git reset, checkout

and just Git, in general

main source:
https://git-scm.com/book/en/v2/Git-Tools-Reset-Demystified

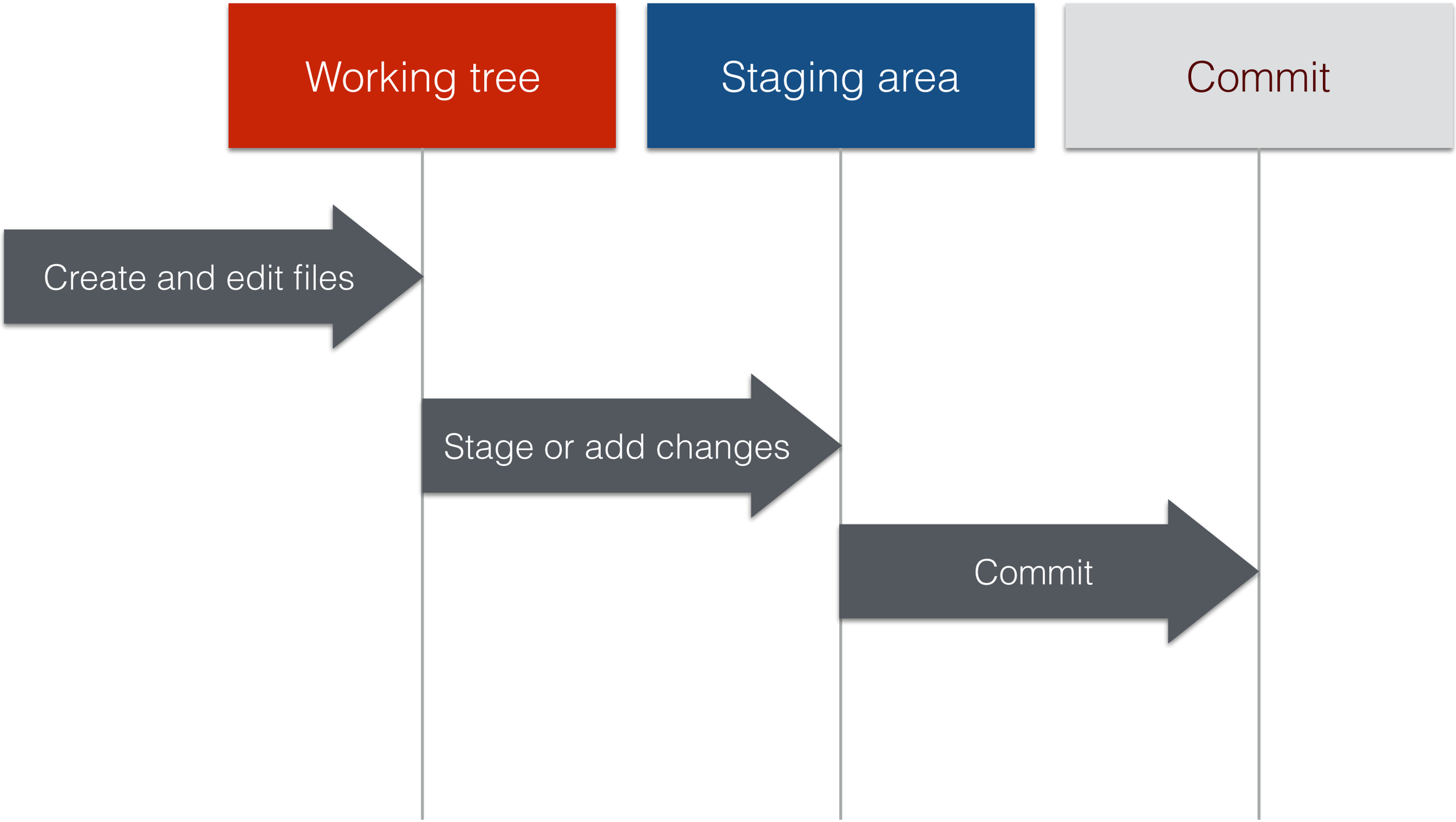
	HEAD	Index	Workdir	WD Safe?
Commit Level				
`reset --soft [commit]`	REF	NO	NO	YES
`reset [commit]`	REF	YES	NO	YES
`reset --hard [commit]`	REF	YES	YES	**NO**
`checkout [commit]`	HEAD	YES	YES	YES
File Level				
`reset (commit) [file]`	NO	YES	NO	YES
`checkout (commit) [file]`	NO	YES	YES	**NO**

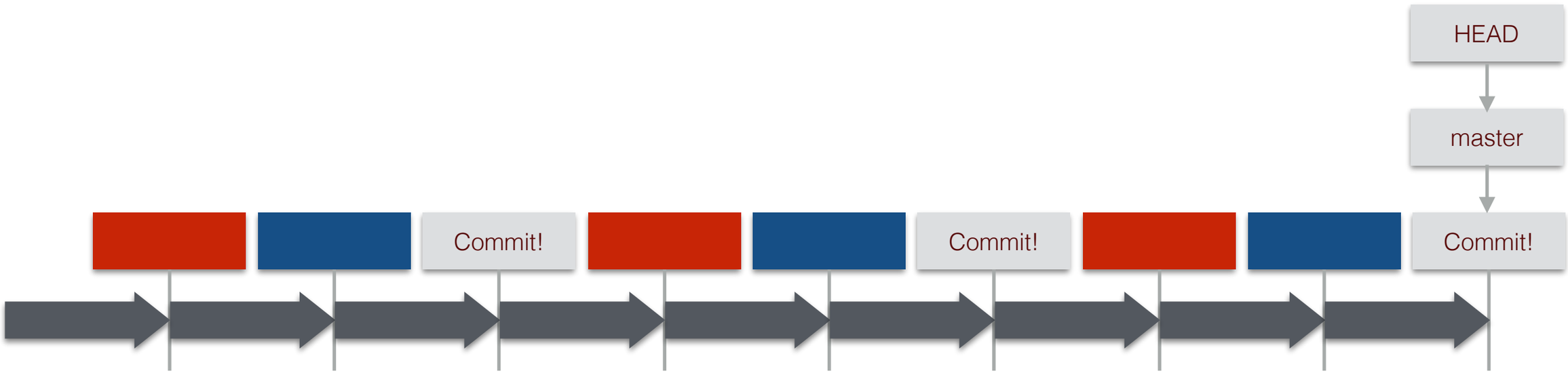
Two branches diverged on Git, and I—
I pulled the one less traveled by,
And now there are several merge conflicts.

Two branches diverged on Git, and I—
I pulled the one less traveled by,
And now there are several merge conflicts.

https://twitter.com/marktimemedia/status/469462640314421248

http://mynameismichelle.com/git-frost/


Working tree
Staging area
Commit
Create and edit files
Stage or add changes
Commit



do stuff
stage it
commit it

lather, rinse, repeat

Working directory **Working tree**	“Sandbox”*, current state of filesystem
Index **Staging area**	Proposed next commit snapshot
HEAD	Last commit snapshot, parent of next commit

* IMO my local files feel VERY REAL and not like a “sandbox” at all! YMMV

“do stuff”

Working tree

Two branches diverged on Git, and I—

Staging area

Commit

“stage it”

Working tree

Two branches diverged on Git, and I—

Staging area

Two branches diverged on Git, and I—

Commit

“commit it”

Working tree

Two branches diverged on Git, and I—

Staging area

Two branches diverged on Git, and I—

Commit

Two branches diverged on Git, and I—

“do (more) stuff”

Working tree

Two branches diverged on Git, and I—
I pulled the one less traveled by,

Staging area

Two branches diverged on Git, and I—

Commit

Two branches diverged on Git, and I—

“stage it”

Working tree

Two branches diverged on Git, and I—
I pulled the one less traveled by,

Staging area

Two branches diverged on Git, and I—
I pulled the one less traveled by,

Commit

Two branches diverged on Git, and I—

“commit it”

Working tree

Two branches diverged on Git, and I—
I pulled the one less traveled by,

Staging area

Two branches diverged on Git, and I—
I pulled the one less traveled by,

Commit

Two branches diverged on Git, and I—
I pulled the one less traveled by,

we're going to come back to this!

Working tree

Two branches diverged on Git, and I—
I pulled the one less traveled by,
And now there are several merge conflicts.

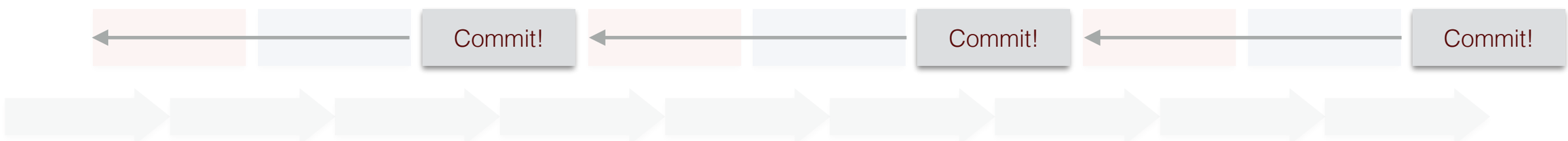
Staging area

Two branches diverged on Git, and I—
I pulled the one less traveled by,
And now there are several merge conflicts.

Commit

Two branches diverged on Git, and I—
I pulled the one less traveled by,
And now there are several merge conflicts.

git history



yes, this is confusing:

big arrows show time,
your progress on the project

little arrows show the Git graph,
point from a commit to its parent,
you know your parents but not necessarily your kids!

HEAD

↓

current branch

↓

commit

HEAD
is a pointer to
tip of current branch reference
which is a pointer to
a commit

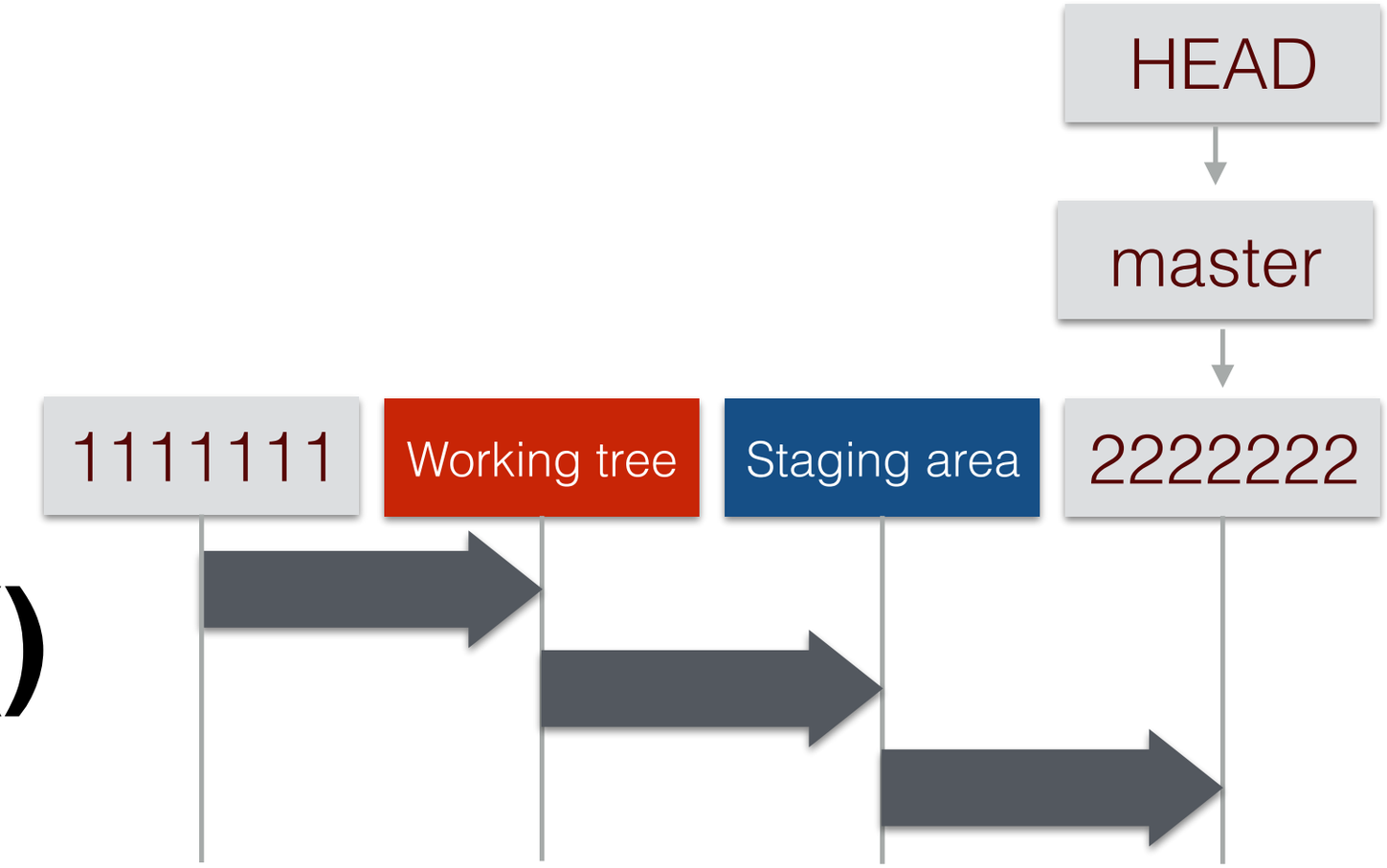
HEAD ≈ “snapshot of the last commit”

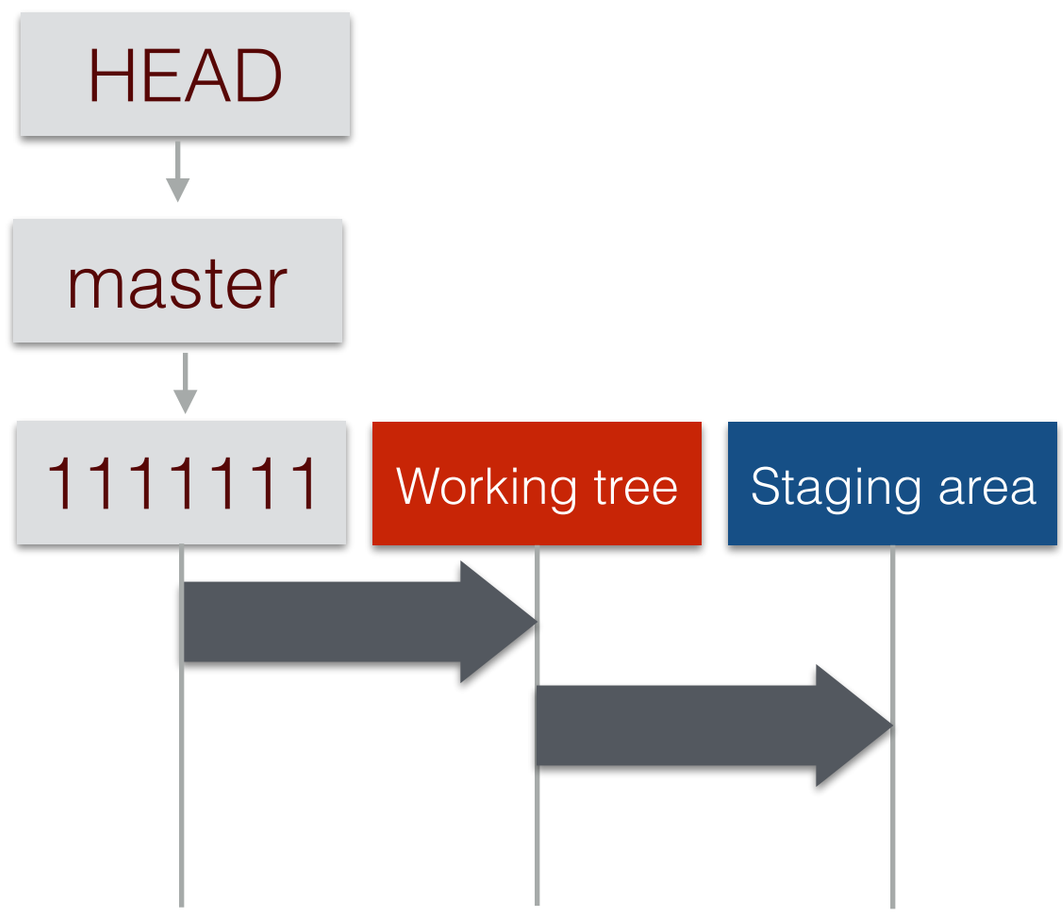
`git reset –TYPE [rev]`

	soft	mixed	hard
Move branch HEAD points at to point at [rev]	✔	✔	✔
Make staging area look like HEAD		✔	✔
Make working tree look like HEAD			✔

git_uncommit()

```
git reset –soft HEAD^
```





uncommit() before

2222222	Two branches diverged on Git, and I— I pulled the one less traveled by,
Working tree	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.
Staging area	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.
3333333	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.

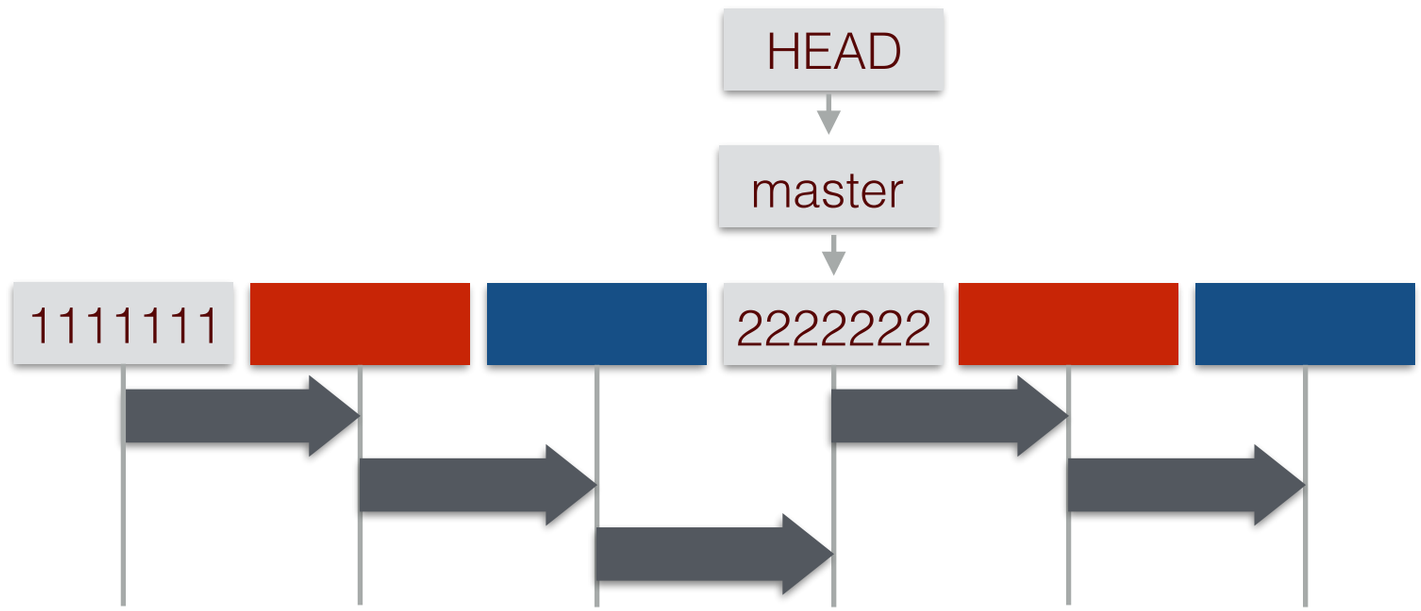
uncommit() after

2222222	Two branches diverged on Git, and I— I pulled the one less traveled by,
Working tree	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.
Staging area	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.
3333333	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.

?git_reset(...)?

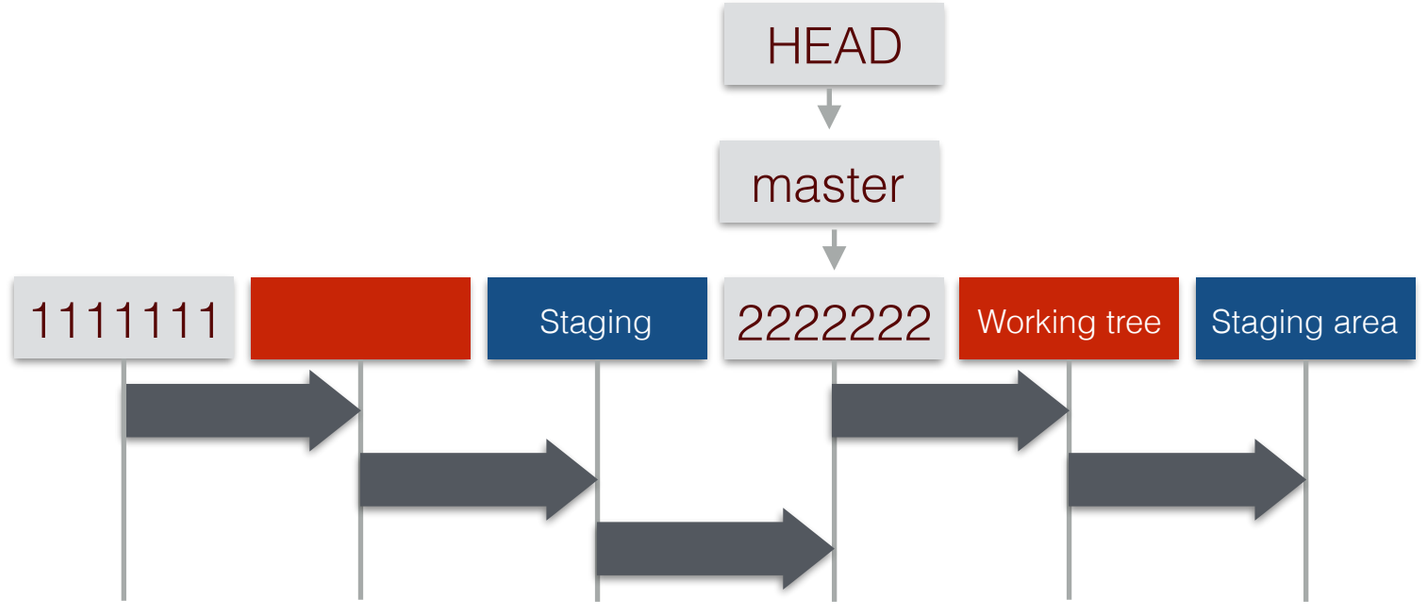
will wrap git2r::reset(<git_commit>, reset_type = "soft")

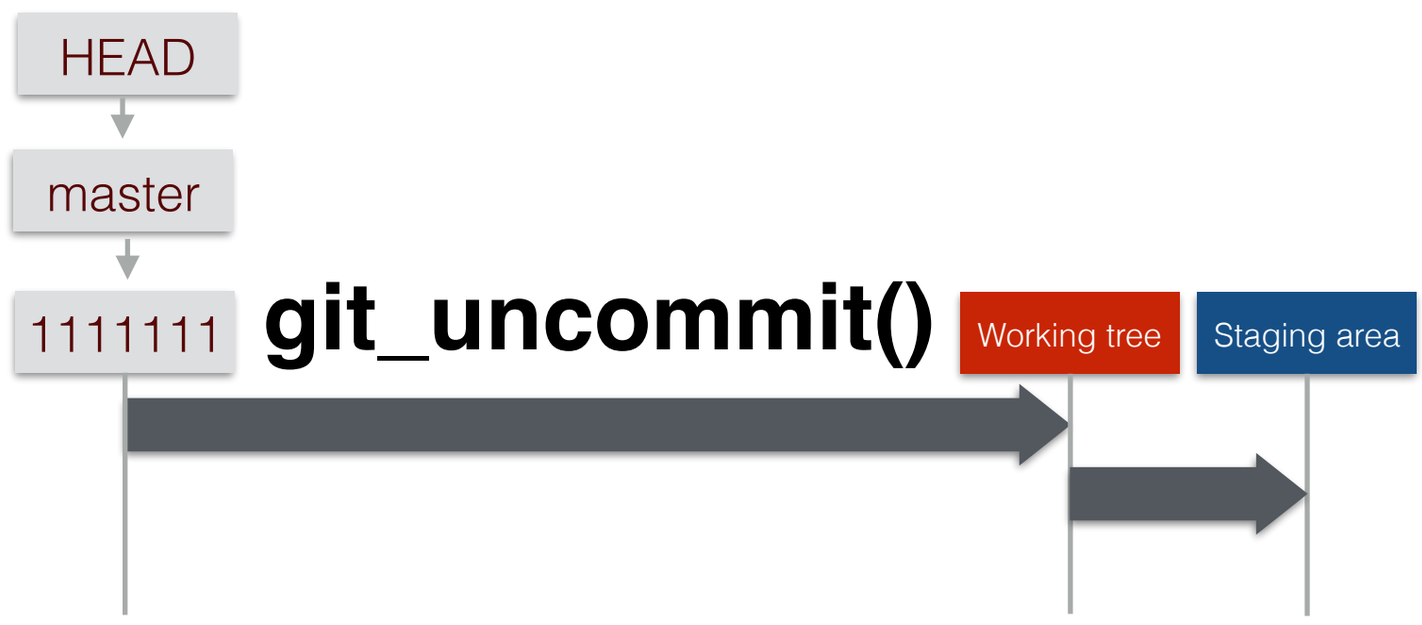
```
git reset –soft [rev]

git reset –soft HEAD^
git reset –soft 1111111
```

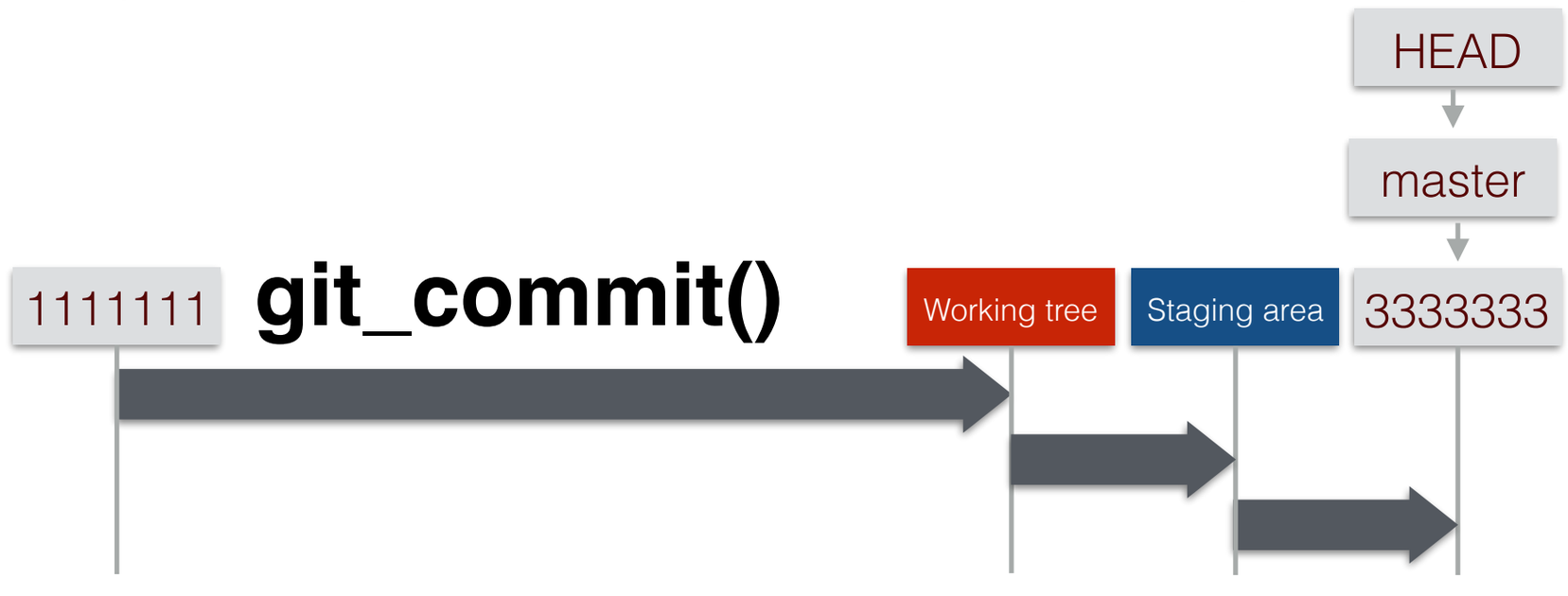


git_amend()

```
git commit –amend

git reset –soft HEAD^
git commit
```







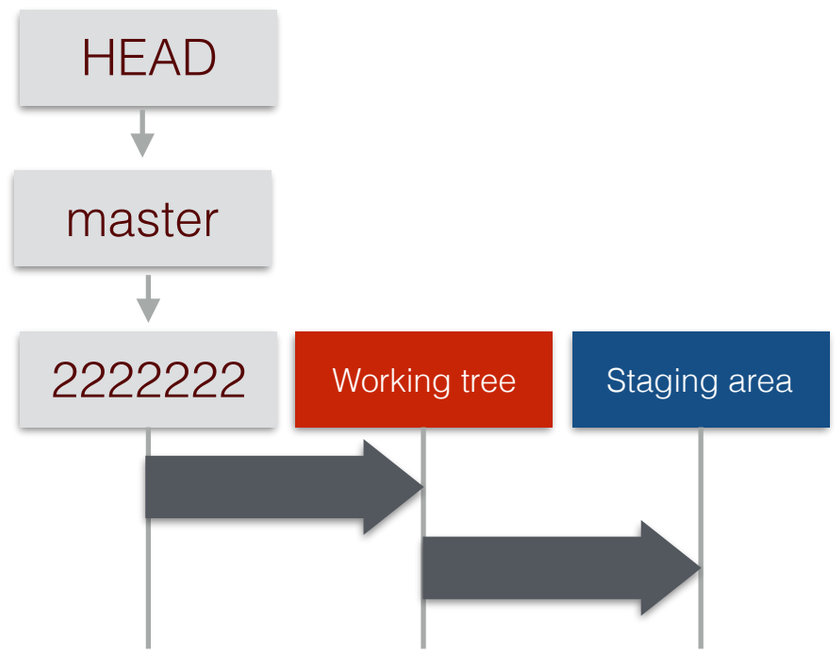
git_unstage(…)

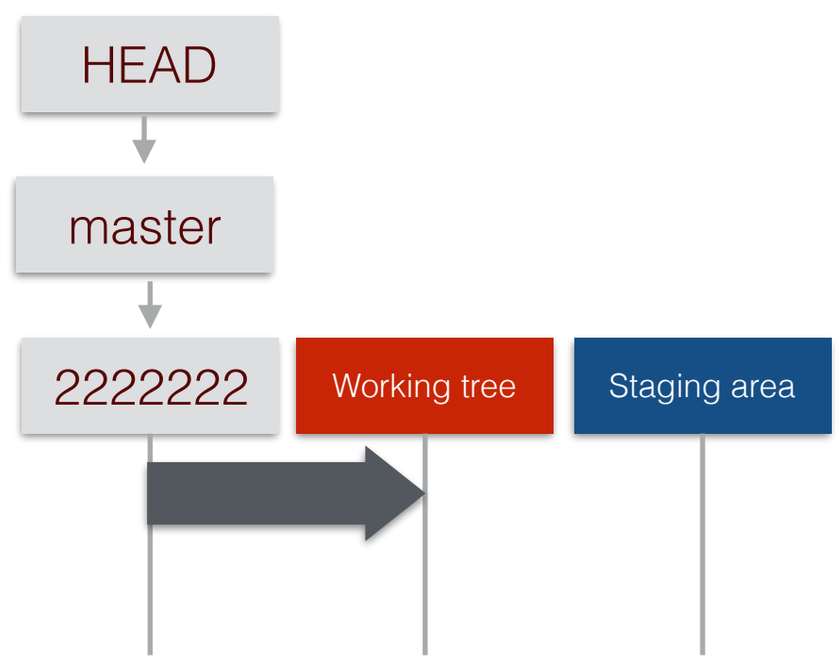
wraps git2r::reset(<git_repository>, path), which is hard-wired to mixed reset to HEAD

```
git reset
git reset —mixed HEAD
```

undoes

```
git add
```





unstage() before

2222222	Two branches diverged on Git, and I— I pulled the one less traveled by,
Working tree	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.
Staging area	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.
3333333	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.

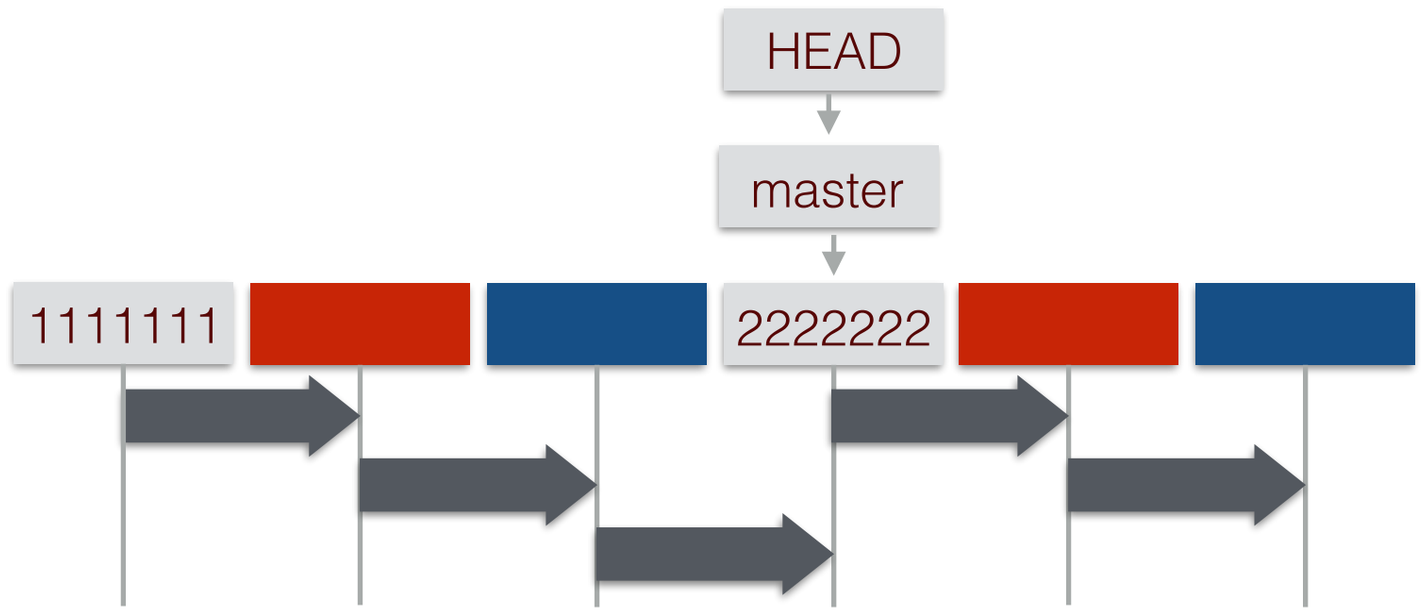
unstage() after

2222222	Two branches diverged on Git, and I— I pulled the one less traveled by,
Working tree	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.
Staging area	Two branches diverged on Git, and I— I pulled the one less traveled by, ← !!
3333333	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.

?git_reset(…)?

will wrap git2r::reset(<git_commit>, reset_type = "mixed")

```
git reset [rev]
git reset –mixed [rev]

git reset –mixed HEAD^
git reset –mixed 1111111
```



why do soft resets feel like they do more than mixed?
the names imply soft << mixed, in terms of effect!

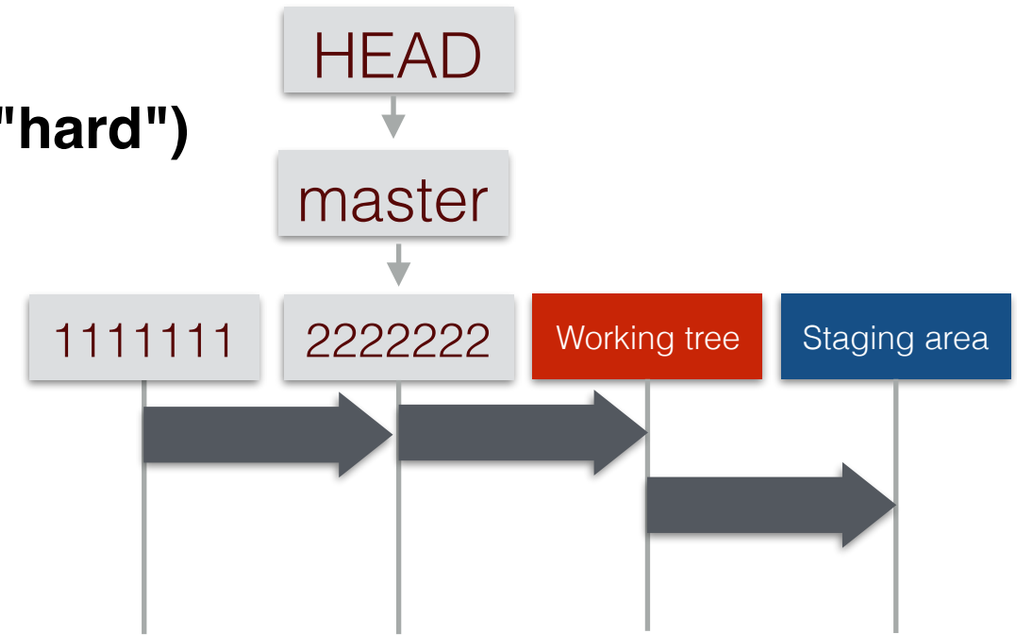
because you usually soft reset to **HEAD^** (uncommit)
but mixed reset to **HEAD** (unstage)

soft reset to HEAD is a no-op, btw

?git_reset(...)?

will wrap git2r::reset(<git_commit>, reset_type = "hard")

```
git reset –hard [rev]
```




```
git reset –hard
```


```
git reset –hard HEAD^
git reset –hard 1111111
```

this is the only reset that is not “working directory safe”
none of them is “history safe”

hard reset before

2222222	Two branches diverged on Git, and I— I pulled the one less traveled by,
Working tree	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.
Staging area	Two branches diverged on Git, and I— I pulled the one less traveled by,
3333333	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.

hard reset after

2222222	Two branches diverged on Git, and I— I pulled the one less traveled by,
Working tree	Two branches diverged on Git, and I— I pulled the one less traveled by, ← !!
Staging area	Two branches diverged on Git, and I— I pulled the one less traveled by,
3333333	Two branches diverged on Git, and I— I pulled the one less traveled by, And now there are several merge conflicts.

`git reset –TYPE [rev]`

	soft	mixed	hard
Move branch HEAD points at to point at [rev]	✔	✔	✔
Make staging area look like HEAD		✔	✔
Make working tree look like HEAD			✔

`git reset [rev] — file.txt`

you don’t have to reset the whole repo
you can reset specific paths

file-specific resets are special, though:
obviously don’t move the branch that HEAD is pointing to

soft file reset? no such thing, would be no-op

mixed file reset? yes, it’s the only kind that exists
copies version of file from [rev] to the staging area

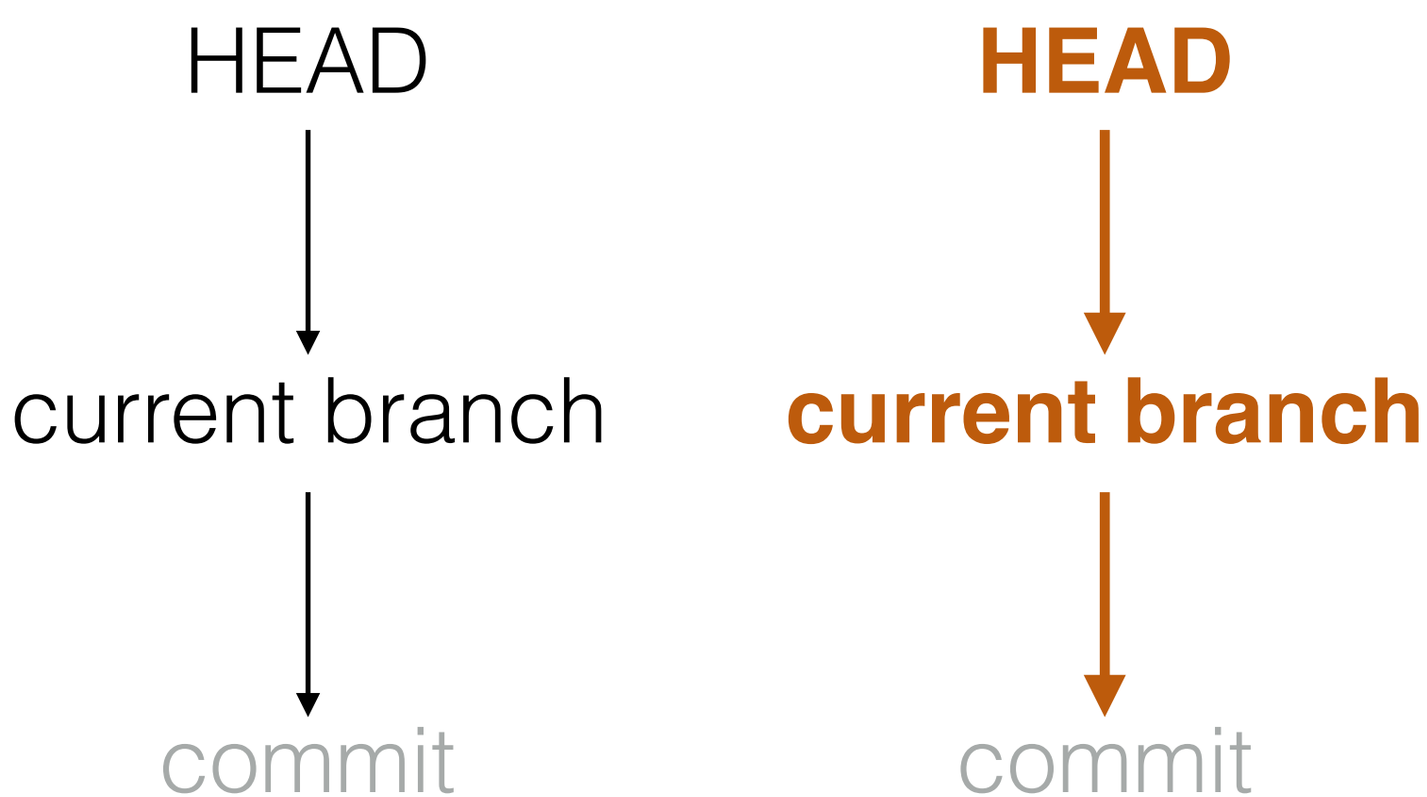
hard file reset? no such thing, but morally it’s same as checkout

```
git reset [rev]
```

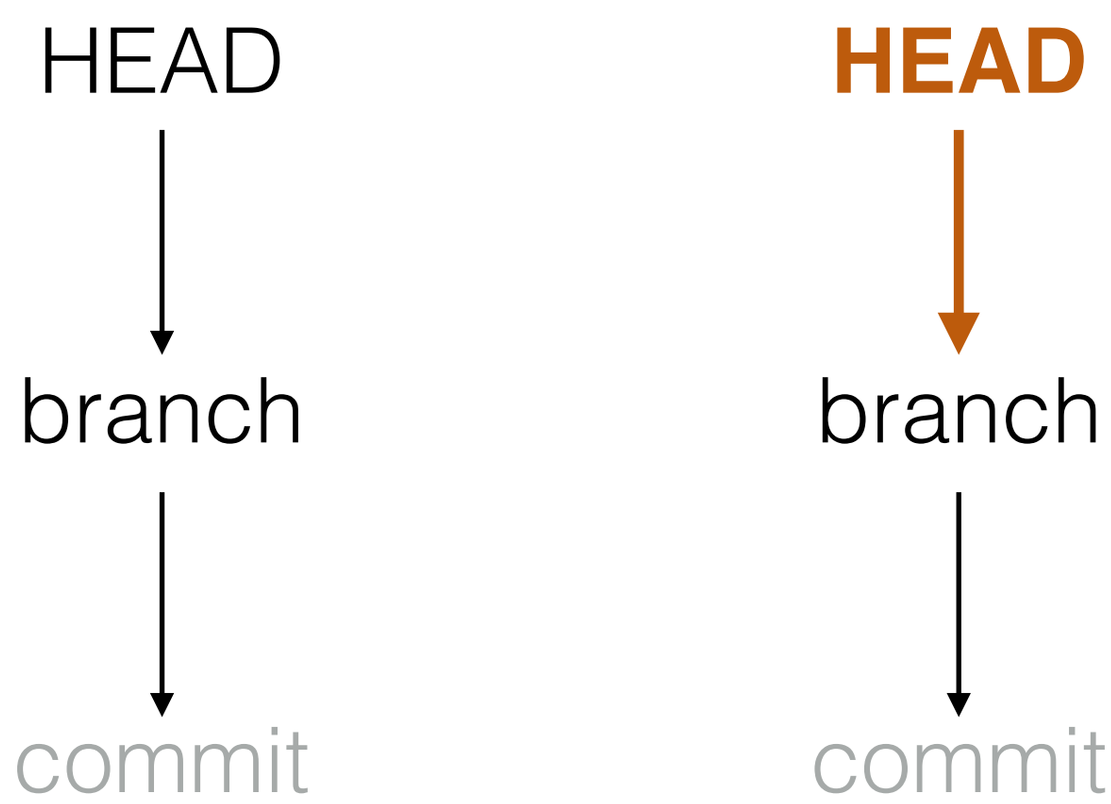
vs.

```
git checkout [rev]
```

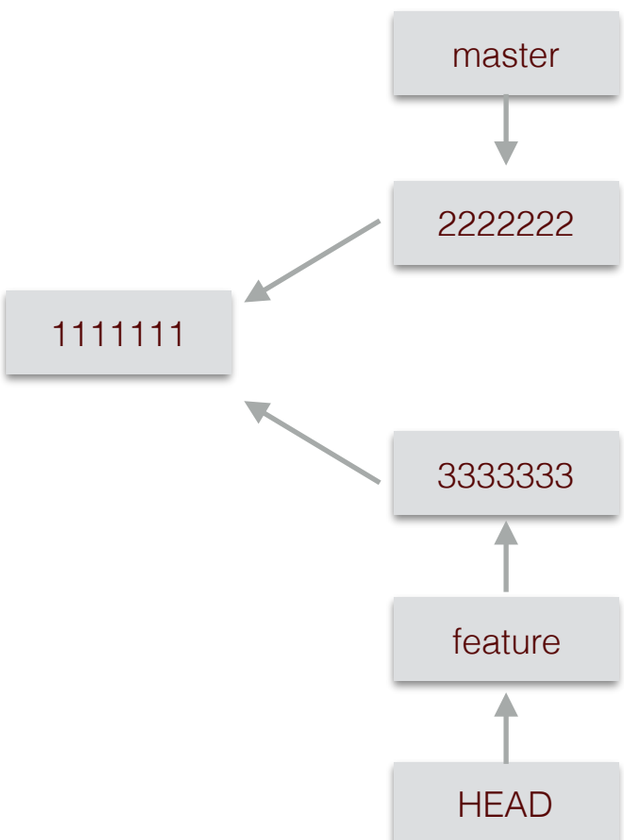
`git reset` moves these pointers as a unit



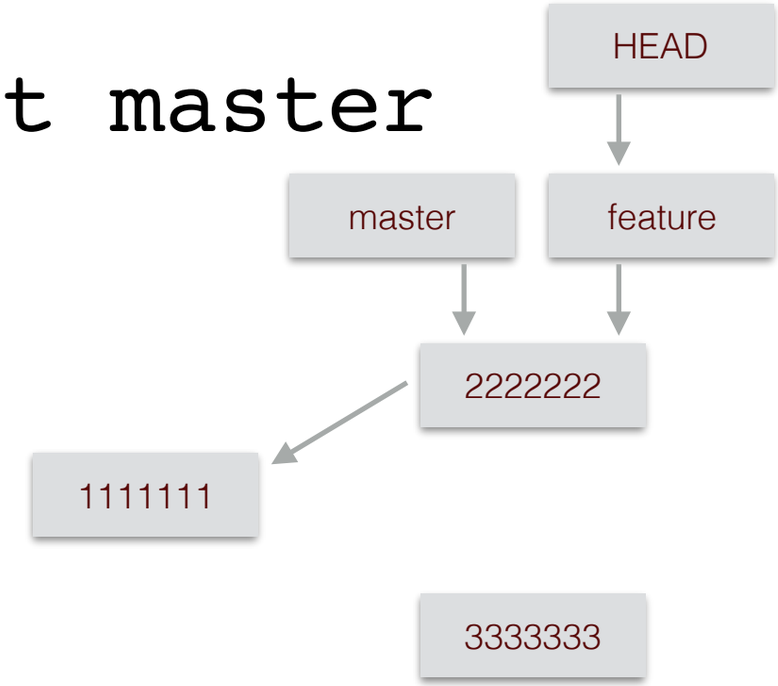
`git checkout` only moves HEAD



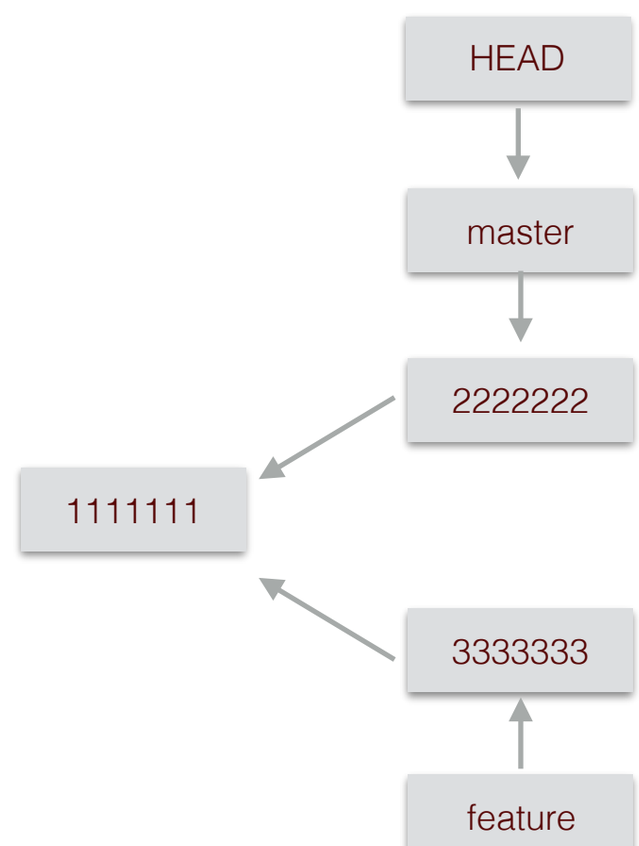
start here,
on feature …



git reset master



git checkout master



w/r/t “working directory safety”,
what is the difference between a reset and checkout?

`git reset –hard` will destroy data immediately

to do similar damage with checkout, I highly recommend
`git checkout –force`

this trivial merge idea still not captured

First, unlike `reset --hard`, `checkout` is working-directory safe; it will check to make sure it’s not blowing away files that have changes to them. Actually, it’s a bit smarter than that – it tries to do a trivial merge in the Working Directory, so all of the files you *haven’t* changed in will be updated. `reset --hard`, on the other hand, will simply replace everything across the board without checking.

`git reset [rev] file.txt`
makes file.txt look like its version in [rev] in the staging area
“working directory safe”

`git checkout [rev] file.txt`
makes file.txt look like its version in [rev] in the **working tree**
NOT “working directory safe”
it will clobber even w/o explicit `—force`

`git reset vs. git checkout`

1. reset moves branch (and HEAD), checkout moves HEAD

2. both can destroy data but you need different flags 😬

3. neither is consistently “working directory safe” so mind your paths and flags