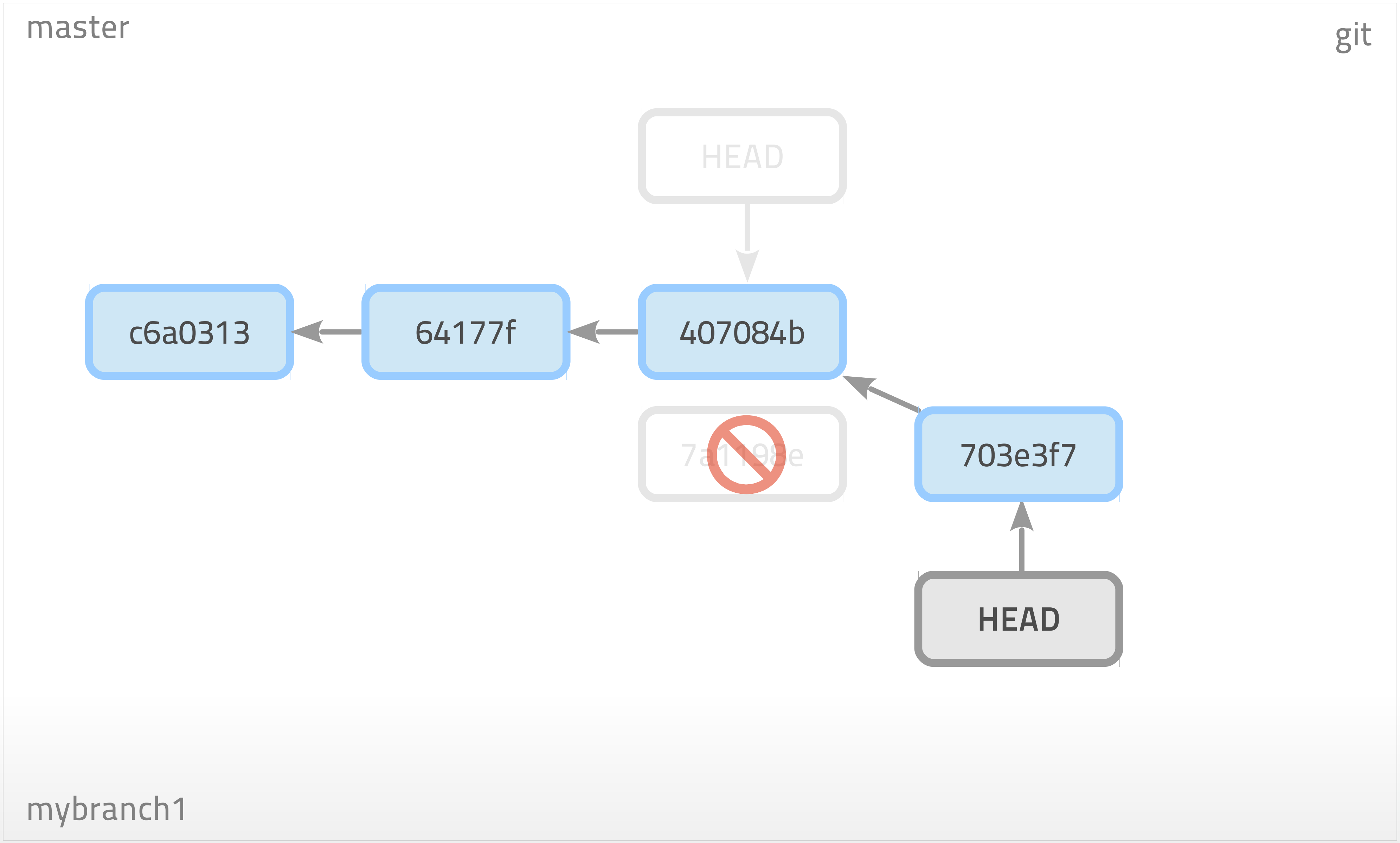
mybranch1

git rebase on mybranch1

- Prepare mybranch1 for merge into master
- Resolve conflicts in the branch
- Rebasing means rewriting of history

```
Local ~mytest$ git checkout mybranch1
Local ~mytest$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Longer sentence
Local ~mytest$ git log --oneline --decorate
703e3f7 (HEAD, mybranch1) Longer sentence
407084b (master) First readme version
641771f Again
c6a0313 Just started
```

We rebased

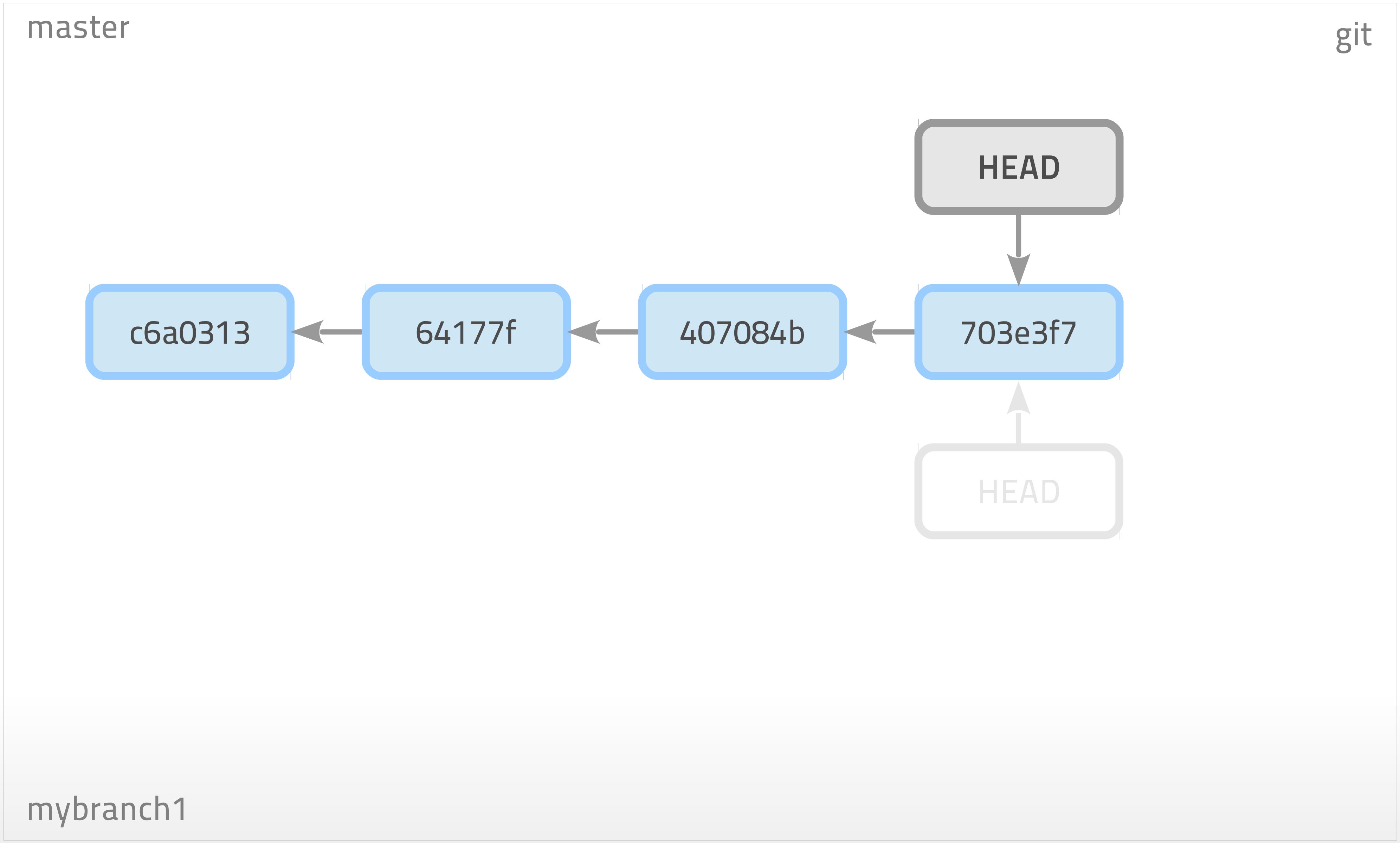

master
git
HEAD
c6a0313
64177f
407084b
7a1198e
703e3f7
HEAD
mybranch1

git merge into master

- Get your changes into the master
- Should run great because you rebased first
- Git will remember that the branch was merged
- Git will remember the content aka the SHA-1s

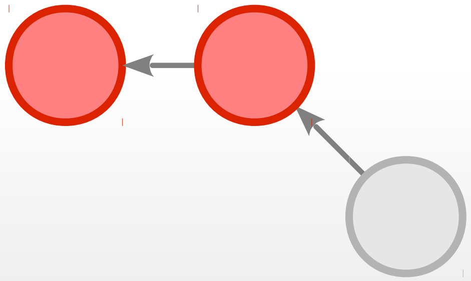
```
Local ~mytest$ git checkout master
Local ~mytest$ git merge mybranch1
Updating 407084b..703e3f7
Fast-forward
 foo.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
Local ~mytest$ git log --oneline --decorate
703e3f7 (HEAD, mybranch1, master) Longer sentence
407084b First readme version
641771f Again
c6a0313 Just started
```

We merged via fast-forward

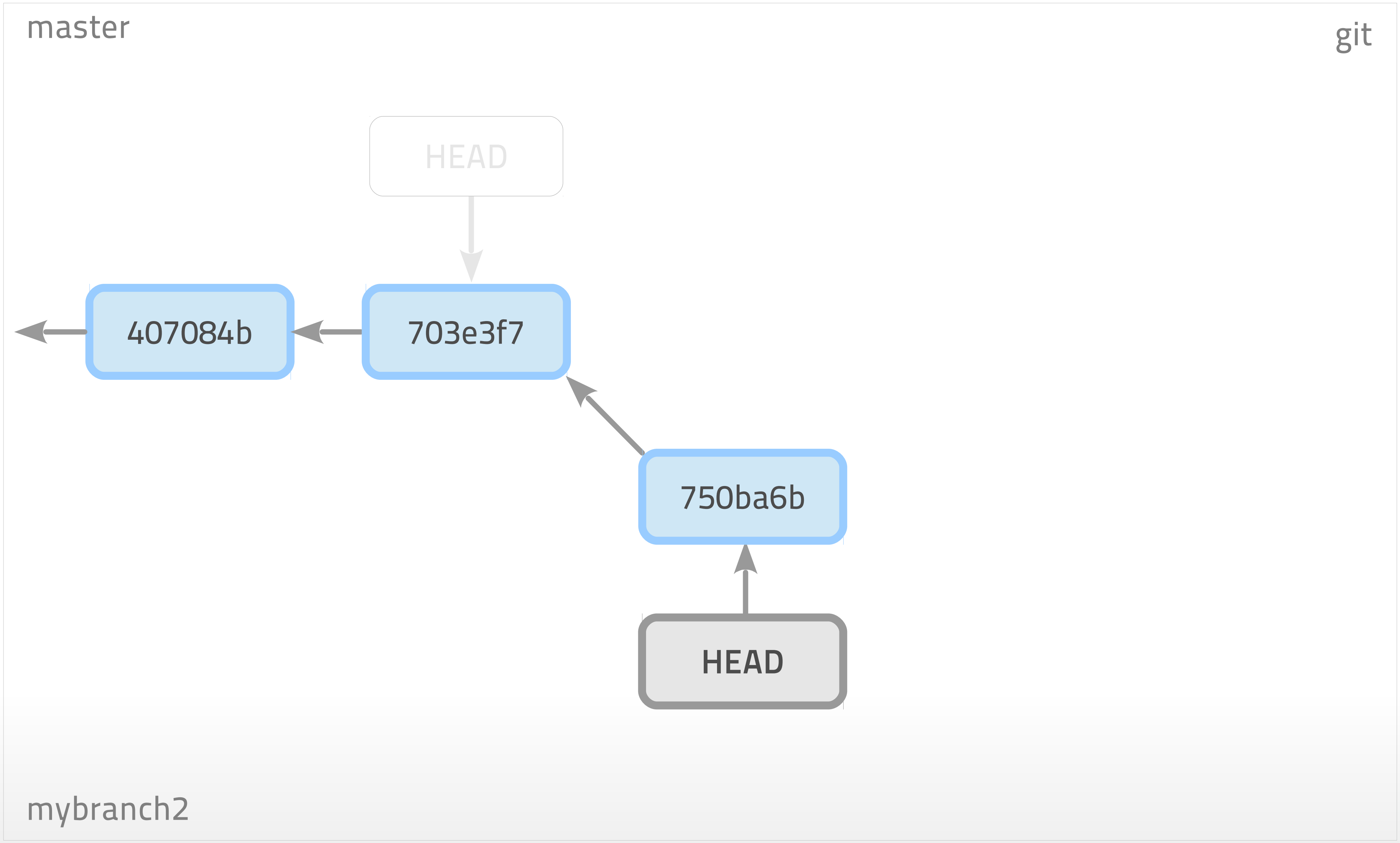


That was normal but not cool

- We fast-forwarded and lost merge info
- Fast-forward makes it look like a direct commit
- Will retain all single commits and smuggle them into the target branch
- Hard to figure out what was going on
- Use **--no-ff** all the time when merging
- Fast-forwarding is great for rebasing but not for merging (exceptions apply)

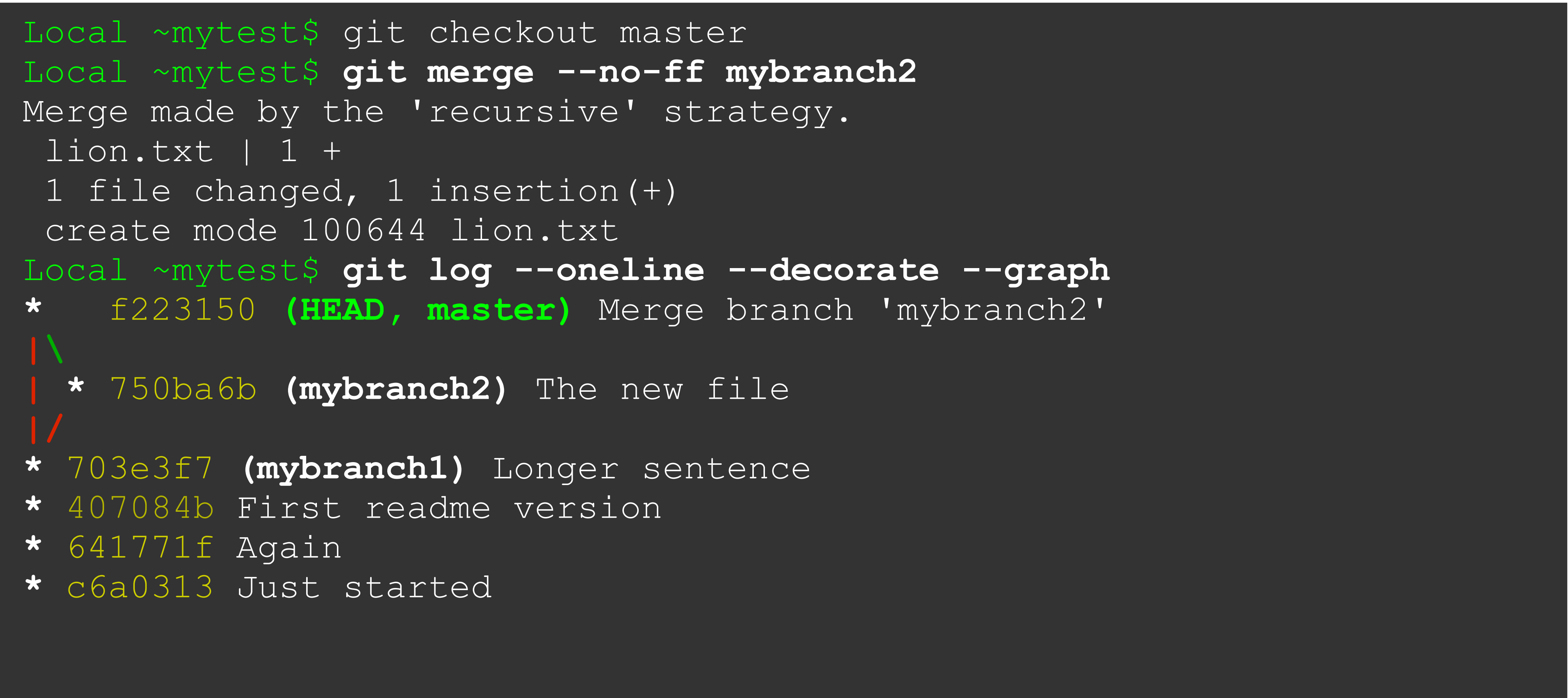


Prepare again



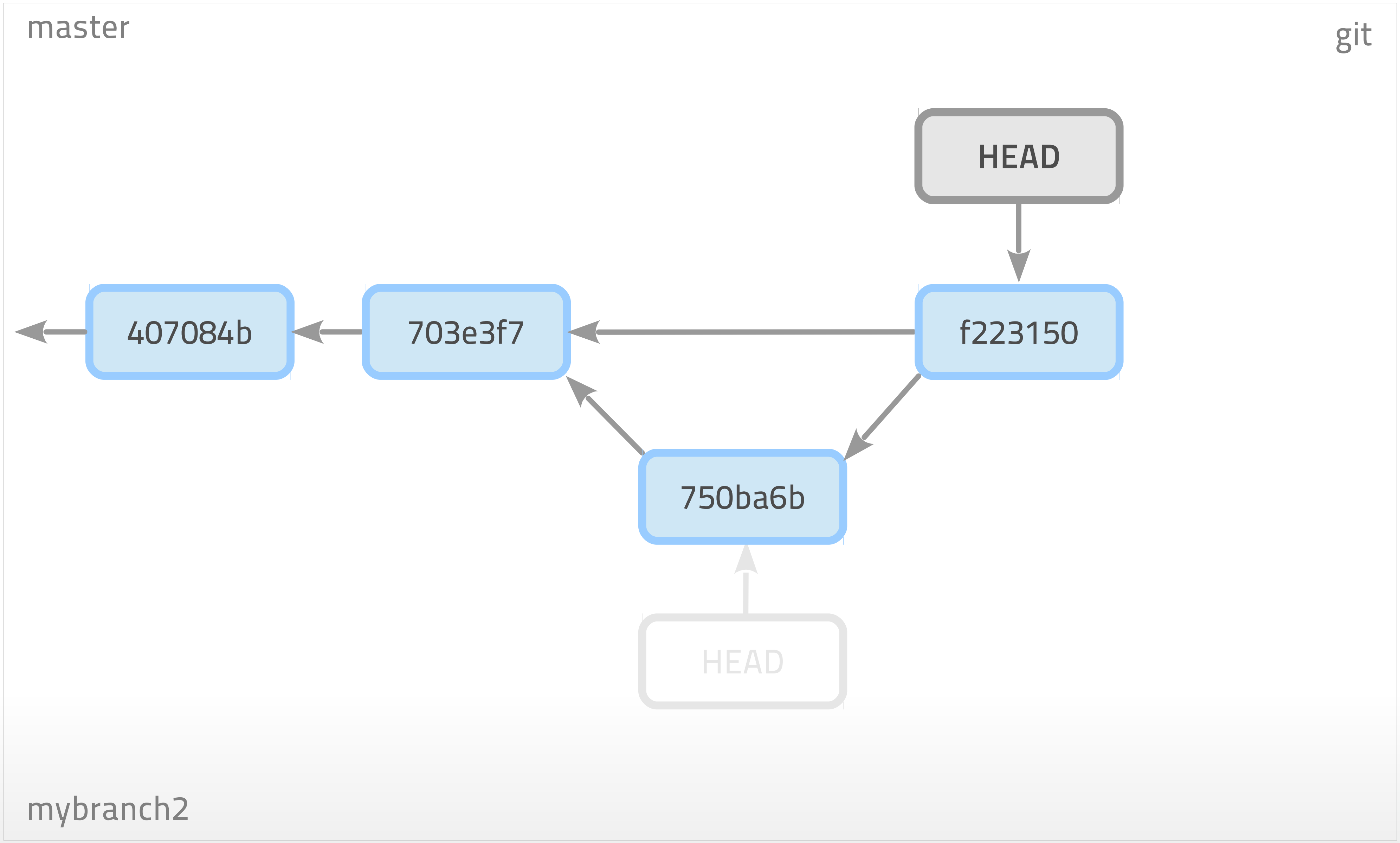
Prevent fast-forwarding

- Use --no-ff all the time



```
Local ~mytest$ git checkout master
Local ~mytest$ git merge --no-ff mybranch2
Merge made by the 'recursive' strategy.
 lion.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 lion.txt
Local ~mytest$ git log --oneline --decorate --graph
*   f223150 (HEAD, master) Merge branch 'mybranch2'
|\
| * 750ba6b (mybranch2) The new file
|/
* 703e3f7 (mybranch1) Longer sentence
* 407084b First readme version
* 641771f Again
* c6a0313 Just started
```

Perfect merge via --no-ff

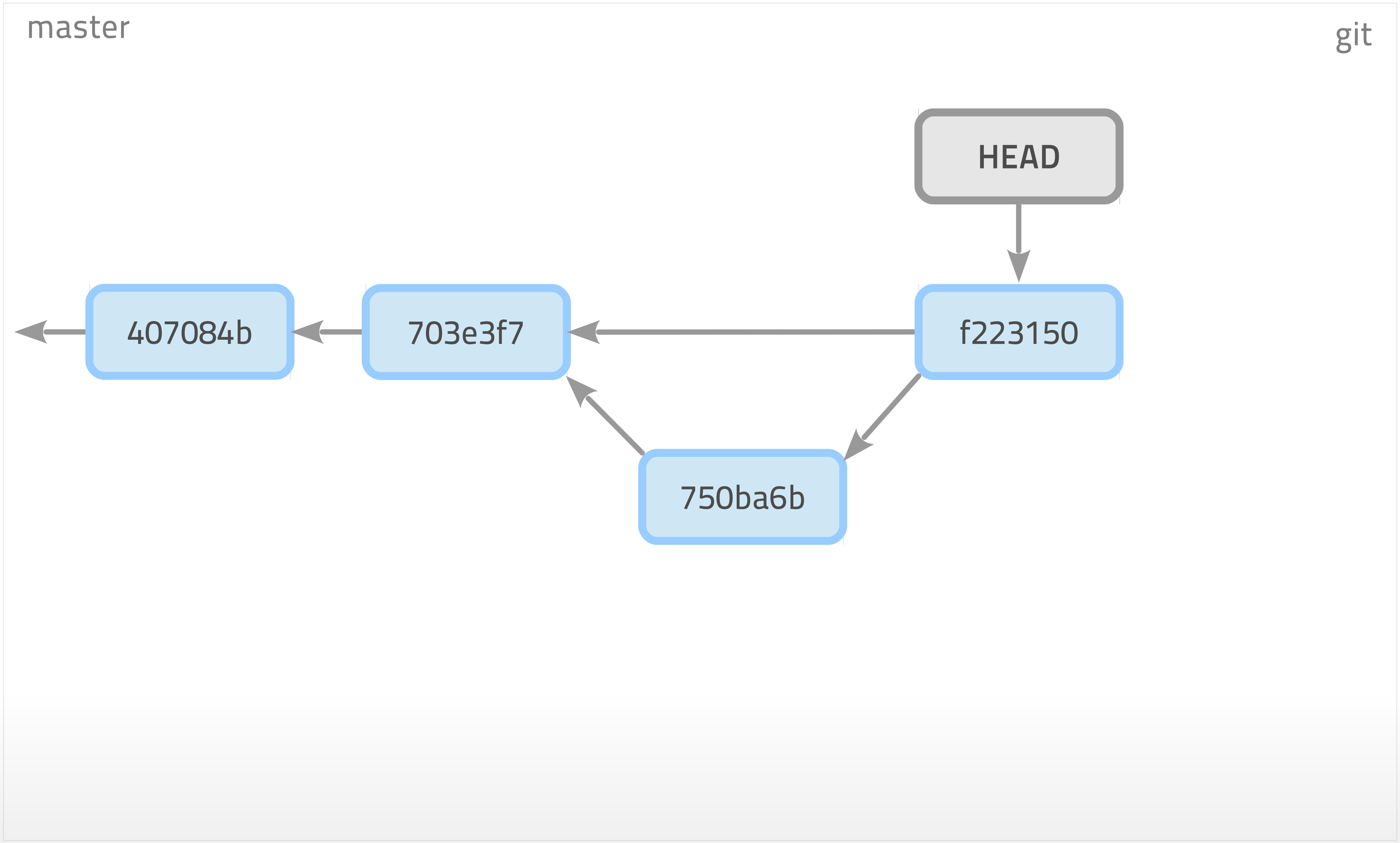


Remove merged branches

- Always clean up never not continue
- Merging means finishing up!
- More reasons: see reverting a merge

```
Local ~mytest$ git branch -d mybranch1
Deleted branch mybranch1 (was 703e3f7).
Local ~mytest$ git branch -d mybranch2
Deleted branch mybranch2 (was 750ba6b).
Local ~mytest$ git log --oneline --decorate --graph
*   f223150 (HEAD, master) Merge branch 'mybranch2'
|\
| * 750ba6b The new file
|/
* 703e3f7 Longer sentence
* 407084b First readme version
* 641771f Again
* c6a0313 Just started
```

Clean up after merges



The small helpers

Other commands you might need

git rm – remove stuff

- git rm removes and stages
- Register already deleted files with git rm

```
Local ~mytest$ git rm lion.txt
rm 'lion.txt'
Local ~mytest$ git status
# On branch remove1
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#  deleted:    lion.txt
#
Local ~mytest$ rm readme.txt
Local ~mytest$ git rm readme.txt
```

git mv – rename stuff

- git mv renames/relocates and stages
- Recommended but not required
- git tracks content not files so it guesses external moves based on content similarity

```
Local ~mytest$ git mv readme.txt README
Local ~mytest$ git status
# On branch remove1
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#  renamed:    readme.text -> README
#
Local ~mytest$ git commit -m “Adjusted to GitHub style”
[remove1 72f240a] Adjusted to Github style
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename readme.text => README (100%)
```

git diff – what changed?

- To see changes locally: now vs. HEAD

```
Local ~mytest$ change a file, e.g. README
Local ~mytest$ git diff
diff --git a/README b/README
index a3a48b3..81db0bb 100644
--- a/README
+++ b/README
@@ -1 +1 @@
-My readme and it is more
+My readme and it is more.
Local ~mytest$ git diff --word-diff
diff --git a/README b/README
index a3a48b3..81db0bb 100644
--- a/README
+++ b/README
@@ -1 +1 @@
My readme and it is [-more-]{+more.+}
Local ~mytest$ git diff README
...
```

git diff – what changed?

- To see changes of commits

```
Local ~mytest$ git log <you know the rest>
* 079aaab (HEAD, origin/master, master) master 3
* 7ba35ee master 2
* cad3fee master 1
Local ~mytest$ git show 079aaab
What changed with 079aaab
Local ~mytest$ git diff 7ba35ee
What changed since 7ba35ee aka the change introduced by 079aaab
Local ~mytest$ git diff 703e3f7 README
Same as before, but limited to a specific file
Local ~mytest$ git log --since=2.weeks readme.txt
List all commits that touched readme.txt in the last two weeks.
Local ~mytest$ git log --since=2.weeks -G Rene
Where 'Rene' was in the changes including removals in the last two weeks.

You can go nuts here. Google it when you need it. It is powerful and allows
basically everything possible to find and identify changes and problems.
```

git mergetool – conflict solver

- Solve conflicts during rebasing and merging
- Uses the configured tool of your choice

```
Local ~mytest$ git rebase origin/qa
First, rewinding head to replay your work on top of it...
Applying: Conflict
Using index info to reconstruct a base tree...
M   qa/start.txt
Falling back to patching base and 3-way merge...
Auto-merging qa/start.txt
CONFLICT (content): Merge conflict in qa/start.txt
Failed to merge in the changes.
Patch failed at 0001 Conflict
The copy of the patch that failed is found in:
   /tmp/learn_git/.git/rebase-apply/patch

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase -abort".
Local ~mytest$ git mergetool
Local ~mytest$ git rebase --continue
```

git cherry-pick – simply the best

- Rescue commits
- Preserves all data but the SHA-1 because of the new predecessor
- Good for downstream hotfix merges
- Use to preserve pieces of a dead branch

```
Local ~mytest (master)$ git checkout -b kirsche
Local ~mytest (kirsche)$ git cherry-pick 56fe781
[tets b8c3f20] New readme file
 1 file changed, 1 insertion(+)
```

Dealing with Oops!

The reason for version control

Forgot something? --amend

- Changes the last commit (when not pushed!)
- **Removes** old commit, **creates** new commit

```
Local ~mytest$ change two files
Local ~mytest$ git commit -m “Updated two files”
Local ~mytest$ git log --stat
* 5cb60ce (HEAD, master) Updated two files
|  foobar.txt | 1 +
|  1 file changed, 1 insertion(+)
* 0da605d Merge branch 'oauth.fix'
Local ~mytest$ git add file2.txt
Local ~mytest$ git commit --amend -m “Really updated two files”
Local ~mytest$ git log --stat
* c3ae1b1 (HEAD, master) Really updated two files
|  foobar.txt | 1 +
|  readme.txt | 1 +
|  2 files changed, 2 insertions(+)
* 0da605d Merge branch 'oauth.fix'
```

Staged too much?

- Remove from the staging area
- Not very intuitive… sadly enough

```
Local ~mytest$ git add test.txt
Local ~mytest$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#  new file:   test.txt
#
Local ~mytest$ git reset HEAD test.txt
Local ~mytest$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#  test.txt
nothing added to commit but untracked files present (use "git add" to
track)
```

Get rid off the last commit

- Kills the last commit without a trace
- Of course, only when not shared yet

```
Local ~mytest$ git log
* 4db3017 (HEAD, master) Too much, oops
* a0a842d Fine 2
* 99ca82a Fine 1
Local ~mytest$ git reset HEAD~1
Unstaged changes after reset:
M   readme.txt
Local ~mytest$ git log
* a0a842d (HEAD, master) Fine 2
* 99ca82a Fine 1
```

git revert – undo things I

- Reverting means committing reversed content
- Step 1: Identify hash aka revision

```
Local ~mytest$ git log --oneline --graph --decorate
*   01d9051 (HEAD, master) Cleaner now
|\
| * 72f240a Adjusted to Github style
| * 5a9e51a Redundant file removed
|/
*   f223150 Merge branch 'mybranch2'
|\
| * 750ba6b The new file
|/
* 703e3f7 Longer sentence
* 407084b First readme version
* 641771f Again
* c6a0313 Just started
```

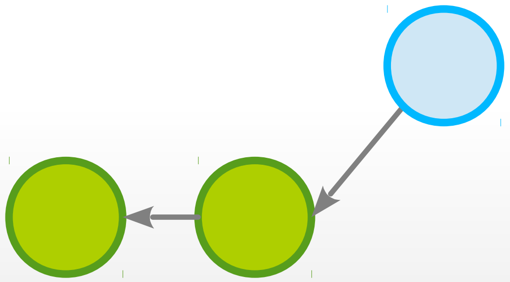
git revert – undo things II

- Step 2: Undo the work with a "negative" commit
- Revert a revert the same way

```
Local ~mytest$ git revert 5a9e51a
[master 7b6e6cf] Revert "Redundant file removed"
 1 file changed, 1 insertion(+)
 create mode 100644 lion.txt
Local ~mytest$ git log <you know the rest> --stat
* 7b6e6cf (HEAD, master) Revert "Redundant file removed"
|  lion.txt | 1 +
|  1 file changed, 1 insertion(+)
*   01d9051 Cleaner now
|\
| * 72f240a Adjusted to Github style
| |  README      | 1 +
| |  readme.text | 1 -
| |  2 files changed, 1 insertion(+), 1 deletion(-)
| * 5a9e51a Redundant file removed
|/
|    lion.txt | 1 -
|    1 file changed, 1 deletion(-)
...
```

git revert – revert a merge

- Merges are reverted like any other commit
- But when you continue to work on it and merge later, it does not work as expected
- Git does not track the content, but the commits
- When commits are already in the tree, the next merge with them again makes no difference aka they are still reverted
- Revert the revert in these cases first



git revert – revert a merge

```
Local ~mytest (master)$ git log
* 5169844 (HEAD, master) Foo is new
* 35a597f Merge branch 'test1'
|\
| * 52212e1 A change in readme
|/
* bbd0c7b (origin/master) Readme 2

# identify predecessor to use
Local ~mytest (master)$ git show 35a597f
Merge: bbd0c7b 52212e1
Author: Peter Meier <r.schwietzke@xceptance2.net>
...
# revert 35a597f
Local ~mytest (master)$ git revert -m 1 35a597f
# create branch to continue
Local ~mytest (master)$ git checkout -b fix-it
# revert the revert in the branch to see original code
Local ~mytest (fix-it)$ git revert 42635ec
# fix all the bad stuff and merge
Local ~mytest (fix-it)$ edit, add, commit...
Local ~mytest (fix-it)$ git checkout master
Local ~mytest (master)$ git merge --no-ff fix-it
# clean up
Local ~mytest (master)$ git branch -d fix-it
```

git revert – revert a merge

- 42635ec – revert in main branch
- b8796e0 – revert of the revert to continue later
- 70e18de – second merge attempt including fix

```
Local ~mytest (master)$ git log
* 70e18de (HEAD, master) Merge branch 'fix-it', previously test1
|\
| * d980ef0 Fixed the bad stuff
| * b8796e0 Revert of "Revert "Merge branch 'test1'""
|/
* 42635ec Revert "Merge branch 'test1'"
* 5169844 Foo is new
* 35a597f Merge branch 'test1'
|\
| * 52212e1 A change in readme
|/
* bbd0c7b (origin/master) Readme 2
```

git reset/checkout – trash it

- Kill or revert all your uncommitted changes

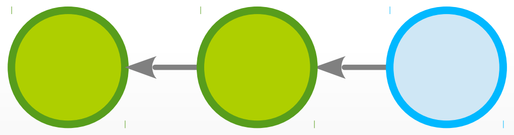
```
# git rid off all changes
Local ~mytest$ change a file, e.g. README
Local ~mytest$ git reset --hard
HEAD is now at 25f967d ! The was the message.

# remove from staging area
Local ~mytest$ change a file
Local ~mytest$ git reset README
Unstaged changes after reset:
M   README

# get README back to last commited state
Local ~mytest$ git checkout README
Local ~mytest$ git status
# On branch master
nothing to commit, working directory clean
```

It is called branching, not treeing

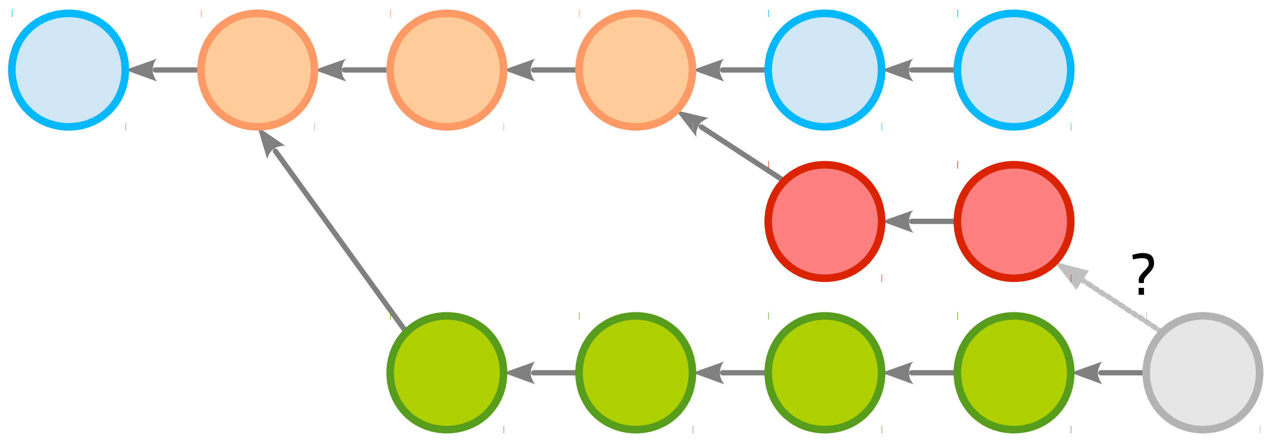
Do not build trees from your branches

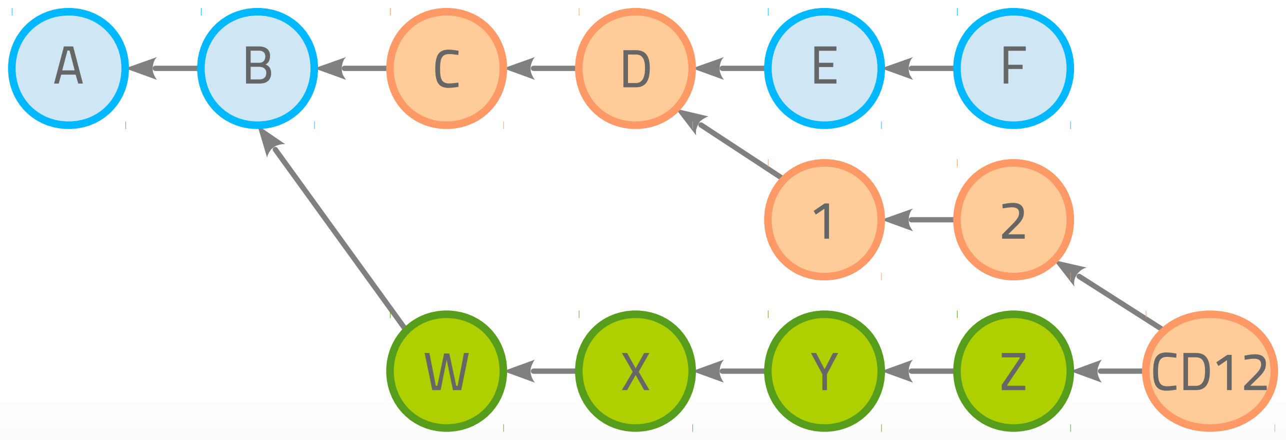
Branches – important to know

- Remember, every commit has at least one parent
- So your branch is based on something
- If you merge your branch to a place you have not started from, you get everything merged that is not yet in... not just what you have touched
- Git makes sure that nothing is lost
- A branch is just a pointer!

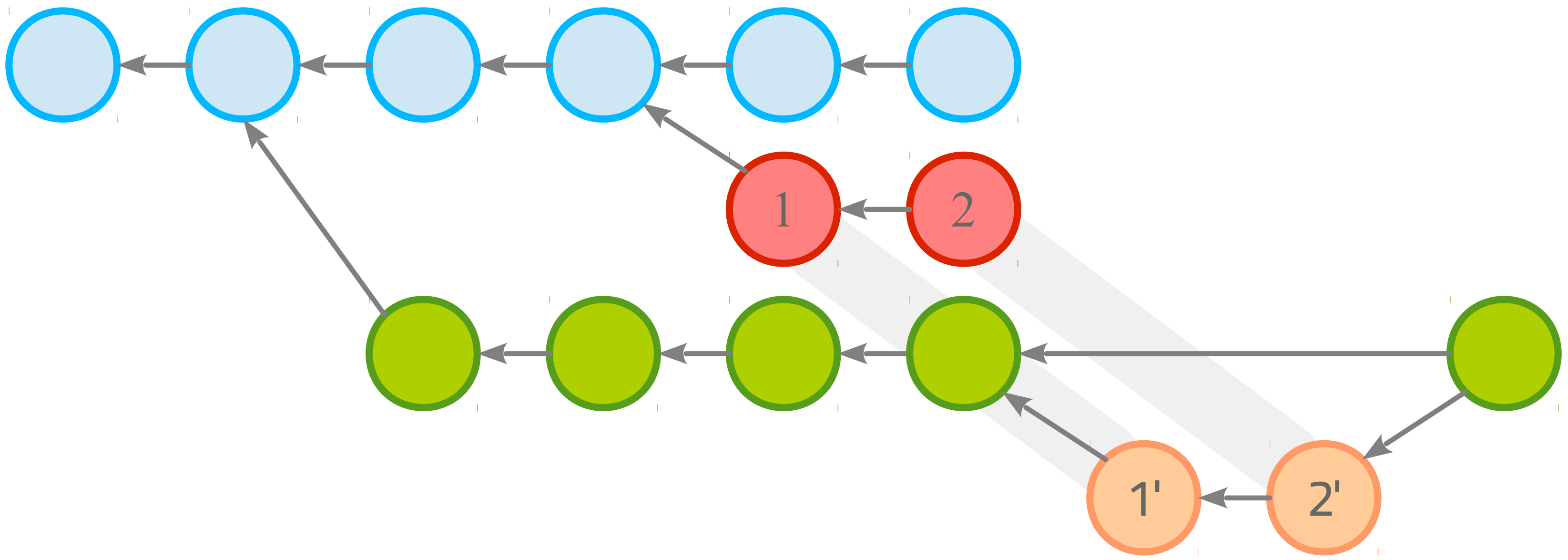


Branches – the unexpected





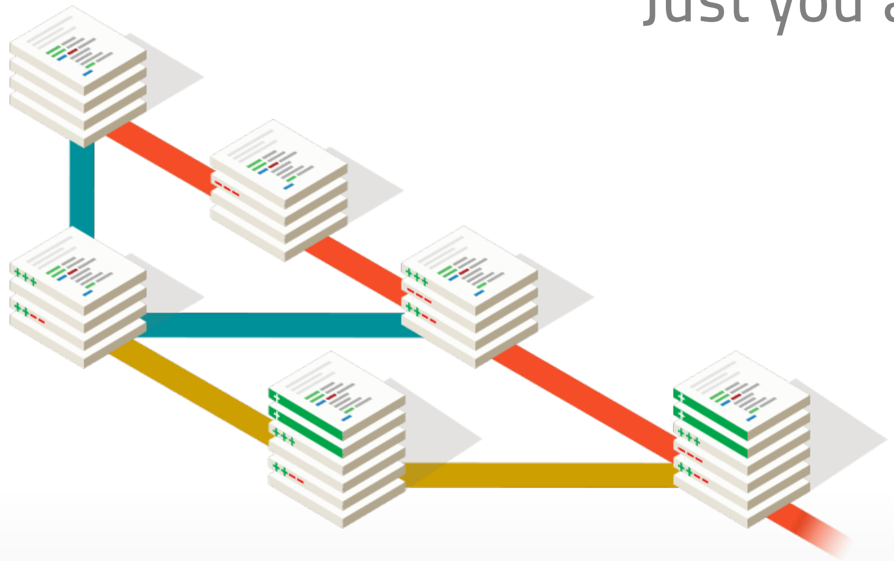
Branches – cherry-pick from red



```
Local (red)$ git checkout -b get-red-in
Switched to new branch 'get-red-in'
Local (get-red-in)$ git cherry-pick 1 2
Local (get-red-in)$ git checkout green
Switched to branch 'green'
Local (green)$ git merge --no-ff get-red-in -m "Cherry picked the red"
...
Local (green)$ git branch -d get-red-in
```

A small workflow

Just you and git. Local and simple.



Small and quick local workflow

- Practice the workflow in your local world

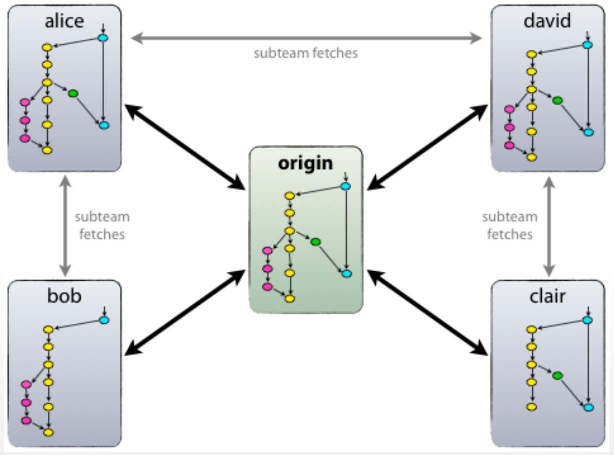
```
Local (abranch)$ git checkout master
Switched to branch 'master'
Local (master)$ git checkout -b new-branch

Local (new-branch)$ edit and commit often

Local (new-branch)$ git rebase master
Local (new-branch)$ git checkout master
Local (master)$ git merge --no-ff new-branch -m “Cleaner now”
Removing lion.txt
Merge made by the 'recursive' strategy.
 readme.text => README | 0
 lion.txt              | 1 -
 2 files changed, 1 deletion(-)
 rename readme.text => README (100%)
 delete mode 100644 lion.txt
Local (master)$ git branch -d new-branch
```

Let's go remote now

Play with others in the same sandbox

Let's go remote

- Every repository is a complete repo
- Except for unsynced changes with “friends”
- No obligation to use a remote
- More than one remote possible
- No obligation to upload (push) everything
- Only what you want to share
- Only what you have to protect



git clone – get a remote repo

- Get a remote repo downloaded
- Git automatically sets up tracking

```
Local ~mytest$ git clone
ssh://git@git0.foobar.net:7999/train/learn_git.git
Cloning into 'learn_git'...
remote: Counting objects: 192, done.
remote: Compressing objects: 100% (160/160), done.
remote: Total 192 (delta 78), reused 0 (delta 0)
Receiving objects: 100% (192/192), 20.35 KiB | 0 bytes/s, done.
Resolving deltas: 100% (78/78), done.
Checking connectivity... done
Local ~mytest$ cd learn_git
Local ~mytest$ git status
* master
Local ~mytest$ git remote --verbose
origin ssh://git@git0.foobar.net:7999/train/learn_git.git (fetch)
origin ssh://git@git0.foobar.net:7999/train/learn_git.git (push)
```

git fetch – get work from others

- Fetches data from remote
- Does not touch your work

```
Local ~mytest$ git fetch origin
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From ssh://git0.svc1.foobar.net:7999/train/learn_git
   aa2e115..d85515d  qa -> origin/qa
Local ~mytest$ git diff aa2e115..d85515d
diff --git a/qa/start.txt b/qa/start.txt
index 006f308..067ab29 100644
--- a/qa/start.txt
+++ b/qa/start.txt
@@ -1,3 +1,5 @@
 This is beginning of a text.
 It might continue.
 Or it might now.
+I am remote.
```

git merge/rebase – pull it in

- Works like a local rebase or merge
- Picks the way that fits best
- Merge to make operation visible
- Rebase to just get you updated to latest

```
Local ~mytest$ git fetch origin
Local ~mytest$ git merge origin/qa
Merge made by the 'recursive' strategy.
 qa/start.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
Local ~mytest$ git rebase origin/qa
First, rewinding head to replay your work on top of it...
```

git pull – fetch and merge

- Pull is fetch and merge in one command
- There is no --no-ff option, git picks the best way
- So prefer --rebase when content is unpushed

```
Local ~mytest$ git pull origin mybranch
From ssh://git0.foobar.net:7999/train/learn_git
 * branch            mybranch -> FETCH_HEAD
Updating 0d1c02a..974c00d
Fast-forward
 qa/start.txt | 2 ++
 1 file changed, 2 insertions(+)
Local ~mytest$ git pull --rebase origin mybranch
```

git push – deliver your work

- Deliver your work to a remote destination
- Nothing changes (commit hashes)
- DO NOT CROSS PUSH

```
Local ~mytest$ branch, edit, commit
Local ~mytest$ git push origin qa
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 377 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To ssh://git@git0.foobar.net:7999/train/learn_git.git
 * [new branch]      qa -> qa
Local ~mytest$ git log
* 0743e1d (HEAD, origin/qa, qa) QA directory
*   d0400d5 (origin/master, origin/HEAD, master) Merge pull request
#12...
...
```

git push – too slow and you fail

- Someone was faster
- You have to get the remote changes first

```
Local ~mytest$ branch, edit, commit
Local ~mytest$ git push origin master
To ssh://git@gitlab.foobar.de/test/training.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to
'ssh://git@gitlab.foobar.de/test/training.git'
hint: Updates were rejected because the remote contains work that you
do not have locally. This is usually caused by another repository
pushing to the same ref. You may want to first integrate the remote
changes (e.g., 'git pull ...') before pushing again.
See the 'Note about fast-forwards' in 'git push --help' for details.
Local ~mytest$ git log
* 3fc556d (HEAD, master) New text
*   ba74a5a (origin/master) Merge branch 'michael'
|\
| * bd8bf6e Micha branch
* | 57cdc9a Code addded
|/
```

git pull --rebase

- You have not shared the commit, so rebase for clean history with the remote content.
- See, no non-sense merge!

```
Local ~mytest$ git log
* 3fc556d (HEAD, master) New text
*   ba74a5a (origin/master) Merge branch 'michael'
...
Local ~mytest$ git pull --rebase origin master
...
First, rewinding head to replay your work on top of it...
Applying: New text
Local ~mytest$ git log
* 85f62f6 (HEAD, master) New text
* c2e9a1a (origin/master) One more thing added
*   ba74a5a Merge branch 'michael'
|\
| * bd8bf6e Micha branch
...
```

git pull – the ugly merge

- By default you get a merge when you have local changes

```
Local ~mytest$ git log
* 2272ff5 (HEAD, master) No stuff anymore
* f5d6db1 (origin/master) Entry 10
* fd4fdd7 Entry 9
Local ~mytest$ git pull origin master
...
From ssh://gitlab.foobar.de/asdf/traning
 * branch            master     -> FETCH_HEAD
   f5d6db1..d39a17e  master     -> origin/master
Merge made by the 'recursive' strategy.
 asdf.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 asdf.tct
Local ~mytest$ git log
*   d1247b3 (HEAD, master) Merge branch 'master' of ssh://gitlab.foobar.de...
|\
| * d39a17e (origin/master) New file
* | 2272ff5 No stuff anymore
|/
* f5d6db1 Entry 10
```

A small shared workflow

Just you, git, and other dudes or dudettes.

Let me flow alone

```
# get the master latest changes
Local ~(master)$ git pull --rebase origin master
# create a new branch with a meaningful name and switch
Local ~(master)$ git checkout -b jaropener-4210
# do some development
Local ~(jaropener-4210)$ git add File.java; git commit -m “base class”
Local ~(jaropener-4210)$ git add .; git commit -m “tests added”
# switch to master
Local ~(jaropener-4210)$ git checkout master
# get the latest master changes, no local changes assumed
Local ~(master)$ git pull --rebase origin master
# switch to branch
Local ~(master)$ git checkout jaropener-4210
# rebase branch against master, assuming you have not pushed your branch!
Local ~(jaropener-4210)$ git rebase master
# switch to master
Local ~(jaropener-4210)$ git checkout master
# merge the branch in without fast-forward to get a single commit
Local ~(master)$ git merge --no-ff jaropener-4210
# delete the branch now, you merged it!
Local ~(master)$ git branch -d jaropener-4210
# refresh your master, no further local changes
Local ~(master)$ git pull --rebase origin master
# push your work and be happy
Local ~(master)$ git push origin master
```

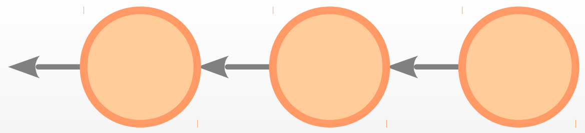
Merge vs. Rebase

The common misunderstanding

But SVN...

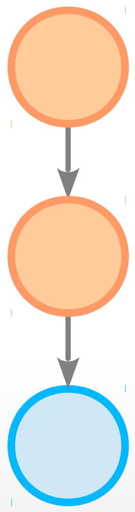
- SVN does not have a rebase
- Server counts up and each commit gets a number
- What is merged ends in mergeinfo properties

- Git uses a cryptographic hash to protect content
- Includes the parents in that aka new parent, new hash
- It is non-linear but linear... confused?
- Git forms a chain of content

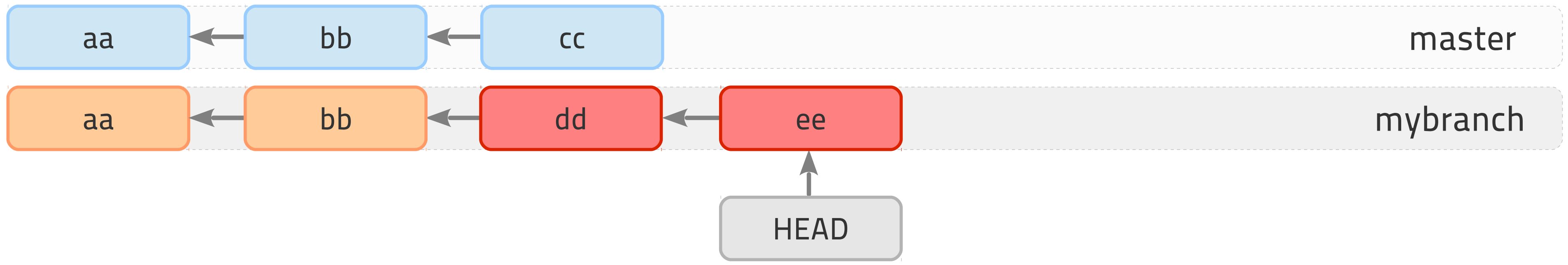


Rebase vs. Merge

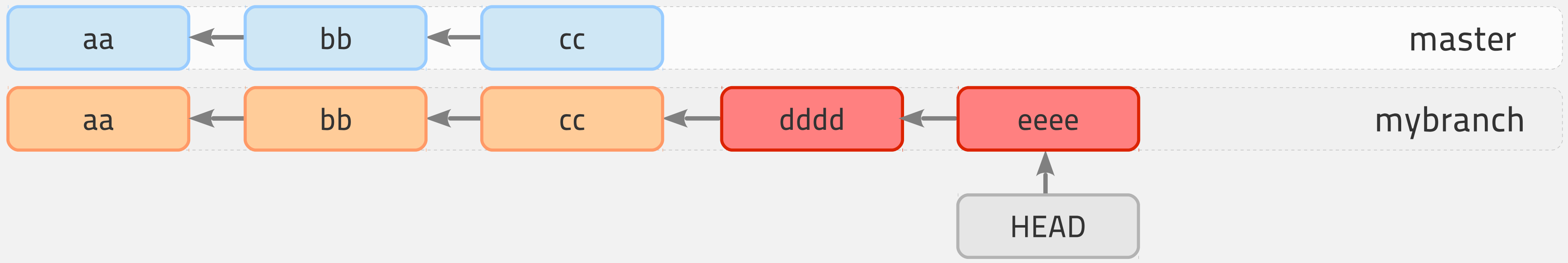
- Rebase
  - Makes your work latest
  - Set's your work aside, updates branch, puts your work on top of it, so your work wins mostly
  - **Removes** all your commits and **creates** them new
  - Bad when already shared
- Merge
  - Fits your work into the timeline
  - Uses the timeline to figure out who is right
  - Creates a commit with the merge content



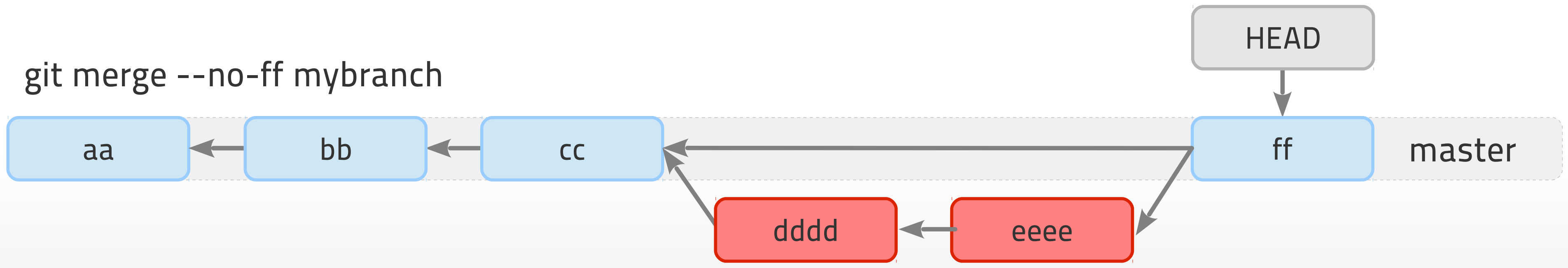
Local rebase and merge visualized



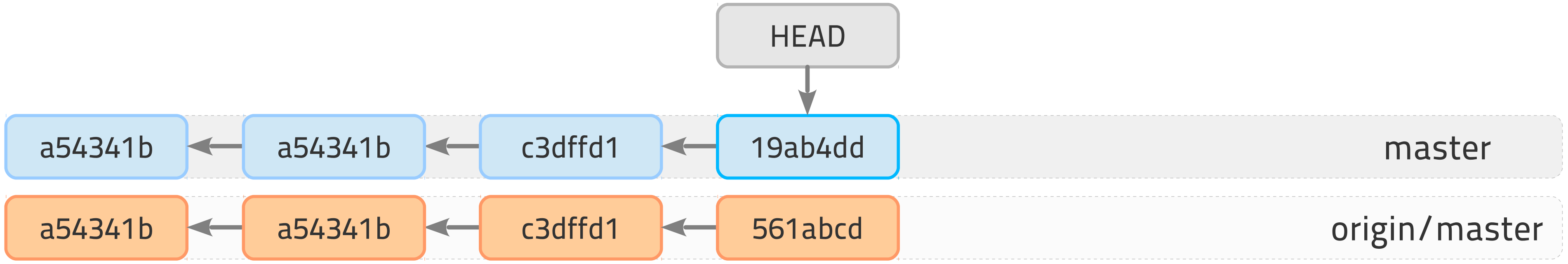
git rebase master



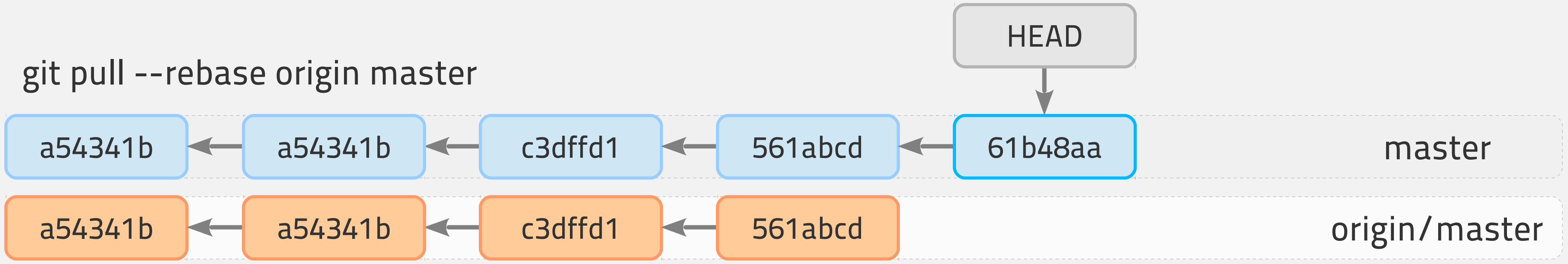
git merge --no-ff mybranch



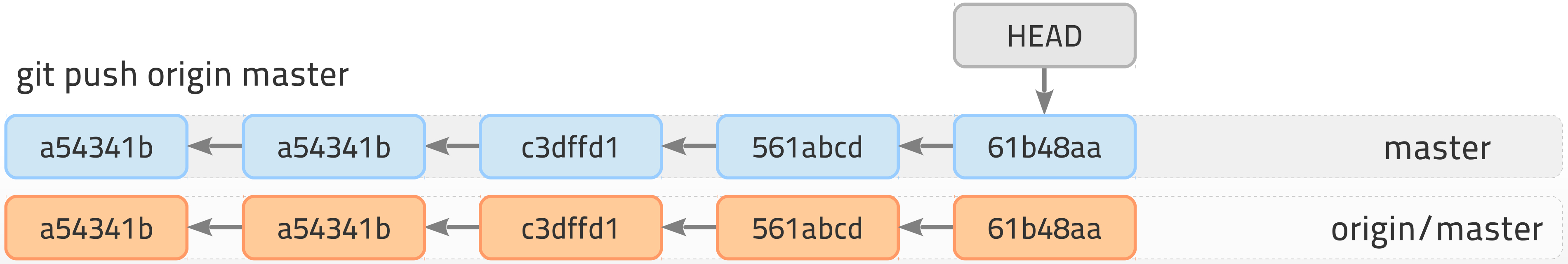
Rebase from remote visualized



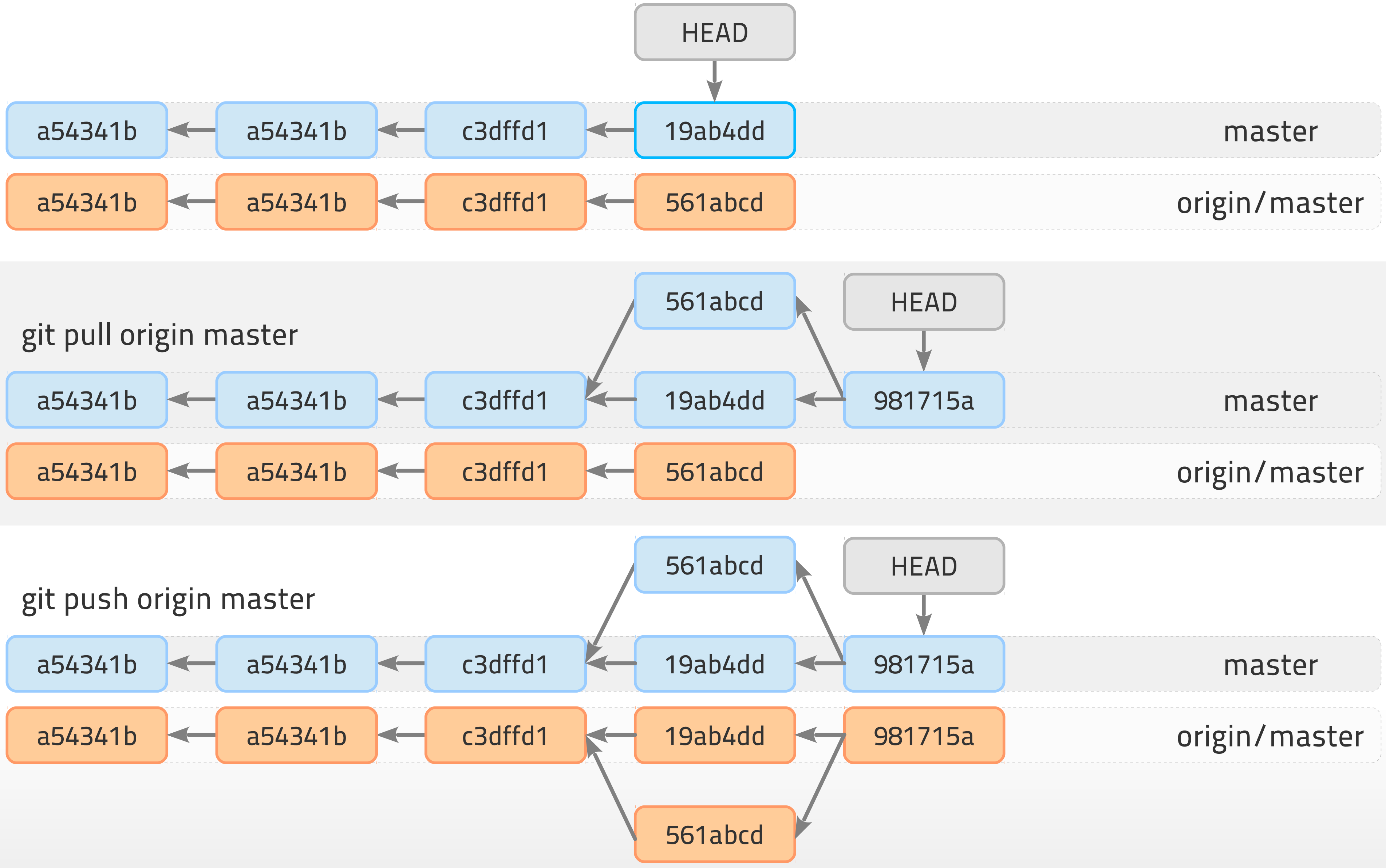
git pull --rebase origin master



git push origin master

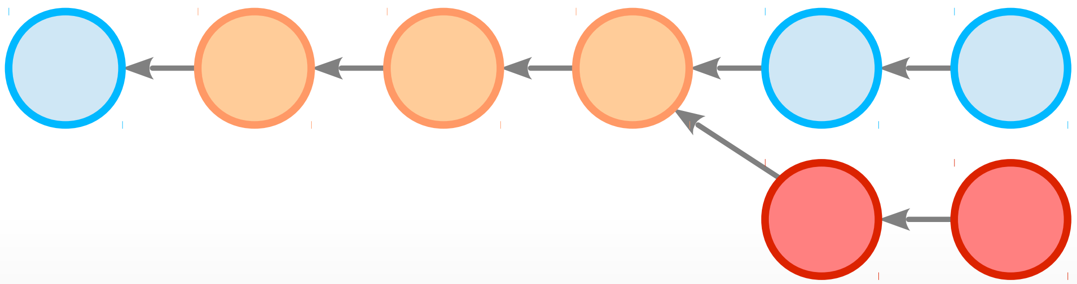


Remote merge visualized



When to rebase

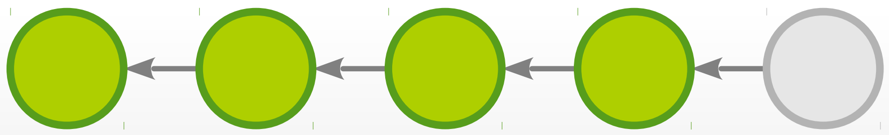
- When you cannot push because someone was faster: git pull --rebase
- When you have not shared anything and want to have a clean branch and solve the possible merge conflicts before merging
- When you do not care about your base



When rebase becomes evil

- When you shared commits already
- When you branch off of the to-be-rebased branch aka you shared you commit with yourself already
- When you already merged in from other branches
- Do not forget, squashing means rewriting too

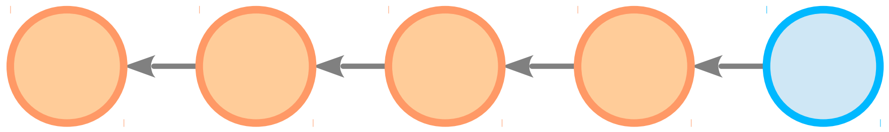
Rebasing means rewriting of commits!



Rebase kills merges

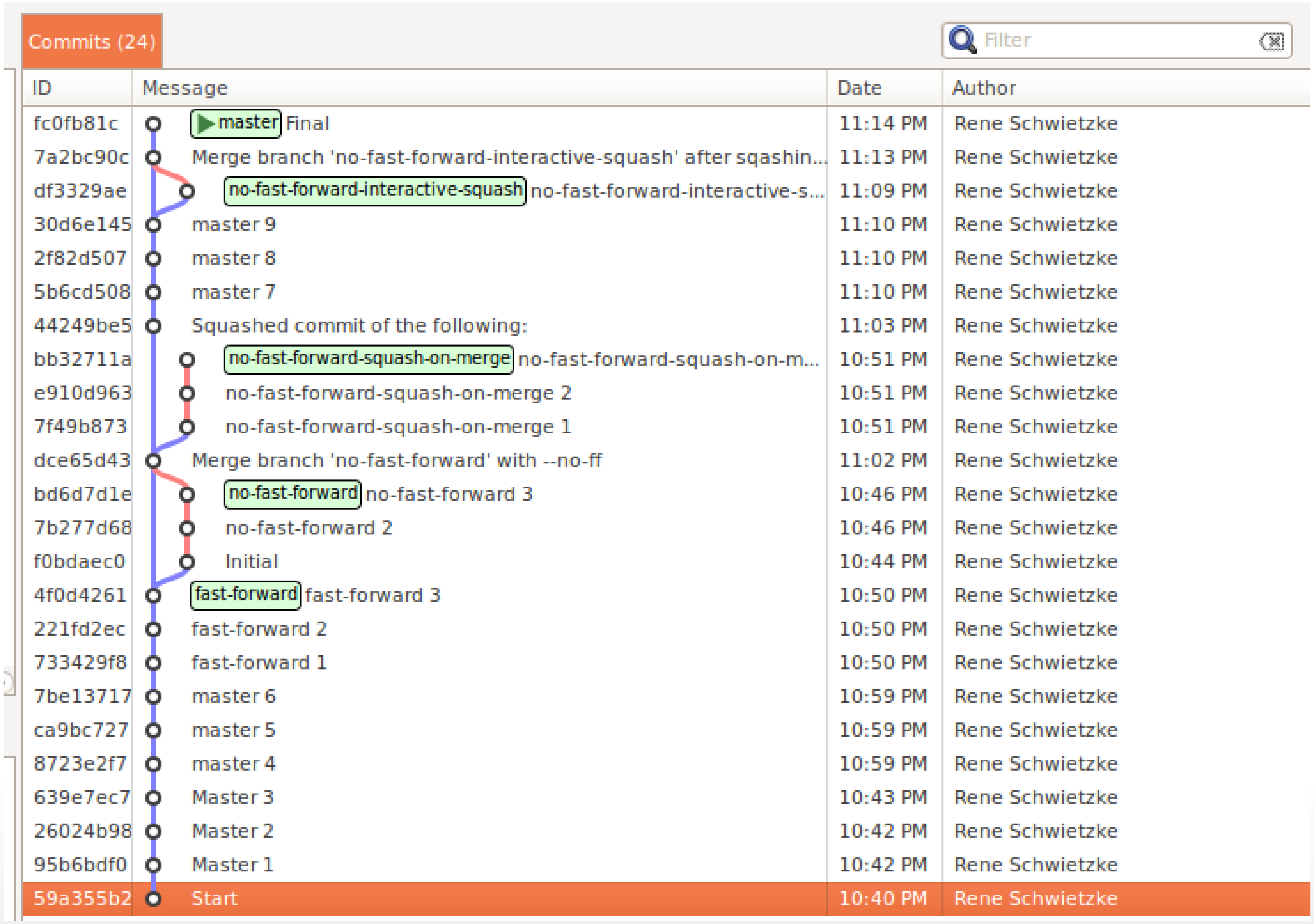
- A rebase after a merge removes the merge and makes everything flat
- How ugly!

http://stackoverflow.com/questions/11863785/make-git-pull-rebase-preserve-merge-commits



```
Local ~mytest$ git fetch origin/master
Local ~mytest$ git rebase origin/master --preserve-merges

# Make it default to preserve (Git >= 1.8.5)
Local ~mytest$ git config pull.rebase preserve
```

Merge/no-ff/squash?



Commits (24)

Filter

ID	Message	Date	Author
fc0fb81c	master Final	11:14 PM	Rene Schwietzke
7a2bc90c	Merge branch 'no-fast-forward-interactive-squash' after sqashin...	11:13 PM	Rene Schwietzke
df3329ae	no-fast-forward-interactive-squash no-fast-forward-interactive-s...	11:09 PM	Rene Schwietzke
30d6e145	master 9	11:10 PM	Rene Schwietzke
2f82d507	master 8	11:10 PM	Rene Schwietzke
5b6cd508	master 7	11:10 PM	Rene Schwietzke
44249be5	Squashed commit of the following:	11:03 PM	Rene Schwietzke
bb32711a	no-fast-forward-squash-on-merge no-fast-forward-squash-on-m...	10:51 PM	Rene Schwietzke
e910d963	no-fast-forward-squash-on-merge 2	10:51 PM	Rene Schwietzke
7f49b873	no-fast-forward-squash-on-merge 1	10:51 PM	Rene Schwietzke
dce65d43	Merge branch 'no-fast-forward' with --no-ff	11:02 PM	Rene Schwietzke
bd6d7d1e	no-fast-forward no-fast-forward 3	10:46 PM	Rene Schwietzke
7b277d68	no-fast-forward 2	10:46 PM	Rene Schwietzke
f0bdaec0	Initial	10:44 PM	Rene Schwietzke
4f0d4261	fast-forward fast-forward 3	10:50 PM	Rene Schwietzke
221fd2ec	fast-forward 2	10:50 PM	Rene Schwietzke
733429f8	fast-forward 1	10:50 PM	Rene Schwietzke
7be13717	master 6	10:59 PM	Rene Schwietzke
ca9bc727	master 5	10:59 PM	Rene Schwietzke
8723e2f7	master 4	10:59 PM	Rene Schwietzke
639e7ec7	Master 3	10:43 PM	Rene Schwietzke
26024b98	Master 2	10:42 PM	Rene Schwietzke
95b6bdf0	Master 1	10:42 PM	Rene Schwietzke
59a355b2	Start	10:40 PM	Rene Schwietzke

Misc

Does not fit anywhere else

Some other stuff

- Force branch delete: **git branch -D name**
- Remote delete: **git push origin :name**
- Stash things away: **git stash save**
- Get things back: **git stash apply**
- Remove stash: **git stash drop**
- Clean up after merge: **git clean -f -d**

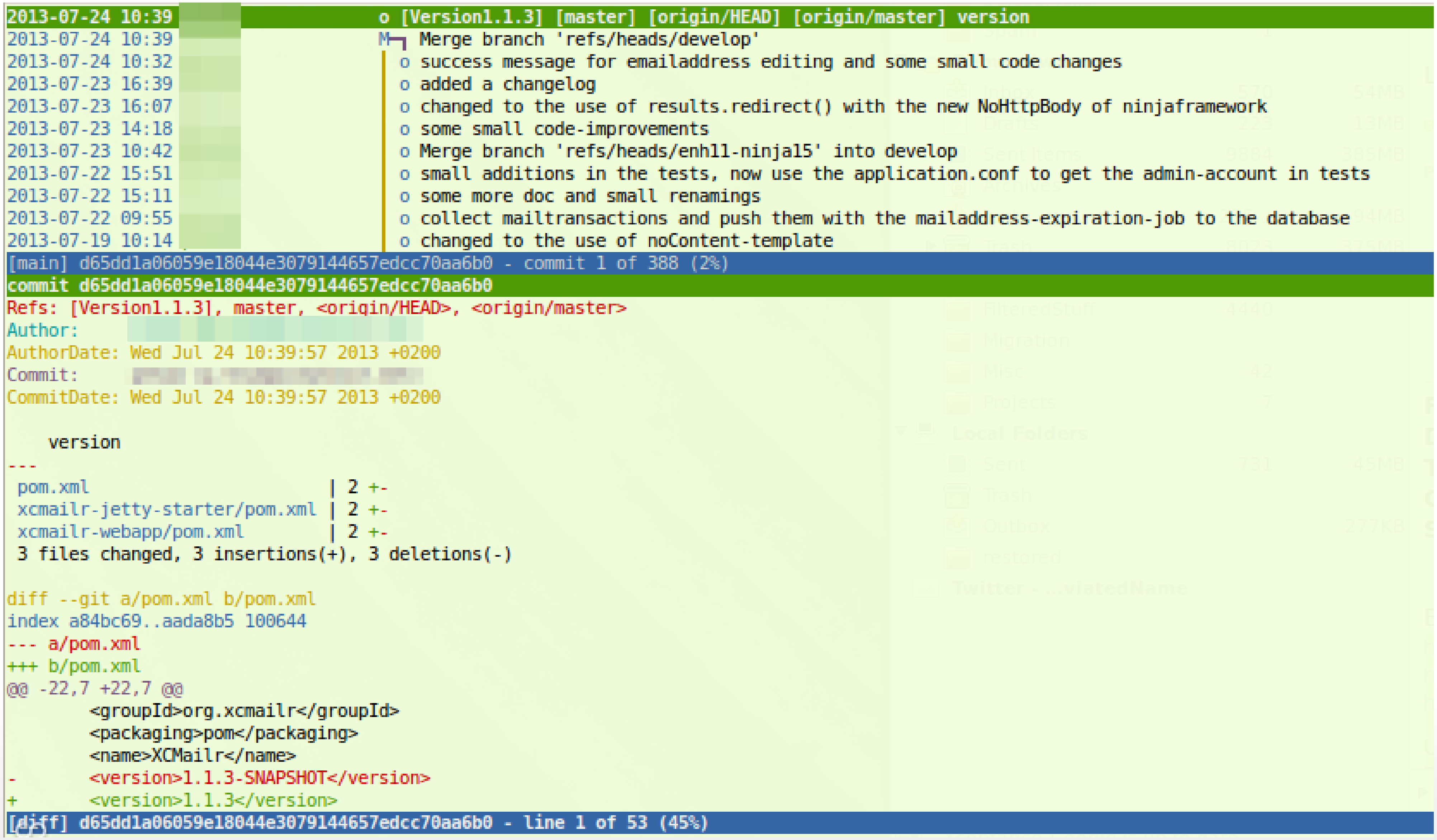
Golden git advantages to use

- Always branch
- Commit often
- Not sure about a merge? New branch and try
- Rebase your branch before merge, if not shared
- Do not run anything advanced, there is barely any reason for doing that
- Keep it simple!

Tools

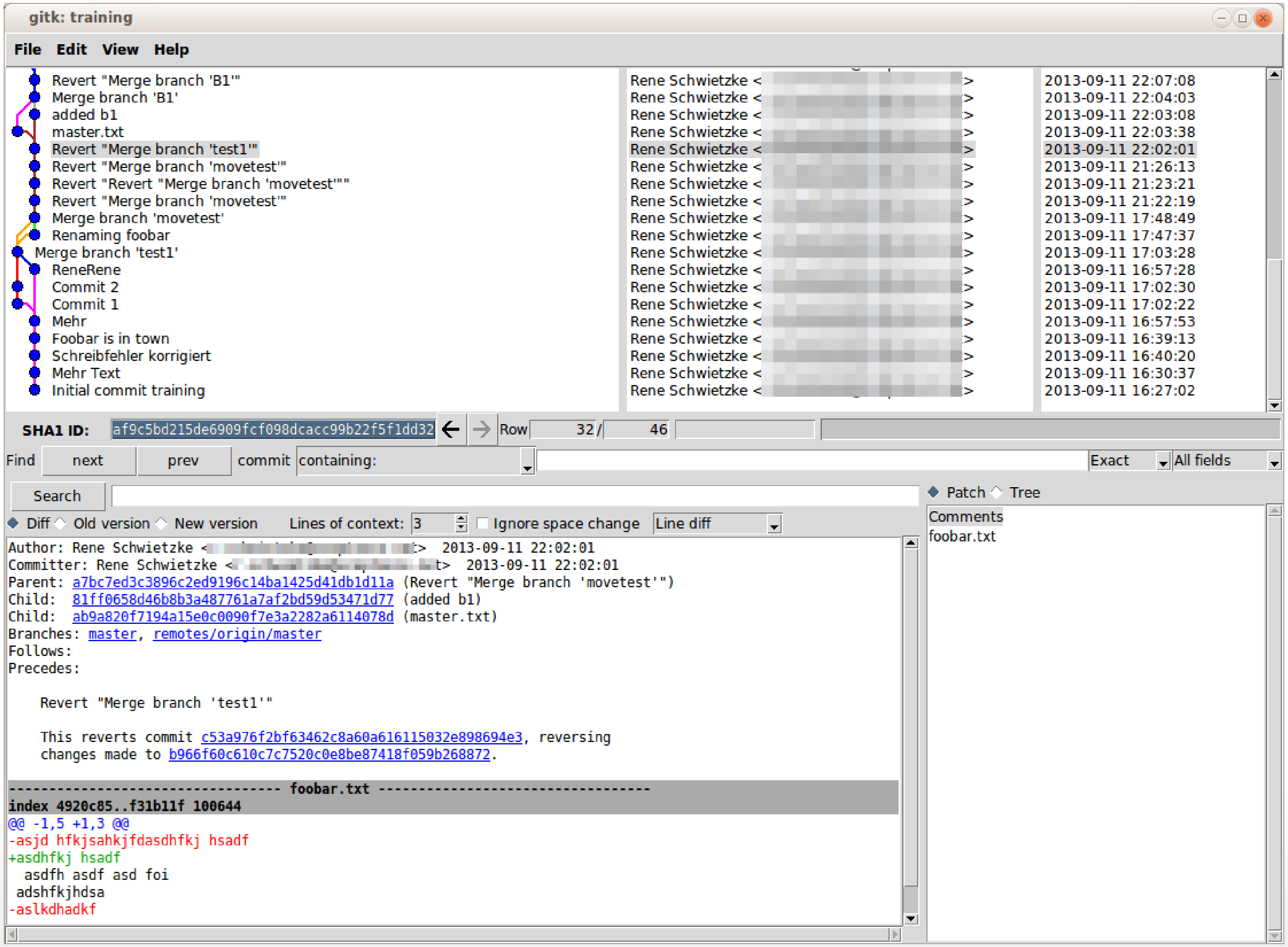
In case you don't like the command line

Tools - tig



```
2013-07-24 10:39          o [Version1.1.3] [master] [origin/HEAD] [origin/master] version
2013-07-24 10:39          M─┐ Merge branch 'refs/heads/develop'
2013-07-24 10:32          │ o success message for emailaddress editing and some small code changes
2013-07-23 16:39          │ o added a changelog
2013-07-23 16:07          │ o changed to the use of results.redirect() with the new NoHttpBody of ninjaframework
2013-07-23 14:18          │ o some small code-improvements
2013-07-23 10:42          │ o Merge branch 'refs/heads/enh11-ninja15' into develop
2013-07-22 15:51          │ o small additions in the tests, now use the application.conf to get the admin-account in tests
2013-07-22 15:11          │ o some more doc and small renamings
2013-07-22 09:55          │ o collect mailtransactions and push them with the mailaddress-expiration-job to the database
2013-07-19 10:14          │ o changed to the use of noContent-template
[main] d65dd1a06059e18044e3079144657edcc70aa6b0 - commit 1 of 388 (2%)
commit d65dd1a06059e18044e3079144657edcc70aa6b0
Refs: [Version1.1.3], master, <origin/HEAD>, <origin/master>
Author:
AuthorDate: Wed Jul 24 10:39:57 2013 +0200
Commit:
CommitDate: Wed Jul 24 10:39:57 2013 +0200

    version
---
 pom.xml                          | 2 +-
 xcmailr-jetty-starter/pom.xml    | 2 +-
 xcmailr-webapp/pom.xml           | 2 +-
 3 files changed, 3 insertions(+), 3 deletions(-)

diff --git a/pom.xml b/pom.xml
index a84bc69..aada8b5 100644
--- a/pom.xml
+++ b/pom.xml
@@ -22,7 +22,7 @@
        <groupId>org.xcmailr</groupId>
        <packaging>pom</packaging>
        <name>XCMailr</name>
-       <version>1.1.3-SNAPSHOT</version>
+       <version>1.1.3</version>
[diff] d65dd1a06059e18044e3079144657edcc70aa6b0 - line 1 of 53 (45%)
```

Tools - gitk



```
Revert "Merge branch 'B1'"                      Rene Schwietzke <[redacted]>   2013-09-11 22:07:08
Merge branch 'B1'                               Rene Schwietzke <[redacted]>   2013-09-11 22:04:03
added b1                                        Rene Schwietzke <[redacted]>   2013-09-11 22:03:08
master.txt                                      Rene Schwietzke <[redacted]>   2013-09-11 22:03:38
Revert "Merge branch 'test1'"                   Rene Schwietzke <[redacted]>   2013-09-11 22:02:01
Revert "Merge branch 'movetest'"                Rene Schwietzke <[redacted]>   2013-09-11 21:26:13
Revert "Revert "Merge branch 'movetest'""       Rene Schwietzke <[redacted]>   2013-09-11 21:23:21
Revert "Merge branch 'movetest'"                Rene Schwietzke <[redacted]>   2013-09-11 21:22:19
Merge branch 'movetest'                         Rene Schwietzke <[redacted]>   2013-09-11 17:48:49
Renaming foobar                                 Rene Schwietzke <[redacted]>   2013-09-11 17:47:37
Merge branch 'test1'                            Rene Schwietzke <[redacted]>   2013-09-11 17:03:28
ReneRene                                        Rene Schwietzke <[redacted]>   2013-09-11 16:57:28
Commit 2                                        Rene Schwietzke <[redacted]>   2013-09-11 17:02:30
Commit 1                                        Rene Schwietzke <[redacted]>   2013-09-11 17:02:22
Mehr                                            Rene Schwietzke <[redacted]>   2013-09-11 16:57:53
Foobar is in town                               Rene Schwietzke <[redacted]>   2013-09-11 16:39:13
Schreibfehler korrigiert                        Rene Schwietzke <[redacted]>   2013-09-11 16:40:20
Mehr Text                                       Rene Schwietzke <[redacted]>   2013-09-11 16:30:37
Initial commit training                         Rene Schwietzke <[redacted]>   2013-09-11 16:27:02
```

SHA1 ID: af9c5bd215de6909fcf098dcacc99b22f5f1dd32 Row 32 / 46

Find next prev commit containing: Exact All fields

Search

Diff Old version New version Lines of context: 3 Ignore space change Line diff

Patch Tree

Comments
foobar.txt

```
Author: Rene Schwietzke <[redacted]>  2013-09-11 22:02:01
Committer: Rene Schwietzke <[redacted]>  2013-09-11 22:02:01
Parent: a7bc7ed3c3896c2ed9196c14ba1425d41db1d11a (Revert "Merge branch 'movetest'")
Child:  81ff0658d46b8b3a487761a7af2bd59d53471d77 (added b1)
Child:  ab9a820f7194a15e0c0090f7e3a2282a6114078d (master.txt)
Branches: master, remotes/origin/master
Follows:
Precedes:

    Revert "Merge branch 'test1'"

    This reverts commit c53a976f2bf63462c8a60a616115032e898694e3, reversing
    changes made to b966f60c610c7c7520c0e8be87418f059b268872.

---------------------------------- foobar.txt ----------------------------------
index 4920c85..f31b11f 100644
@@ -1,5 +1,3 @@
-asjd hfkjsahkjfdasdhfkj hsadf
+asdhfkj hsadf
  asdfh asdf asd foi
 adshfkjhdsa
-aslkdhadkf
```

A list of tools

- GitEye
- SmartGit
- TortoiseGit
- EGit for Eclipse

git blame

Q & A

Your moment to ask questions, to show off, to blame us for wasted life time, blame git, or blame Linus.

git resources

- Atlassian's Git Pages
  - https://www.atlassian.com/git
- The official web page
  - http://git-scm.com/
- Stuff about Git
  - http://gitready.com/

Sources used

- http://git-scm.com/
- http://gitready.com/
- https://www.atlassian.com/git
- Subversion DB: http://commons.wikimedia.org/wiki/File:Database-vcs-subversion.svg (CC-BY-SA)
- Creative Commons icons: http://creativecommons.org/about/downloads (CC-BY)

License

- Use these slides freely
- Share them freely
- Modify them freely but share them again