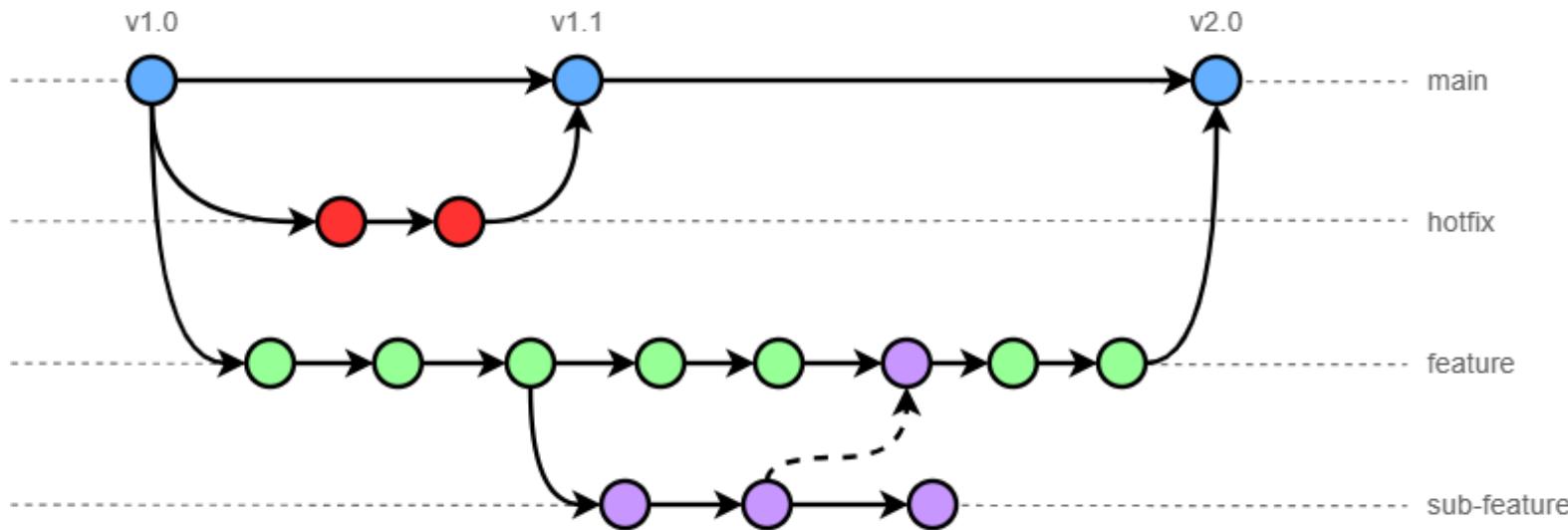


GITTING MORE OUT OF GIT



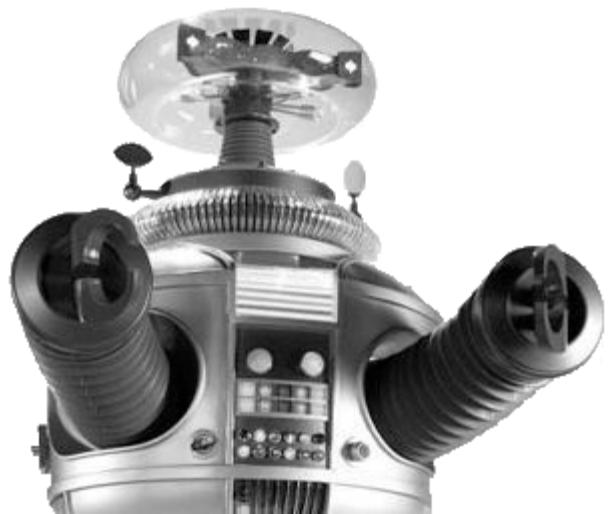
jordankasper.com/git

Session Feedback:





Jordan Kasper | [@jakerella](https://twitter.com/jakerella)



DANGER!

GIT IS DISTRIBUTED

A CENTRALIZED VCS

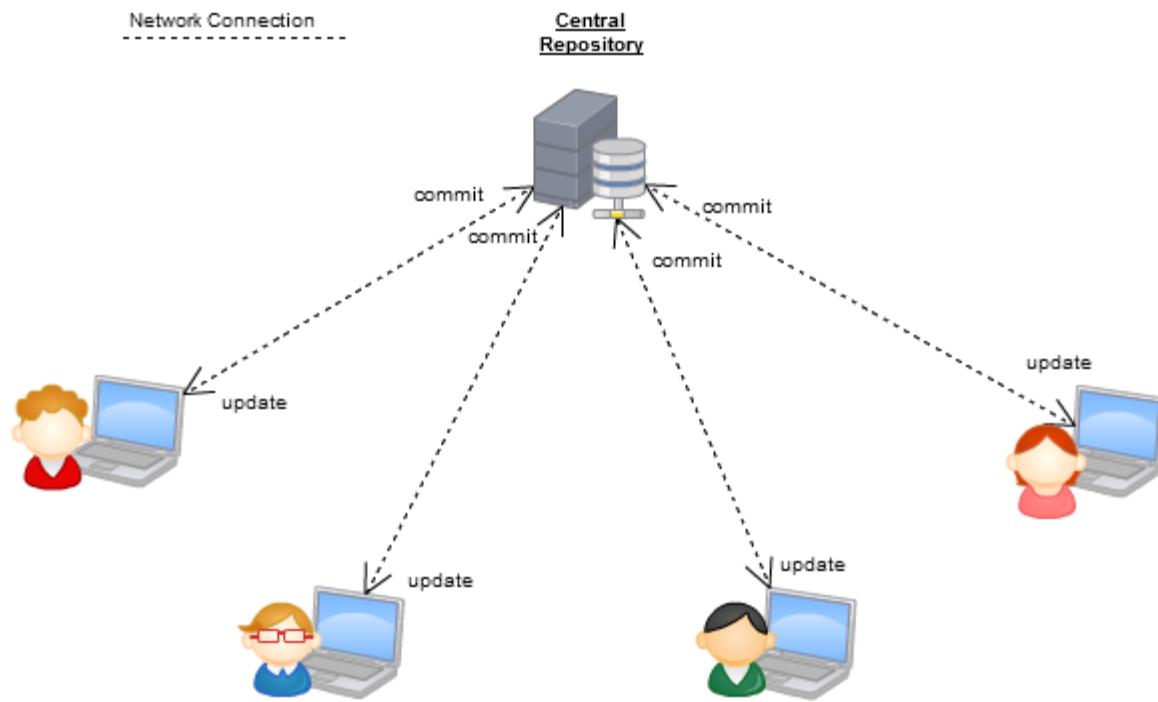
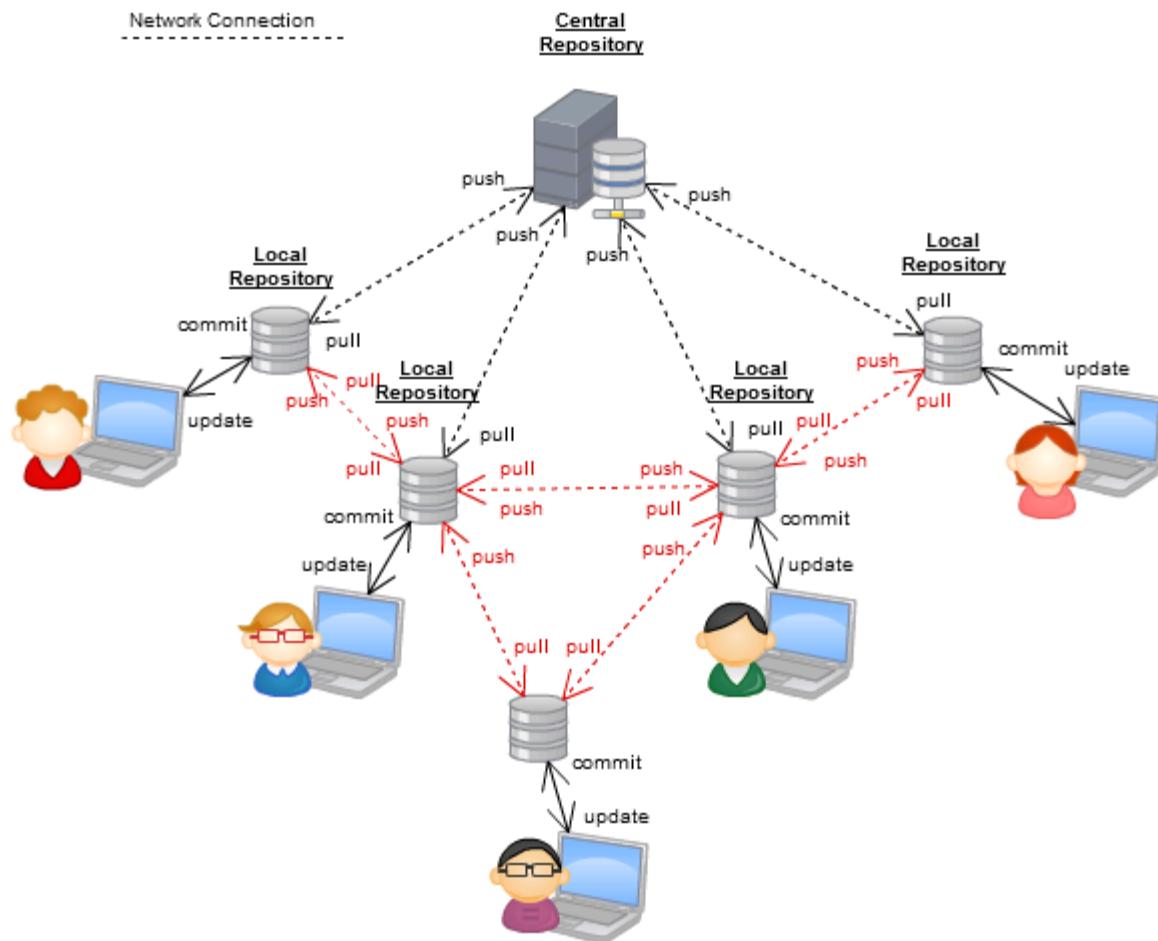


Image credit: <https://docs.joomla.org/Dvcs>

A DISTRIBUTUED VCS



A DISTRIBUTUED VCS (IN PRACTICE)

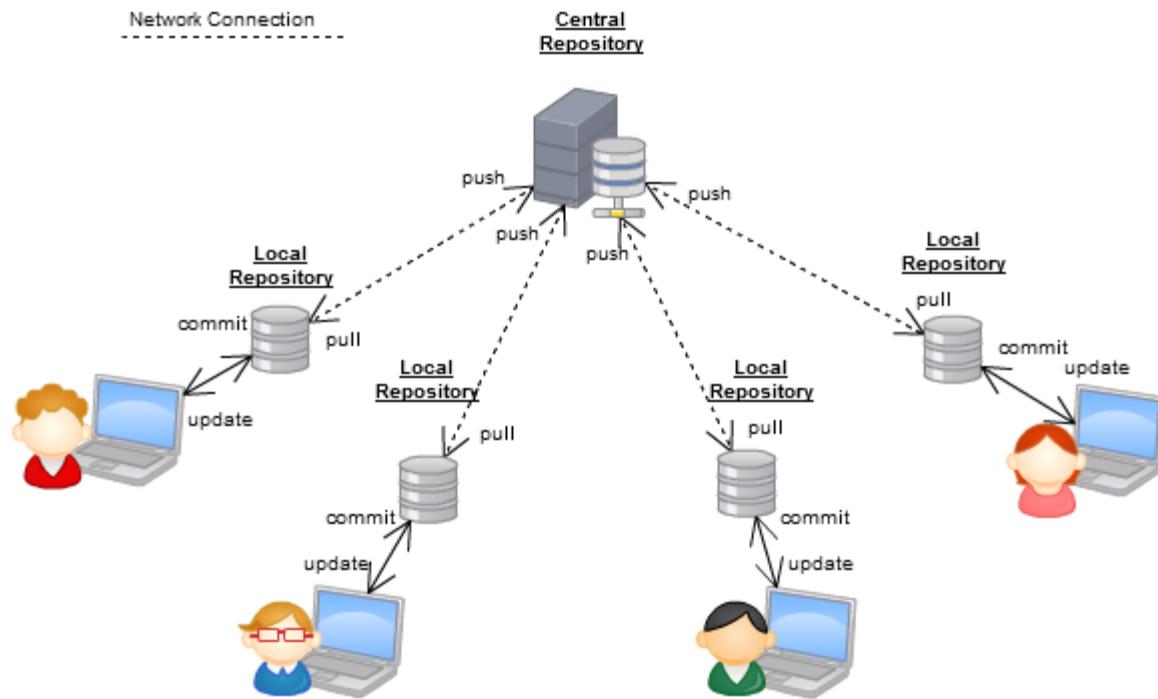


Image credit: <https://docs.joomla.org/Dvcs>

REMOTES

A "remote" is just a repo outside your local environment.

CLONING

```
~$ git clone git@github.com:jakerella/foo.git repo  
~$ cd repo  
~/repo$ git branch -a  
* main  
  remotes/origin/main
```

Cloning creates a remote called "origin",
which appears as a local branch.

TRACKING

Any local branch can "track" a remote URL:

```
~/repo$ git branch -vv
* main           b956c45c [origin/main] Initial commit
  some-local-branch a74b2950 Implemented that cool feature
```

Note the double "vv" here. A single "v" will not show you the remote branch being tracked.

WHEREZITAT?

```
~/repo$ git remote -v  
origin  git@github.com:jakerella/foo.git (fetch)  
origin  git@github.com:jakerella/foo.git (push)
```

Note we can have different URLs for fetching and pushing!

GOING UPSTREAM

How do I make a new branch "track" that remote repo?

```
~/repo$ git checkout -b new-feature
~/repo$ git branch --set-upstream-to=origin/new-feature
~/repo$ git push -u origin new-feature
~/repo$ git branch -vv
  main              b956c45c [origin/main] Initial commit
* new-feature      b956c45c [origin/new-feature] Initial commit
  some-local-branch a74b2950 Implemented that cool feature
```

CAN THERE BE OTHER REMOTES?

Assume I've **forked** the Nodejs main repo...

```
~$ git clone git@github.com:jakerella/node.git  
~/node$ git remote -v  
origin  git@github.com:jakerella/node.git (fetch)  
origin  git@github.com:jakerella/node.git (push)  
~/node$ git remote add node-source git@github.com:nodejs/node.git  
~/node$ git remote -v  
origin          git@github.com:jakerella/node.git (fetch)  
origin          git@github.com:jakerella/node.git (push)  
node-source     git@github.com:nodejs/node.git (fetch)  
node-source     git@github.com:nodejs/node.git (push)
```

OPEN SOURCE FLOW

How do I get the latest changes from Nodejs?

```
~/node$ git fetch node-source
remote: Enumerating objects: 6018, done.
...
From github.com/nodejs/node
 * [new branch] ...
```

```
~/node$ git checkout main
```

```
~/node$ git merge node-source/main
Updating 40f7a8ea..a479419b
...
~/node$ git checkout new-feature
```

```
~/node$ git merge main
```

BRANCH DIFFERENCES

BRANCH DIFFERENCES

We commit some changes to our local feature branch.

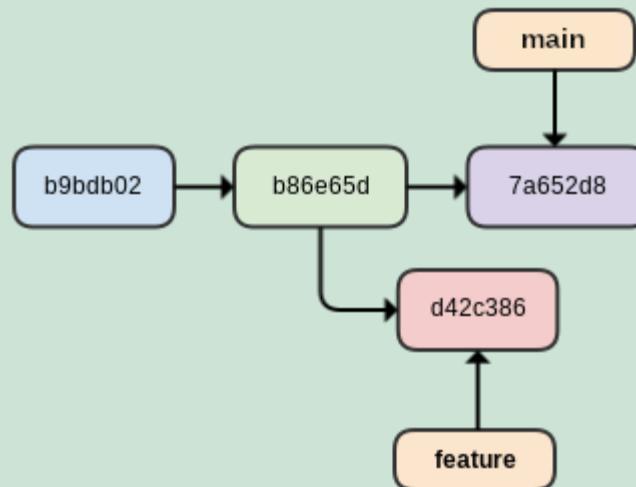
```
~/node$ git diff main new-feature

diff --git a/README.md b/README.md
index a479419b..b5a9776f 100644
--- a/README.md
+++ b/README.md
@@ -1 +0,0 @@
-This is a bad line, remove it!

diff --git a/feature.js b/feature.js
new file mode 100644
index 000000..b5a9776f
...
```

BRANCH DIFFERENCES

What if that README change was from another merge?



```
~$ git diff main new-feature
```

This is ALL changes between `main` and `new-feature`,
(regardless of what branch those came from).

SINGLE-BRANCH DIFFERENCES (..)

```
~$ git diff main..new-feature  
  
diff --git a/feature.js b/feature.js  
new file mode 100644  
index 0000000..b5a9776f  
--- /dev/null  
+++ b/feature.js  
...
```

`..`: show all commits on "new-feature" not on "main"

`space`: show all commit differences between "new-feature" and "main"

`...`: same as `space`

FIXING CHANGES

UNSTAGED CHANGES

```
~/repo$ echo "testing" >> README.md
```

```
~/repo$ git status
```

```
On branch new-feature
```

```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git restore <file>..." to discard changes in working directory)
```

```
modified: README.md
```

UNSTAGED CHANGES

```
~/repo$ git restore README.md
```

```
~/repo$ git status  
On branch new-feature  
nothing to commit, working tree clean
```

You can also restore entire directories:

```
~/repo$ git restore source/routes/
```

UNSTAGED CHANGES

Careful, **NEW** files will not be removed!

```
~/repo$ echo "testing" > source/routes/new-file.js
```

```
~/repo$ git restore source/routes/
```

```
~/repo$ git status
```

```
On branch new-feature
```

Untracked files:

```
(use "git add <file>..." to include in what will be committed)
```

```
source/routes/new-file.js
```

(You can just delete the file if not needed.)

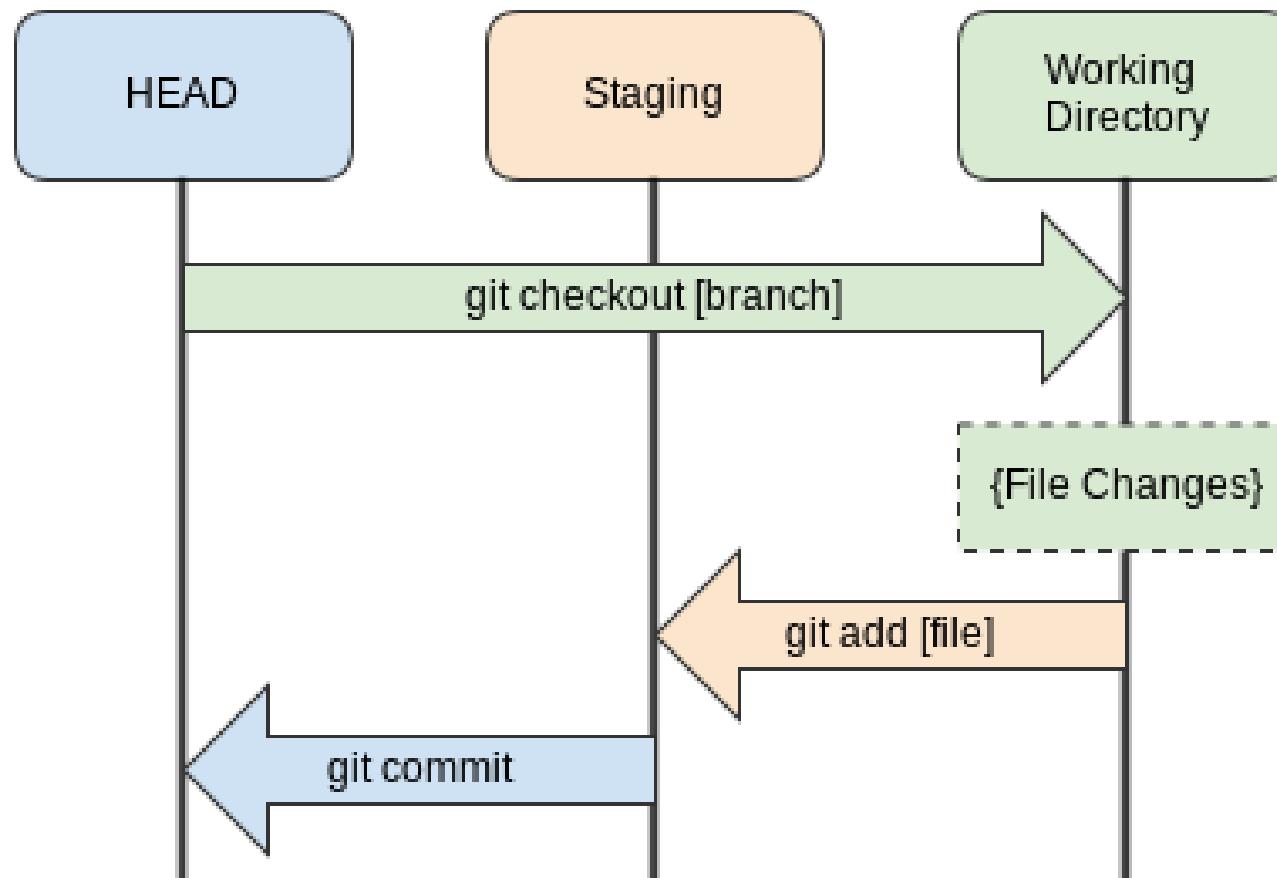
STAGED CHANGES

```
~/repo$ echo "testing" >> README.md
~/repo$ git add README.md

~/repo$ git status
On branch new-feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.md

~/repo$ git restore --staged README.md
~/repo$ git status
On branch new-feature
Changes not staged for commit:
  modified:   README.md
```

REMEMBER THE THREE STATES!



FIXING THINGS AFTER COMMIT

FIXING COMMIT MESSAGES

```
~/repo$ git commit -m "This is teh best"  
[feature-branch 54af593b] This is teh best  
1 file changed, 1 insertion(+)
```

```
~/repo$ git commit --amend -m "This is the best"  
[feature-branch 0486a7d6] This is the best  
1 file changed, 1 insertion(+)
```

FIXING COMMIT MESSAGES

```
~/repo$ git commit -m "This is teh best"  
[feature-branch 54af593b] This is teh best  
1 file changed, 1 insertion(+)
```

```
~/repo$ git commit --amend -m "This is the best"  
[feature-branch 0486a7d6] This is the best  
1 file changed, 1 insertion(+)
```

WHAT ABOUT THE GIT LOG?

```
~/repo$ git log  
commit 0486a7d61ee9bcaa31d7eb062c0bcafee3e530f0  
Author: jdoe <john@doe.com>  
Date: Thu Jan 29 11:45:14 2026 -0400
```

This is the best

```
commit 72267fc26d88fa977d24760252da63b46ca3b81a  
Author: fbar <foo@bar.com>  
Date: Wed Jan 28 09:45:10 2026 -0400
```

Bug fixes for older features

...

WHAT IF I FORGOT A FILE?

Stage it and amend it.

```
~/repo$ git add forgotten.js  
~/repo$ git commit --amend  
[new-feature 3e81873b] This is the best  
Date: Thu Jan 29 11:49:54 2026 -0400  
2 files changed, 1 insertion(+), 1 deletion(-)  
create mode 100644 forgotten.js
```

CAUTION WHEN AMENDING!

If you **push** your bad change,
then amend it,
you might cause a nasty conflict for someone else.

GONE, BUT NOT FORGOTTEN

```
~/repo$ git log --oneline
3e81873b (HEAD -> new-feature) This is the best
72267fc2 (origin/main, origin/HEAD, main) Bug fixes for older features
```

```
~/repo$ git reflog
3e81873b HEAD@{0}: commit (amend): This is the best
0486a7d6 HEAD@{1}: commit (amend): This is the best
54af593b HEAD@{2}: commit: This is teh best.
72267fc2 HEAD@{3}: commit: Bug fixes for older features
...
```

The `reflog` is never pushed to a remote.

WHAT IF I NEED TO AMEND AN OLDER COMMIT?

You'll need to use `git rebase`

INTERACTIVE REBASING

```
~/repo$ git rebase --interactive HEAD^^^
```

```
pick 7e10e41c some old commit
pick 72267fc2 Bug fixes for older features
pick 3e81873b This is the best
```

```
# Rebase 5a39902..3e81873b onto 5a39902 (3 command(s))
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# d, drop <commit> = remove commit
...
```

INTERACTIVE REBASING

```
~/repo$ git rebase --interactive HEAD^^^  
  
pick 7e10e41c some old commit  
edit 72267fc2 Bug fixes for older features  
pick 3e81873b This is the best  
  
# Rebase 5a39902..3e81873b onto 5a39902 (3 command(s))  
# Commands:  
# p, pick <commit> = use commit  
# r, reword <commit> = use commit, but edit the commit message  
# e, edit <commit> = use commit, but stop for amending  
# s, squash <commit> = use commit, but meld into previous commit  
# d, drop <commit> = remove commit  
...
```

INTERACTIVE REBASING

```
~/repo$ git rebase --interactive HEAD^^^  
Stopped at 72267fc2... Bug fixes for older features  
You can amend the commit now, with  
  git commit --amend  
Once you are satisfied with your changes, run  
  git rebase --continue
```

Now amend the commit just like before...

```
~/repo$ git add another-file.js  
~/repo$ git commit --amend  
~/repo$ git rebase --continue  
Successfully rebased and updated refs/heads/new-feature.
```

INTERACTIVE REBASING

Careful! You changed **all history** from that amend forward.

```
~/repo$ git reflog
7650aafe HEAD@{0}: rebase -i (finish): returning to refs/heads/new-f
7650aafe HEAD@{1}: rebase -i (pick): This is the best
bcb51f9a HEAD@{2}: commit (amend): Bug fixes for older features
72267fc2 HEAD@{3}: rebase -i: fast-forward
7e10e41c HEAD@{4}: rebase -i (start): checkout HEAD^^^
3e81873b HEAD@{5}: commit (amend): This is the best
0486a7d6 HEAD@{6}: commit (amend): This is the best
54af593b HEAD@{7}: commit This is teh best
72267fc2 HEAD@{8}: commit: Bug fixes for older features
7e10e41c HEAD@{9}: commit: some old commit
```

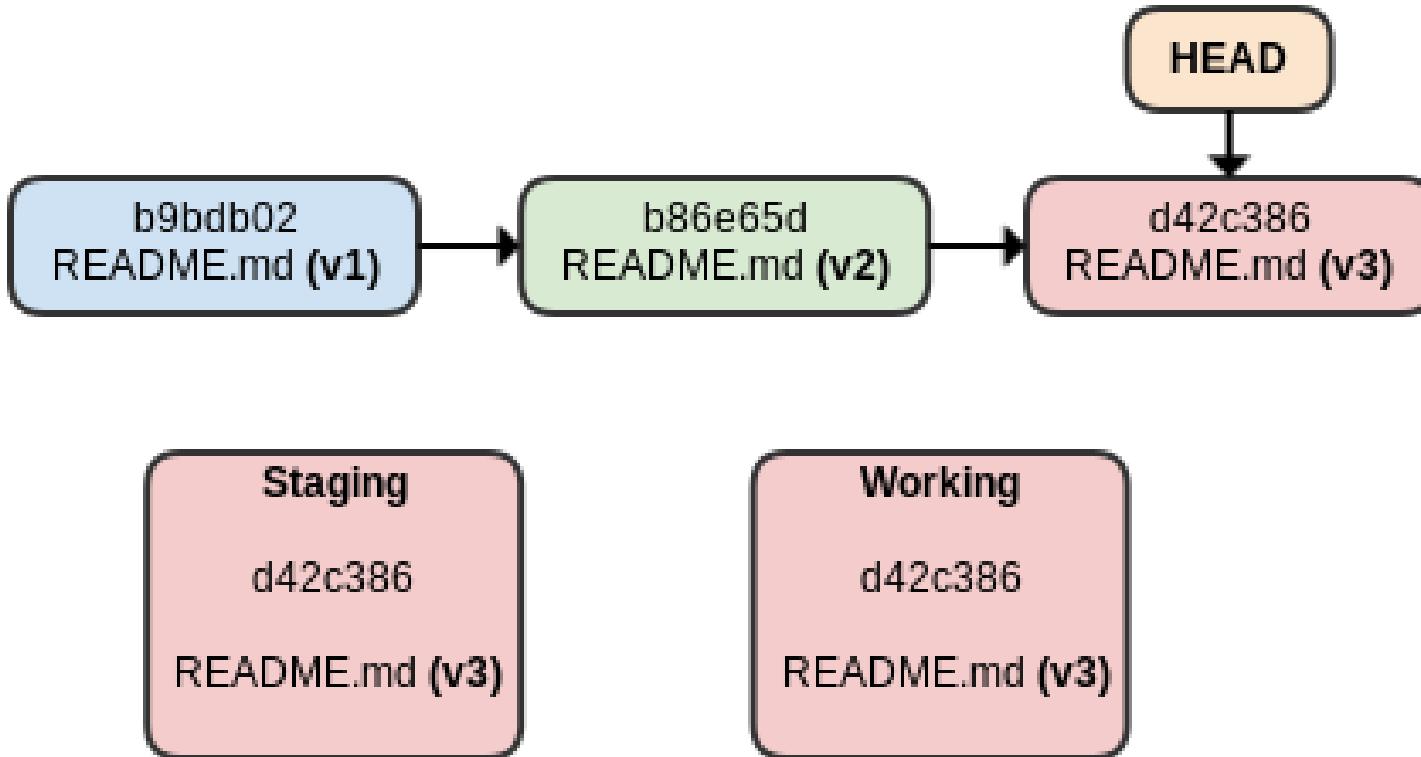
UNDERSTANDING THE THREE SEASHELLS STATES

Or... how to use `reset` for fun and profit.

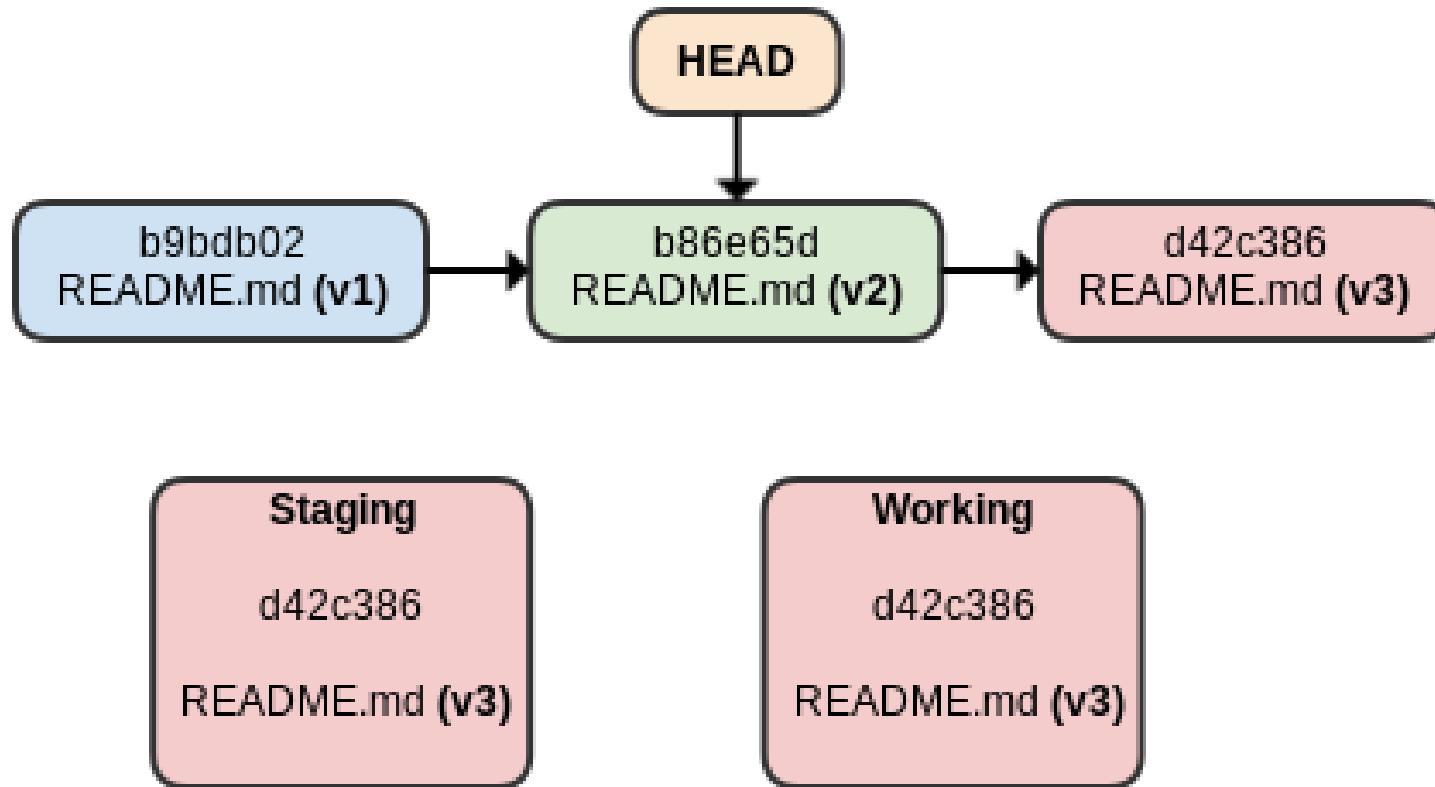
FYI, `reset` is the OG `restore`.

Here's a very helpful StackOverflow answer on the differences.

UNDERSTANDING THE THREE STATES



UNDERSTANDING THE THREE STATES



`git reset --soft HEAD^`

`HEAD` IS JUST A POINTER

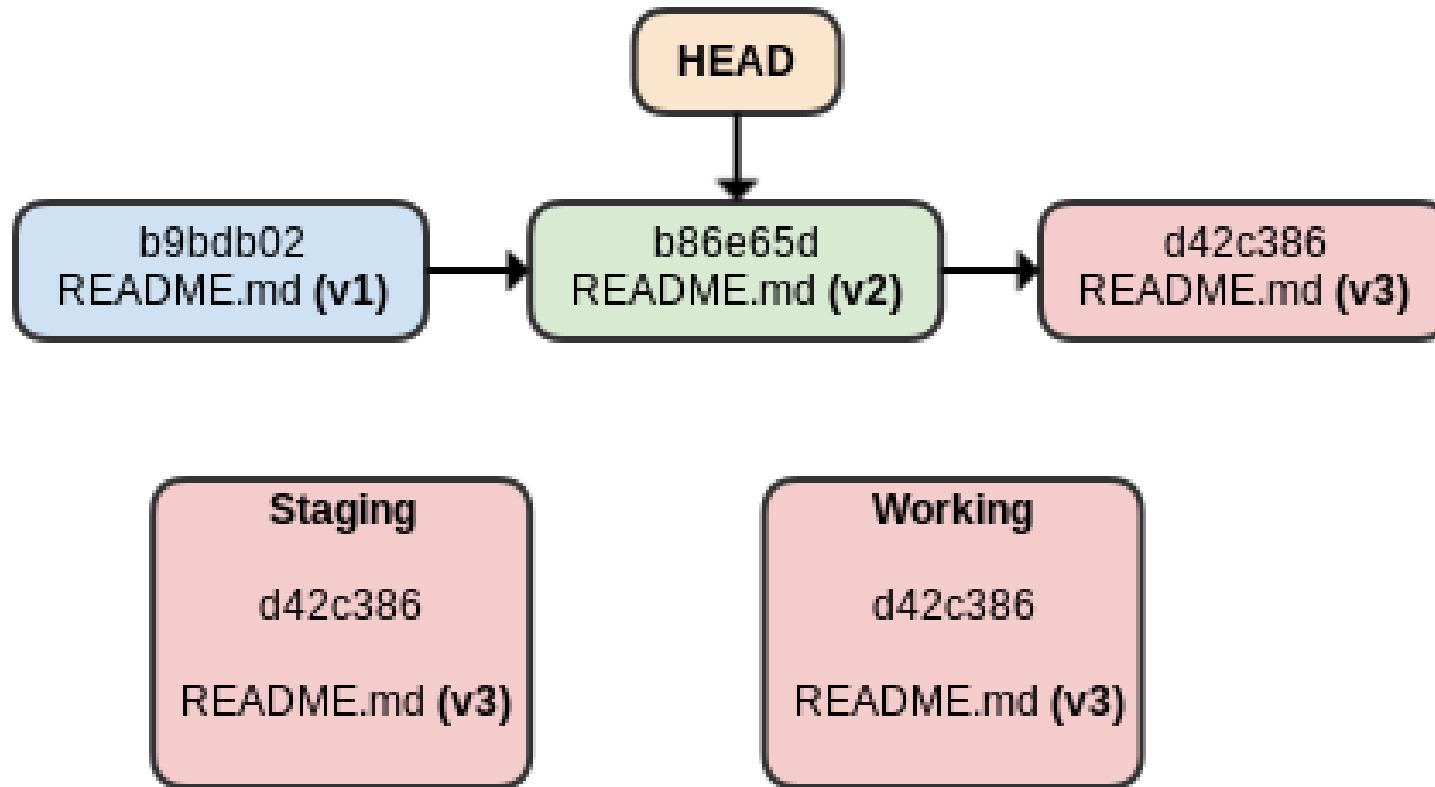
In fact, all branch names are just pointers!

And so are tags!

SOFT RESET

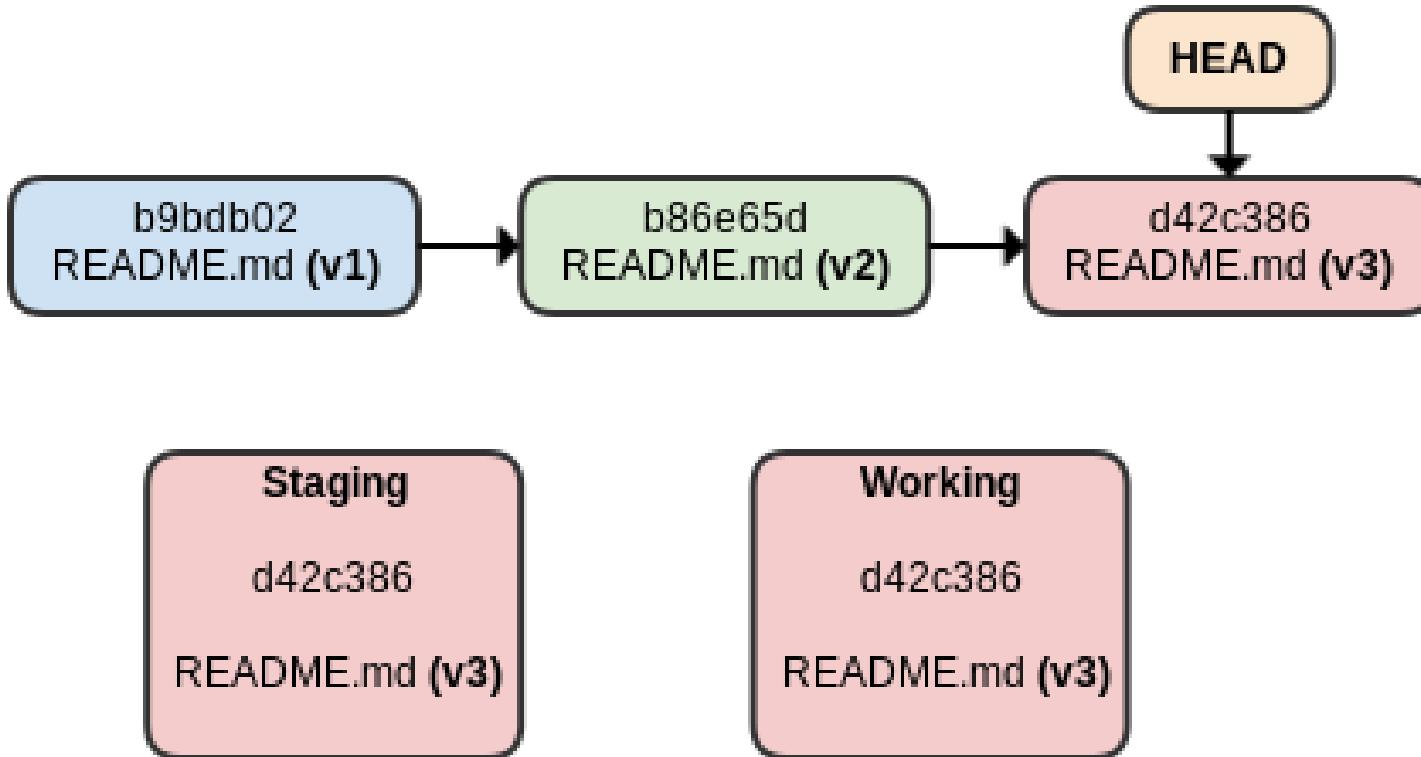
```
~/repo$ git reset --soft HEAD^  
~/repo$ git status  
On branch new-feature  
Changes to be committed:  
(use "git restore --staged <file>..." to unstage)  
modified: README.md
```

UNDERSTANDING THE THREE STATES

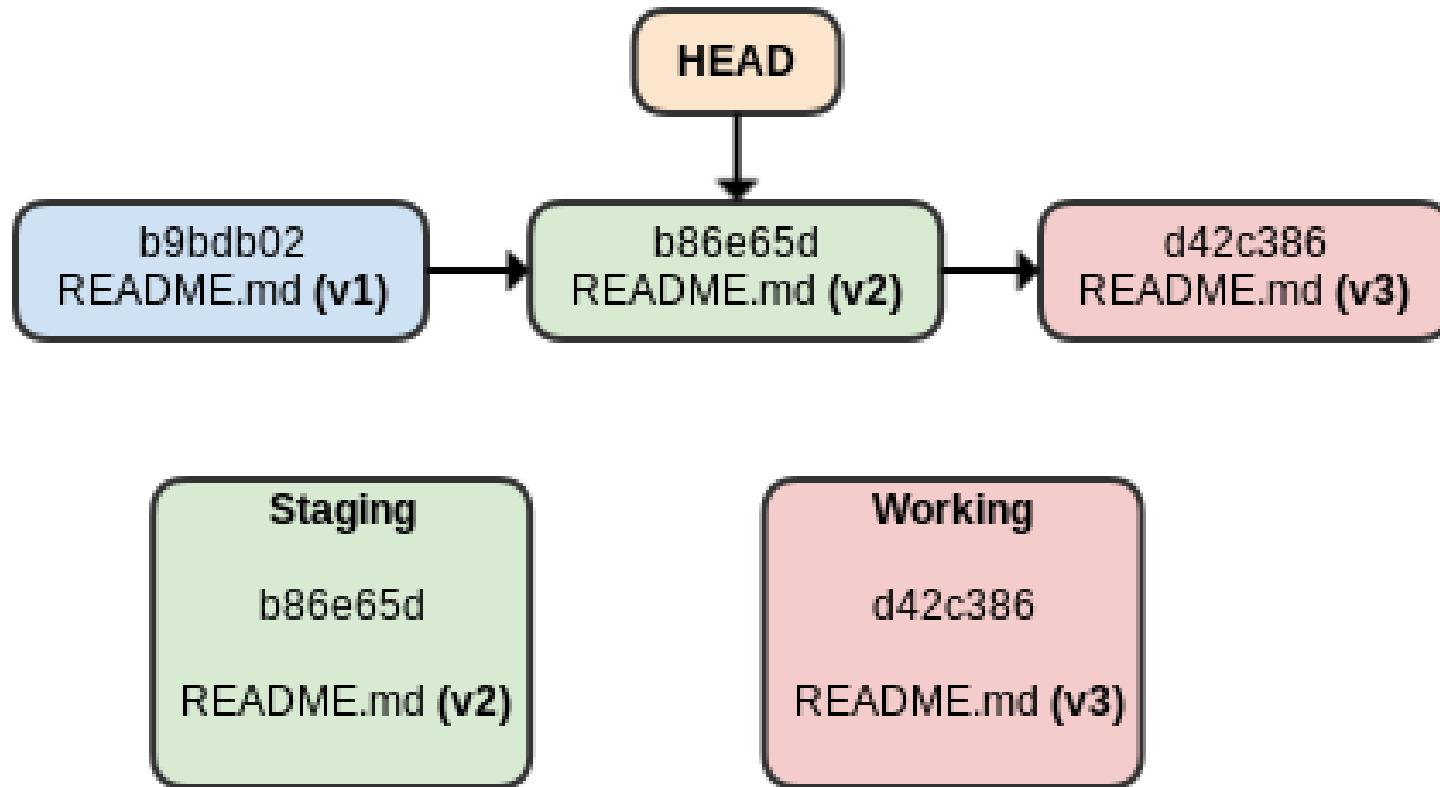


`git reset --soft HEAD^`

UNDERSTANDING THE THREE STATES



UNDERSTANDING THE THREE STATES



`git reset --mixed HEAD^`

MIXED RESET

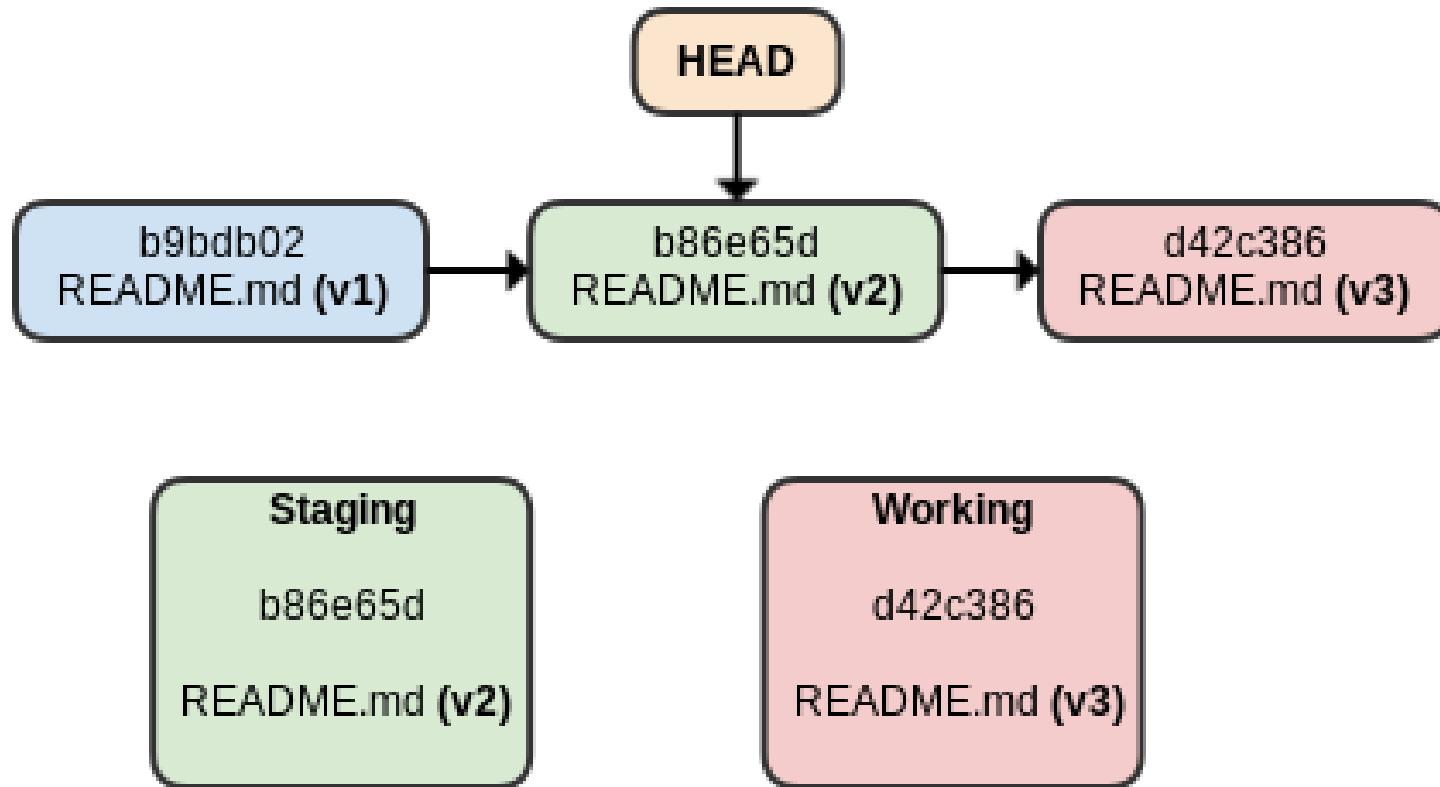
```
~/repo$ git reset --mixed HEAD^
```

```
~/repo$ git status  
On branch new-feature
```

Changes not staged for commit:

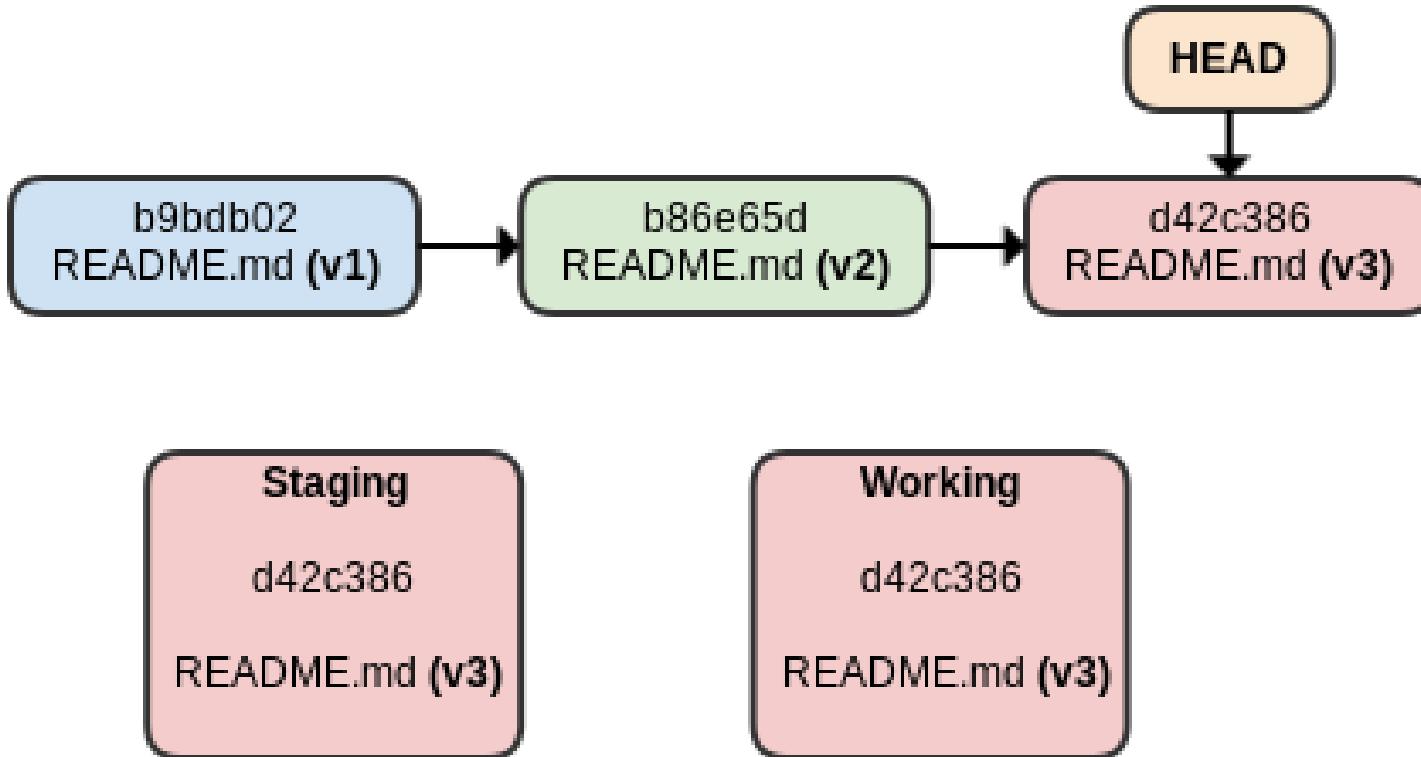
```
(use "git add <file>..." to update what will be committed)  
(use "git restore <file>..." to discard changes in working directory)  
modified: README.md
```

UNDERSTANDING THE THREE STATES

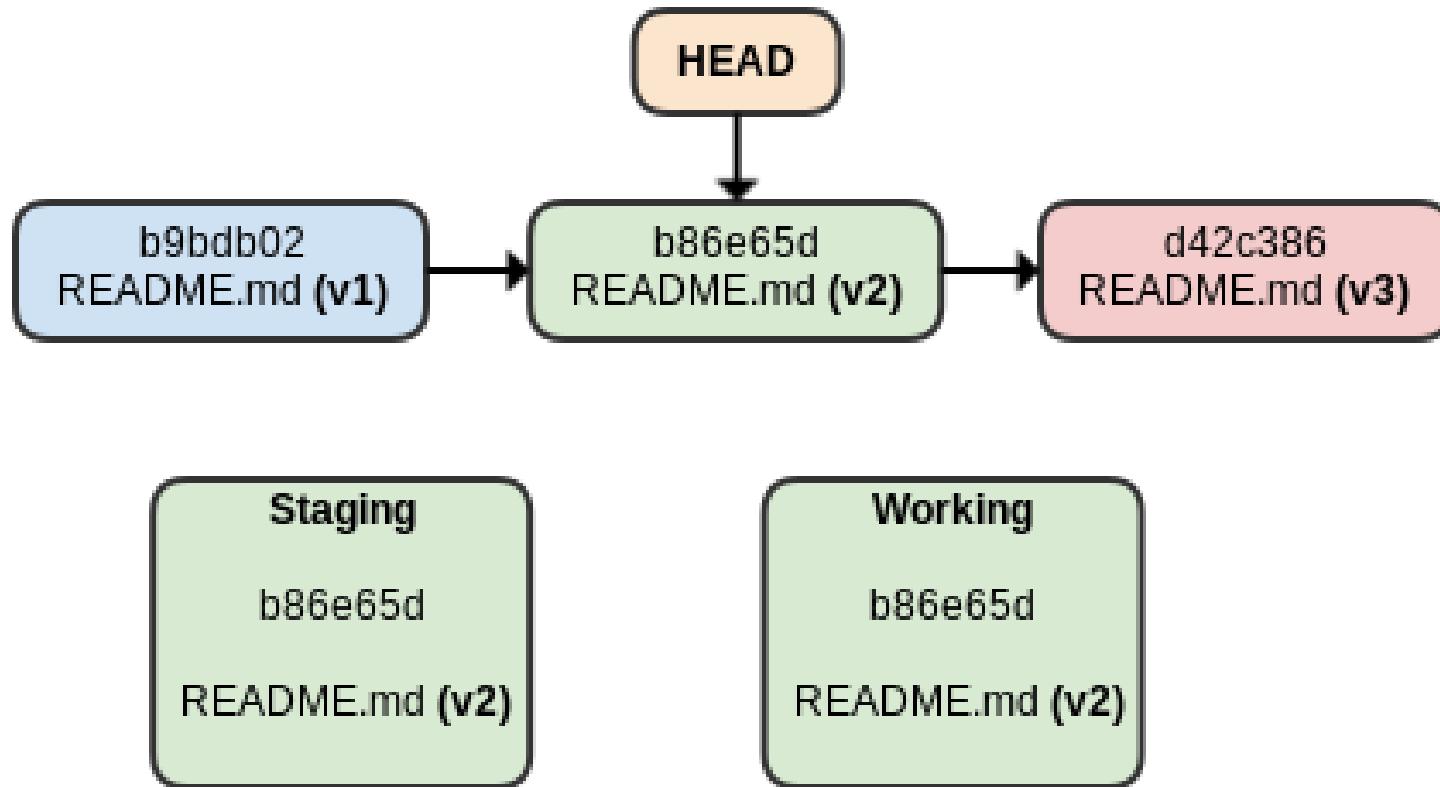


`git reset --mixed HEAD^`

UNDERSTANDING THE THREE STATES



UNDERSTANDING THE THREE STATES

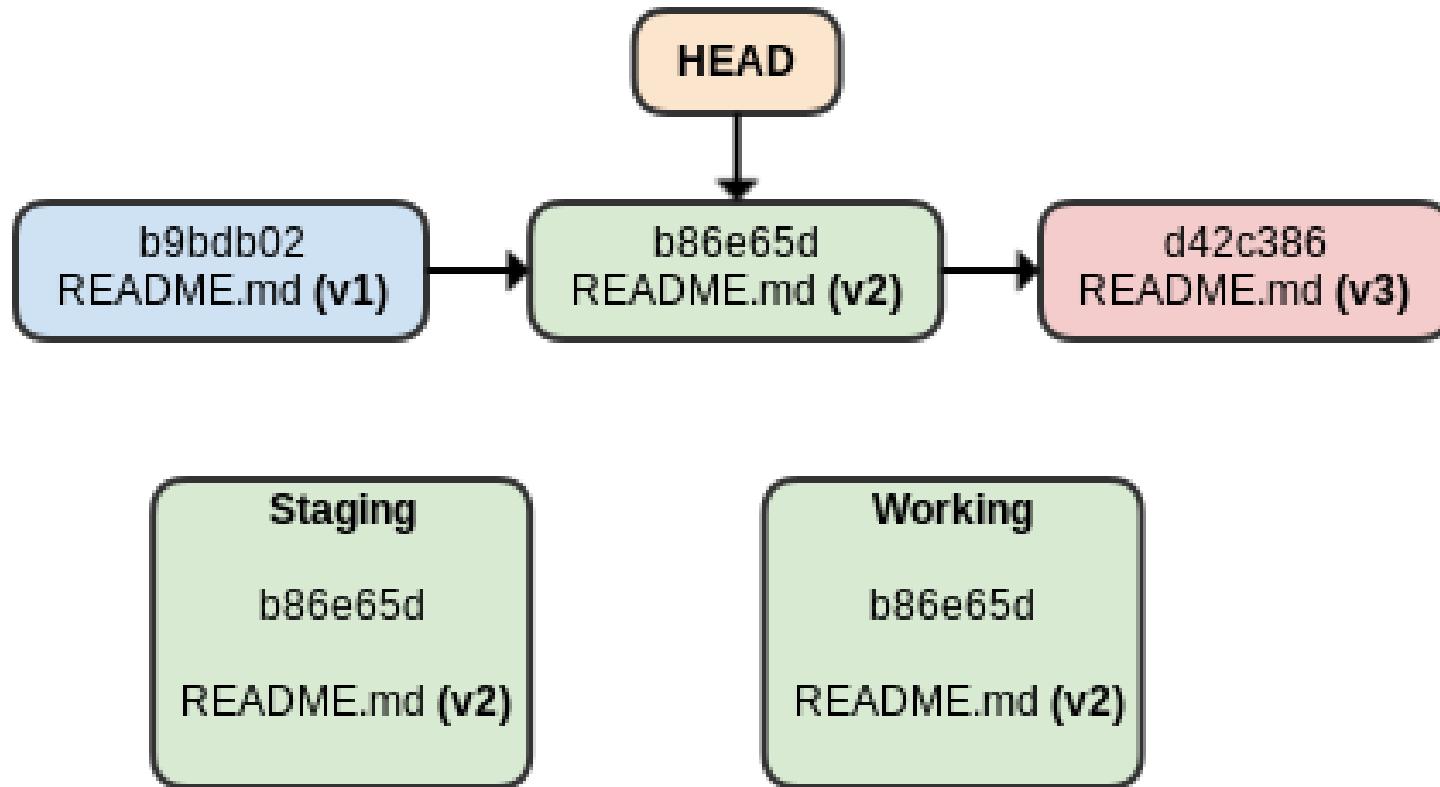


```
git reset --hard HEAD^
```

HARD RESET

```
~/repo$ git reset --hard HEAD^  
~/repo$ git status  
On branch new-feature  
nothing to commit, working tree clean
```

UNDERSTANDING THE THREE STATES



```
git reset --hard HEAD^
```

OH NO!

I `reset --hard` and lost important work!

GIT DOES NOT REMOVE DATA EASILY.

UNDERSTANDING THE REFLG

```
~/repo$ git reflog
```

b86e65da HEAD@{0}: reset: moving to HEAD^

d42c3864 HEAD@{1}: commit: this is not a good change

b86e65da HEAD@{2}: commit: the immediately previous change (totally

b9bdb02f HEAD@{3}: checkout: moving from main to new-feature

b9bdb02f HEAD@{4}: commit: a really good change

```
~/repo$ git reset --hard HEAD@{1}
```

HEAD is now at d42c3864 this is not a good change

UNDERSTANDING THE REFLLOG

```
~/repo$ git reflog
d42c3864 HEAD@{0}: reset: moving to HEAD@{1}
b86e65da HEAD@{1}: reset: moving to HEAD^
d42c3864 HEAD@{2}: commit: this is not a good change
b86e65da HEAD@{3}: commit: the immediately previous change (totally
b9bdb02f HEAD@{4}: checkout: moving from main to new-feature
b9bdb02f HEAD@{5}: commit: a really good change
```

```
~$ git log --oneline
d42c386 this is not a good change
b86e65d the immediately previous change (totally normal)
b9bdb02 a really good change
```

HEAD NOTATION

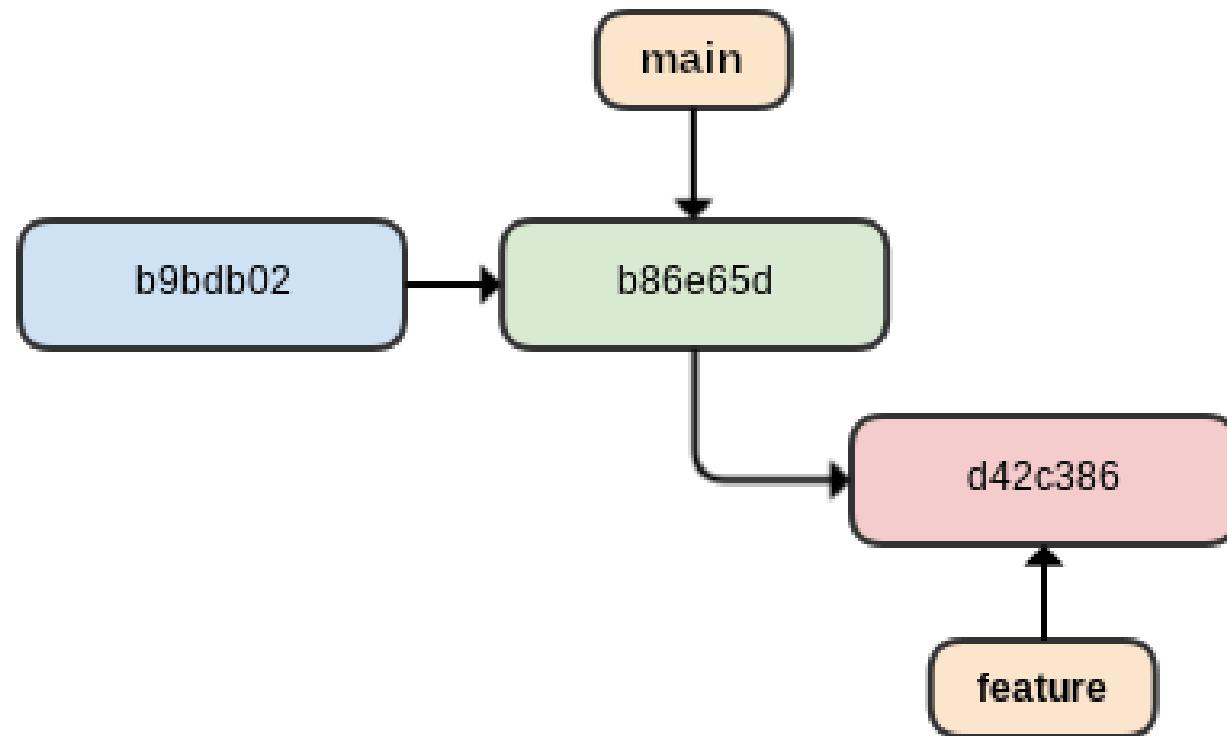
- `HEAD^` (back one place from current HEAD)
- `HEAD^^` (back two places from current HEAD)
- `HEAD~n` (back 'n' places from current HEAD)
- `HEAD@{i}` (back to reflog index 'i')

PLAYING NICE WITH OTHERS

`merge`, `rebase`, and `cherry-pick`

MERGING

With no divergent changes...



FAST FORWARD

With no divergent changes... we can "fast forward"

```
~/repo$ git checkout main  
~/repo$ git merge feature
```

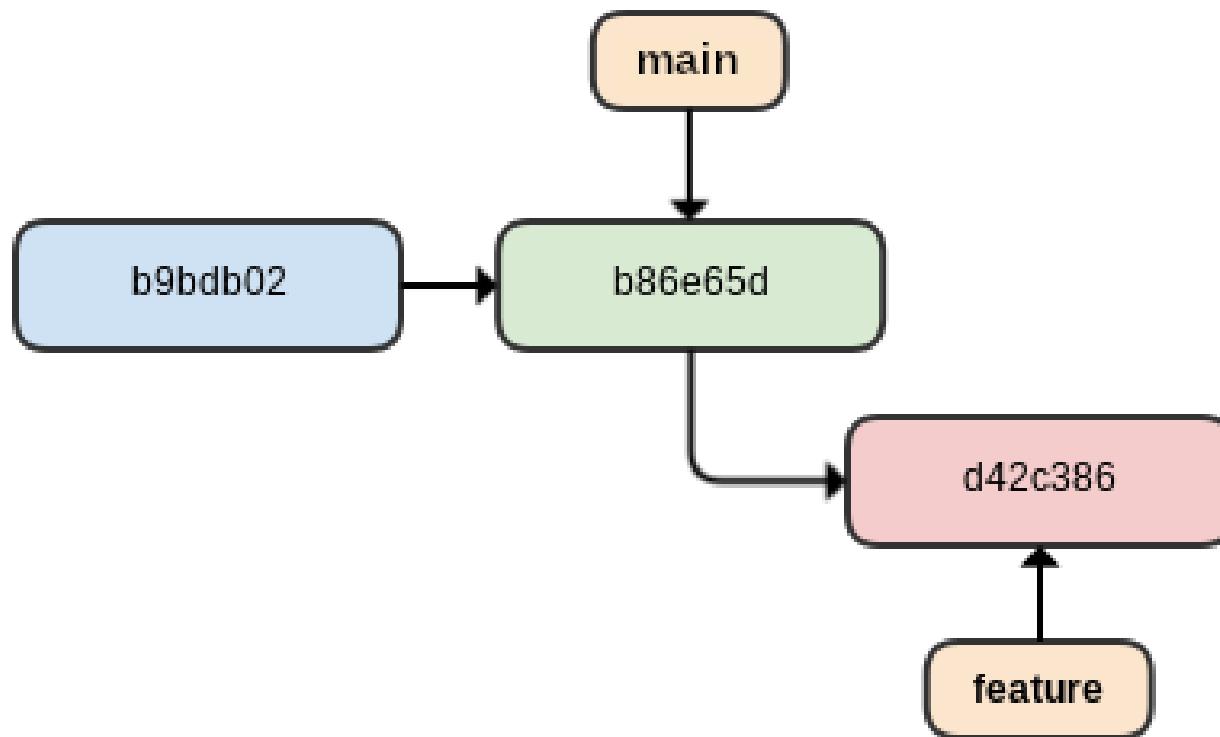
```
Updating b86e65d..d42c386
```

Fast forward

```
source/some-file.js | 13 +++++---++  
1 files changed, 7 insertions(+), 2 deletions(-)
```

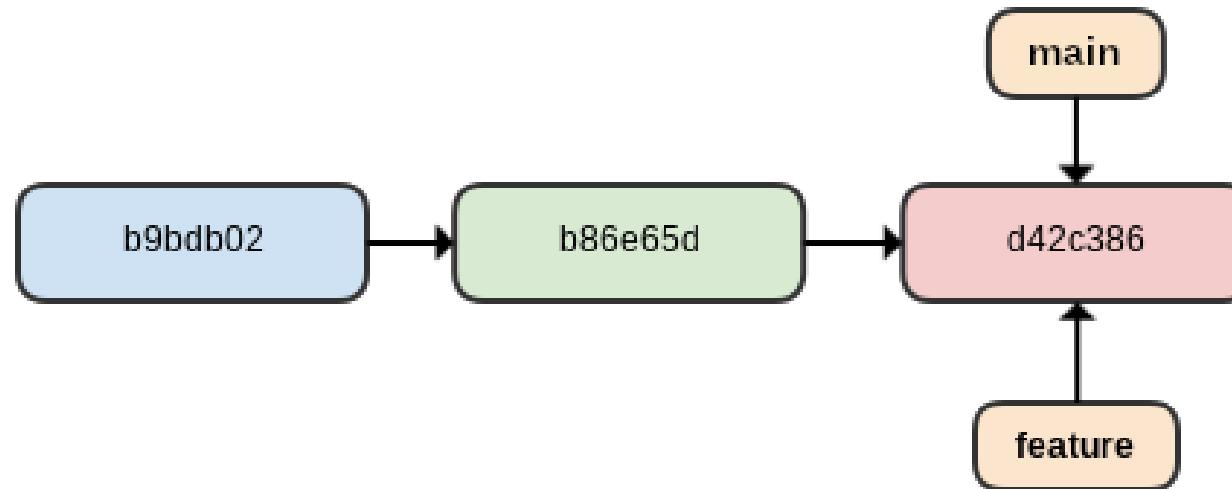
MERGING

With fast forward...



MERGING

With fast forward...



NO FAST FORWARD

```
~/repo$ git checkout main  
~/repo$ git merge feature --no-ff
```

WHY?

Because you lose merge history with fast-forward.

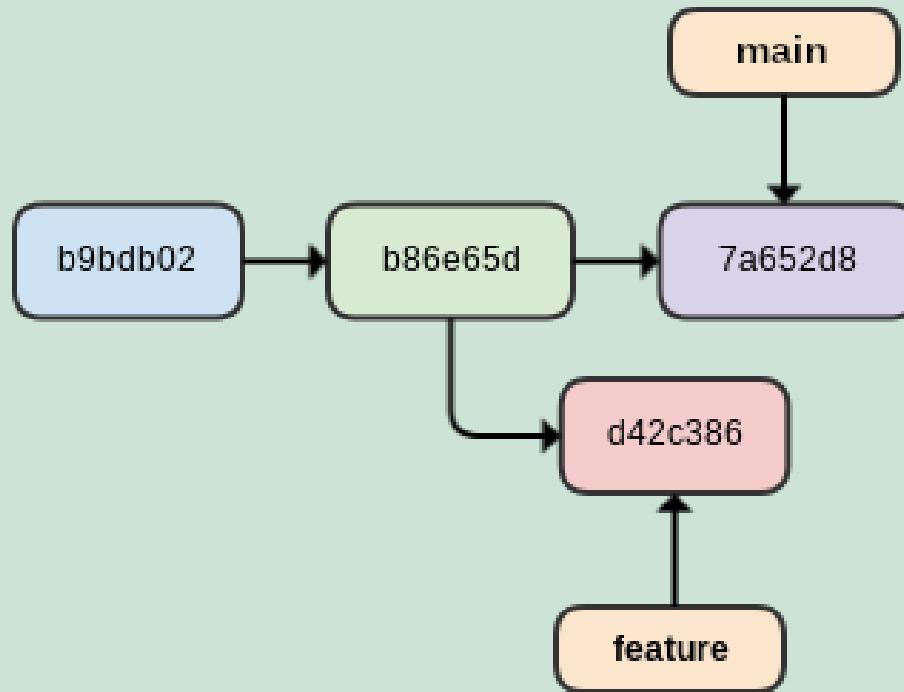
NO FAST FORWARD

```
~/repo$ git log --oneline
```

```
d42c3869 Added new API route to readme docs
b9bdb026 Various grammar edits to documentation
c2cb87e3 Remove unused files from test harness
3e3a8fdf Merge branch 'feature-two-tweak' into feature-two
79eec03a Reorganized documentation sections and renamed
....
```

DIVERGENT CHANGES

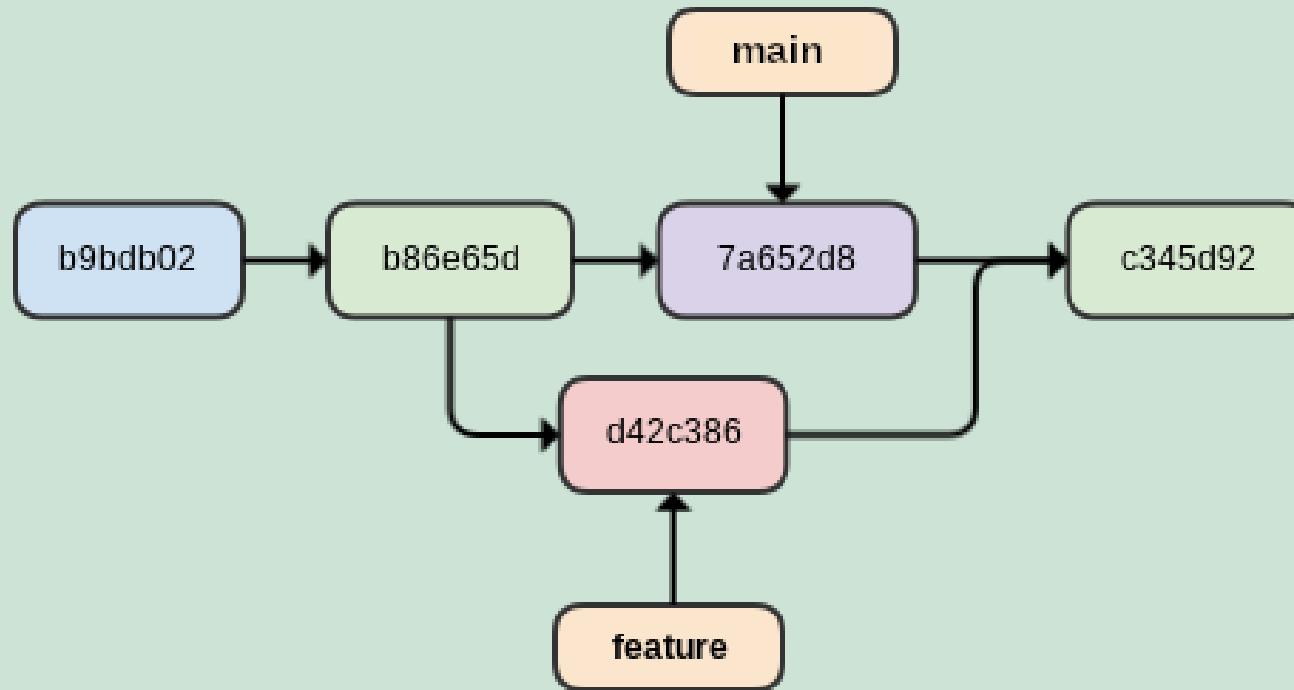
When code in `feature` diverges from the current trunk...



```
~/repo$ git checkout main  
~/repo$ git merge feature
```

DIVERGENT CHANGES

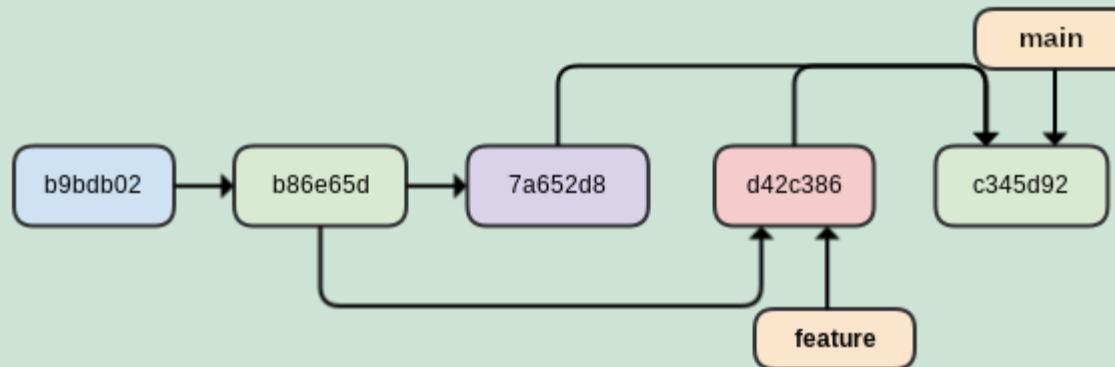
When code in `feature` diverges from the current trunk...



```
~/repo$ git checkout main
~/repo$ git merge feature
```

DIVERGENT CHANGES

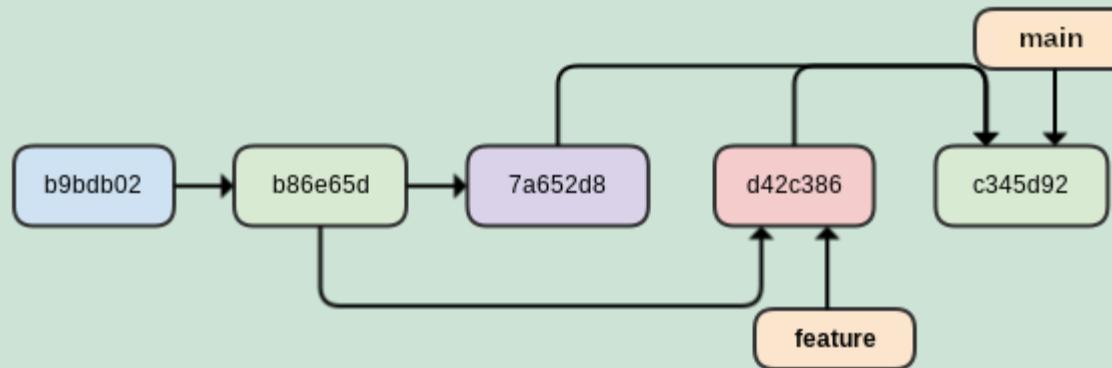
When code in `feature` diverges from the current trunk...



```
~/repo$ git checkout main
~/repo$ git merge feature
```

DIVERGENT CHANGES

When code in `feature` diverges from the current trunk...



```
~/repo$ git log --oneline
c345d92 Merge branch 'feature' into main
d42c386 Added new-feature to library
7a652d8 A piece of divergent work
b86e65d Some older work
```

MERGE CONFLICTS

CONFLICTS

Git can't resolve multiple changes to the same lines.

```
~/repo$ git checkout main  
~/repo$ git merge new-feature
```

Auto-merging README.md

CONFLICT (content): Merge conflict in README.md

Automatic merge failed; fix conflicts and then commit the result.

RESOLVING THE CONFLICT

1. Fix the conflict in the file!

2. Stage the fixed file

```
~$ git add README.md
```

3. Commit the files

(This is your merge commit, it may include other files!)

```
~$ git commit
```

CONFLICTS

Opening up the `README.md` file...

...some text common to both branches

<<<<< HEAD

text only in main

=====

same line, different text in branch

>>>>> new-feature

more common text...

RESOLVING THE CONFLICT

Struggling to resolve the conflict?

```
~/repo$ git merge --abort
```

Abort and ask for help!

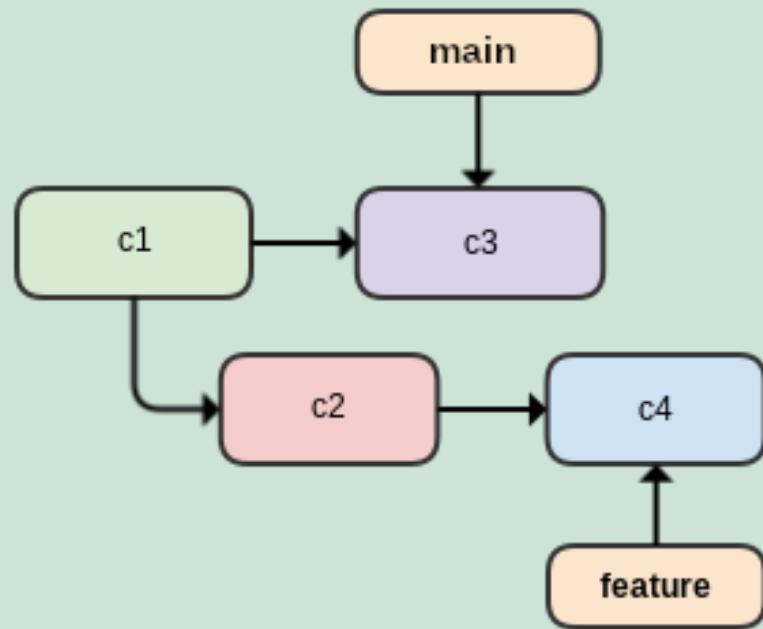
REBASING

Rebasing **rewrites commit history!**

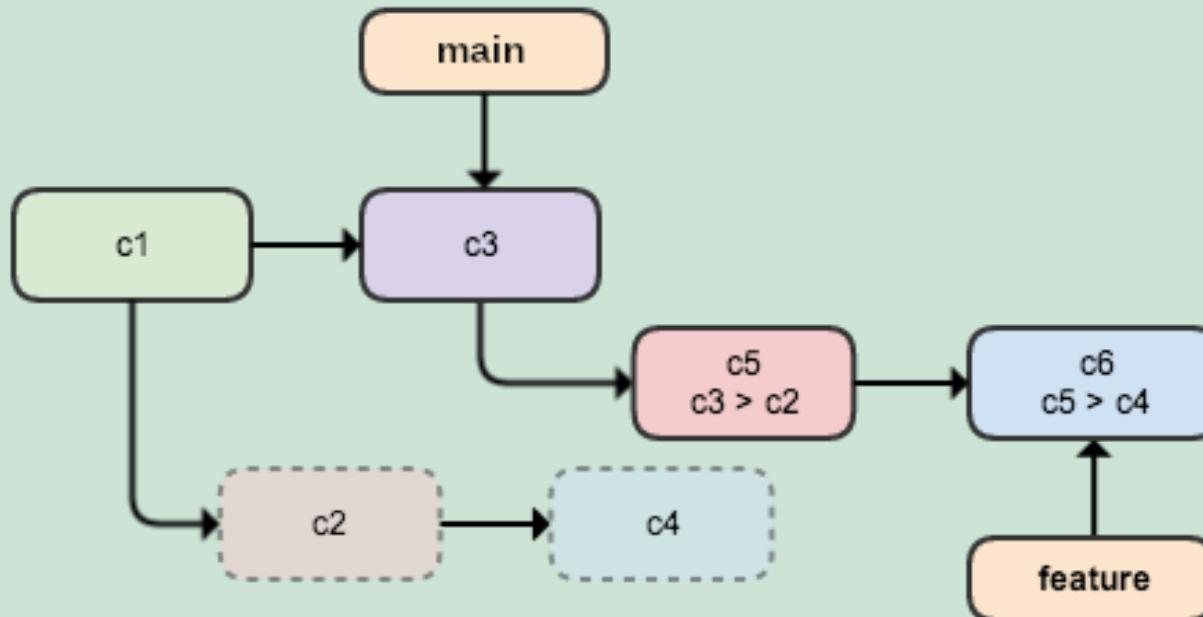
Use this locally, or for your own branches.

Only use with a remote branch if your team has a well-defined git workflow and everyone is using rebase.

DIVERGENT CHANGES



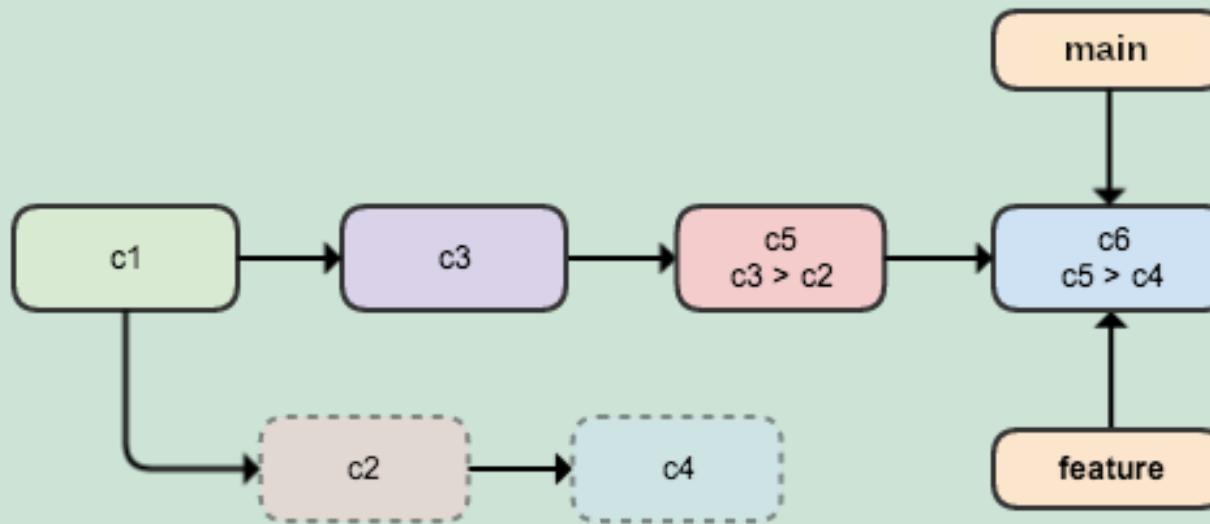
REWRITING HISTORY



```
~/repo$ git checkout feature
~/repo$ git rebase main
```

UPDATING MAIN

We can update `main` with a fast-forward merge.



```
~/repo$ git checkout main
~/repo$ git merge feature
```

YOU CAN STILL GET CONFLICTS WITH REBASE !

REBASE CONFLICTS

```
~/repo$ git rebase main
```

First, rewinding head to replay your work on top of it...

Applying: feature

error: patch failed: README.md:1

error: README.md: patch does not apply

Using index info to reconstruct a base tree...

Falling back to patching base and 3-way merge...

Auto-merging README.md

CONFLICT (content): Merge conflict in README.md

Failed to merge in the changes.

RESOLVING THE CONFLICT

. Fix the conflict in the file!

2. Stage the fixed file

```
~$ git add README.md
```

3. Commit the fix

```
~$ git commit
```

4. Continue the rebase

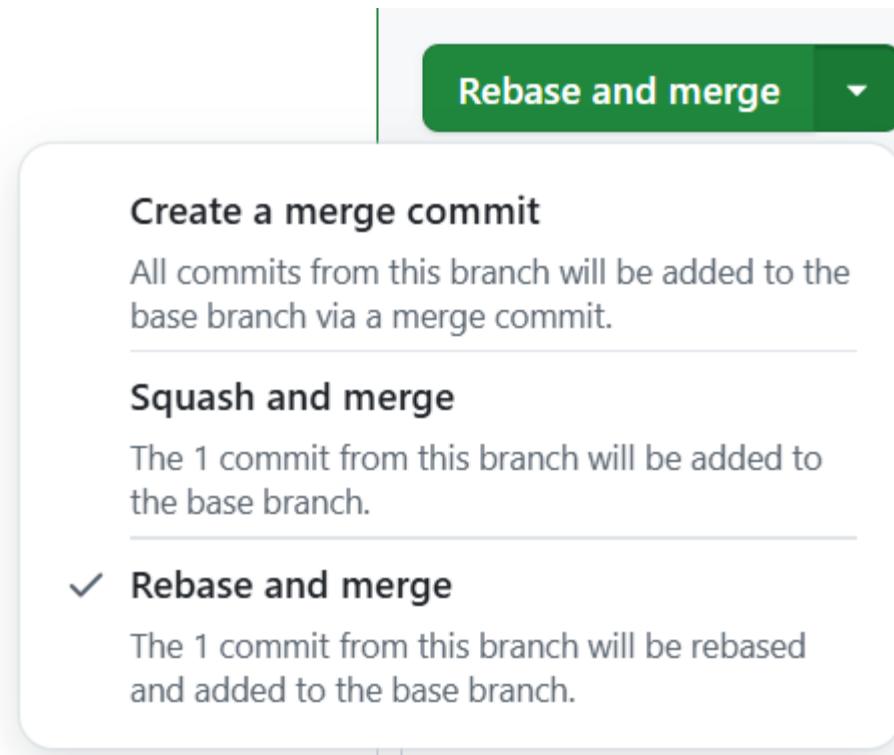
```
~$ git rebase --continue
```

NOT GOING WELL?

You can abort the rebase just like a merge:

```
~/repo$ git rebase --abort
```

USE GITHUB? NO PROBLEM.



A RECOMMENDATION...

If you plan to use rebase versus merge, **always** use rebase.

```
~/repo$ git pull
```

```
~/repo$ git fetch && git merge
```

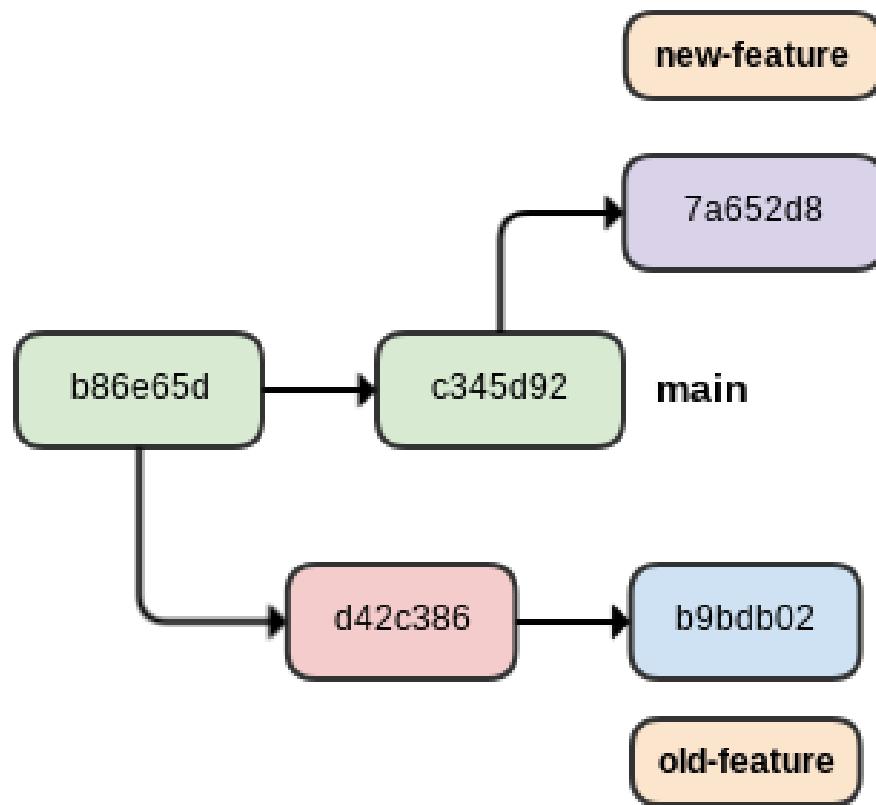
```
~/repo$ git pull --rebase
```

```
~$ git config branch.main.rebase true
```

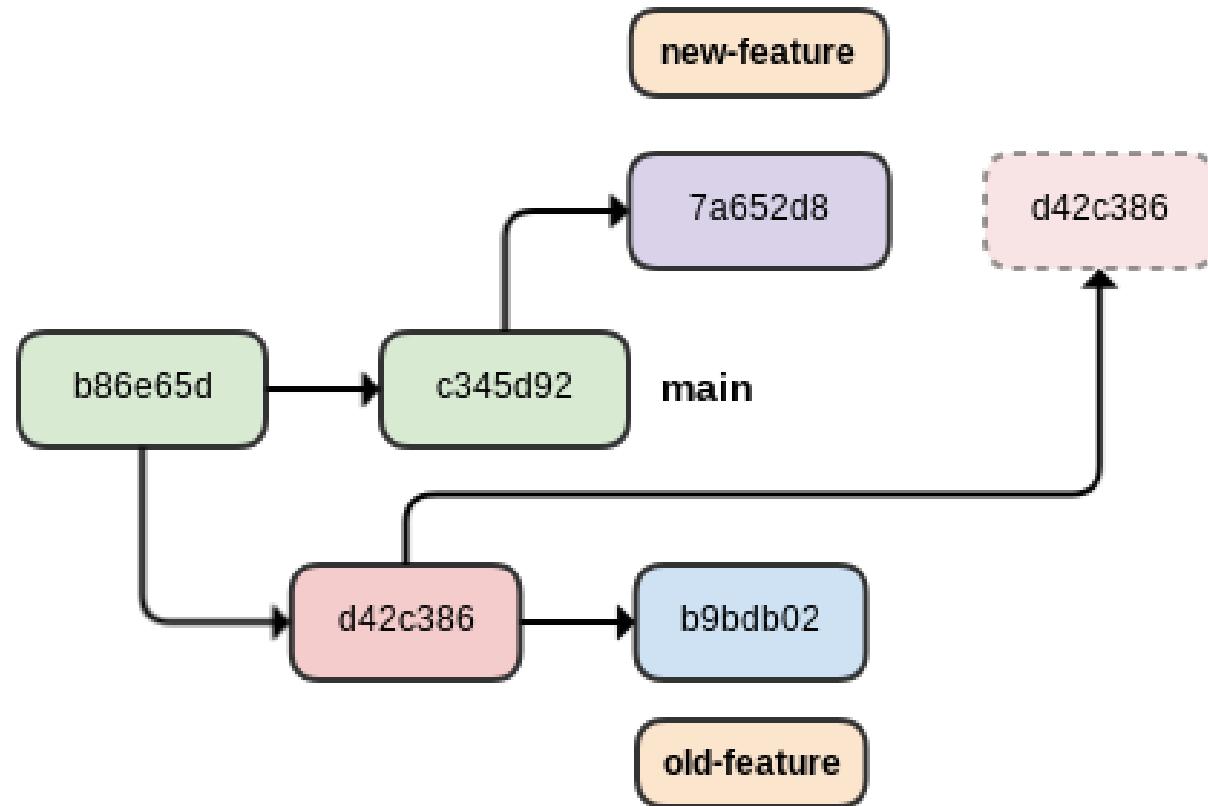
```
~$ git config branch.autosetuprebase always
```

CHERRY-PICKING

CHERRY PICK

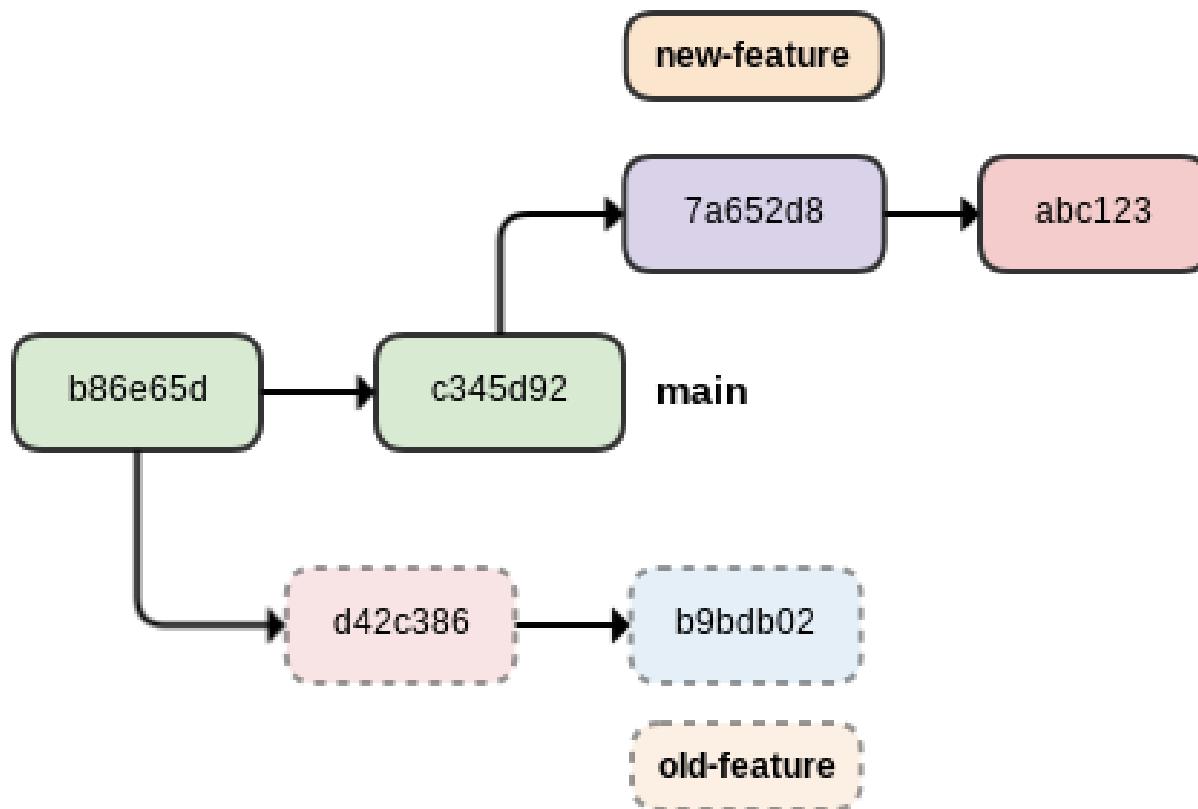


CHERRY PICK



```
~/repo$ git checkout new-feature
~/repo$ git cherry-pick d42c386
```

CHERRY PICK



```
~$ git branch -D old-feature
```

CHERRY PICK

Yes, you can still get conflicts with a `cherry-pick`!
Don't keep that old branch around!

THANK YOU!

GITTING MORE OUT OF GIT

Jordan Kasper | [@jakerella](#)
jordankasper.com/git

Session Feedback:



BONUS CONTENT!

GIT AND DATA INTEGRITY

Git uses snapshots

(versus file diffs)

FILE DIFFS

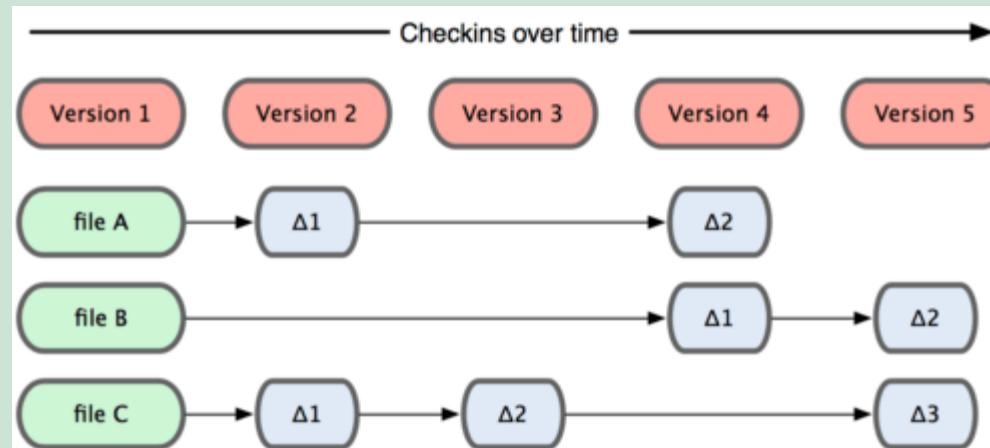


Image credit: <http://git-scm.com/book/en/Getting-Started-Git-Basics>

SNAPSHOTS

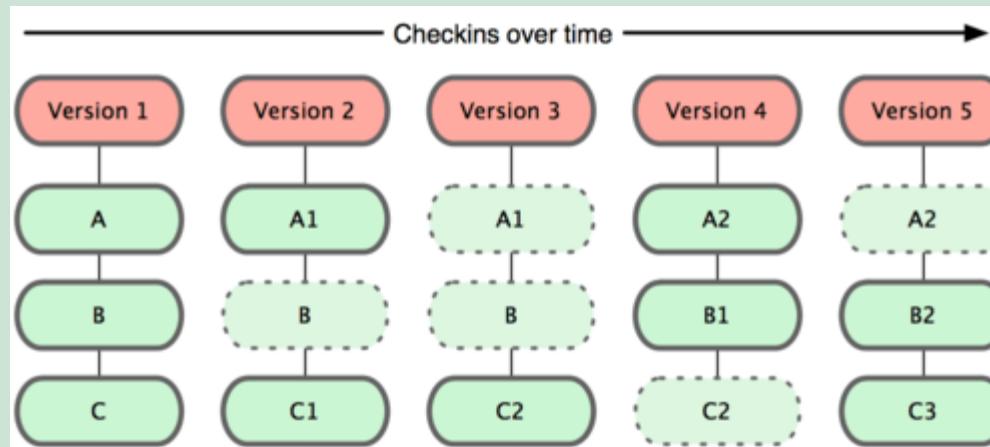


Image credit: <http://git-scm.com/book/en/Getting-Started-Git-Basics>

COMMIT HASHES

```
~/repo$ git commit -m "Added documentation"  
[main (root-commit) b9bdb026] Added documentation
```

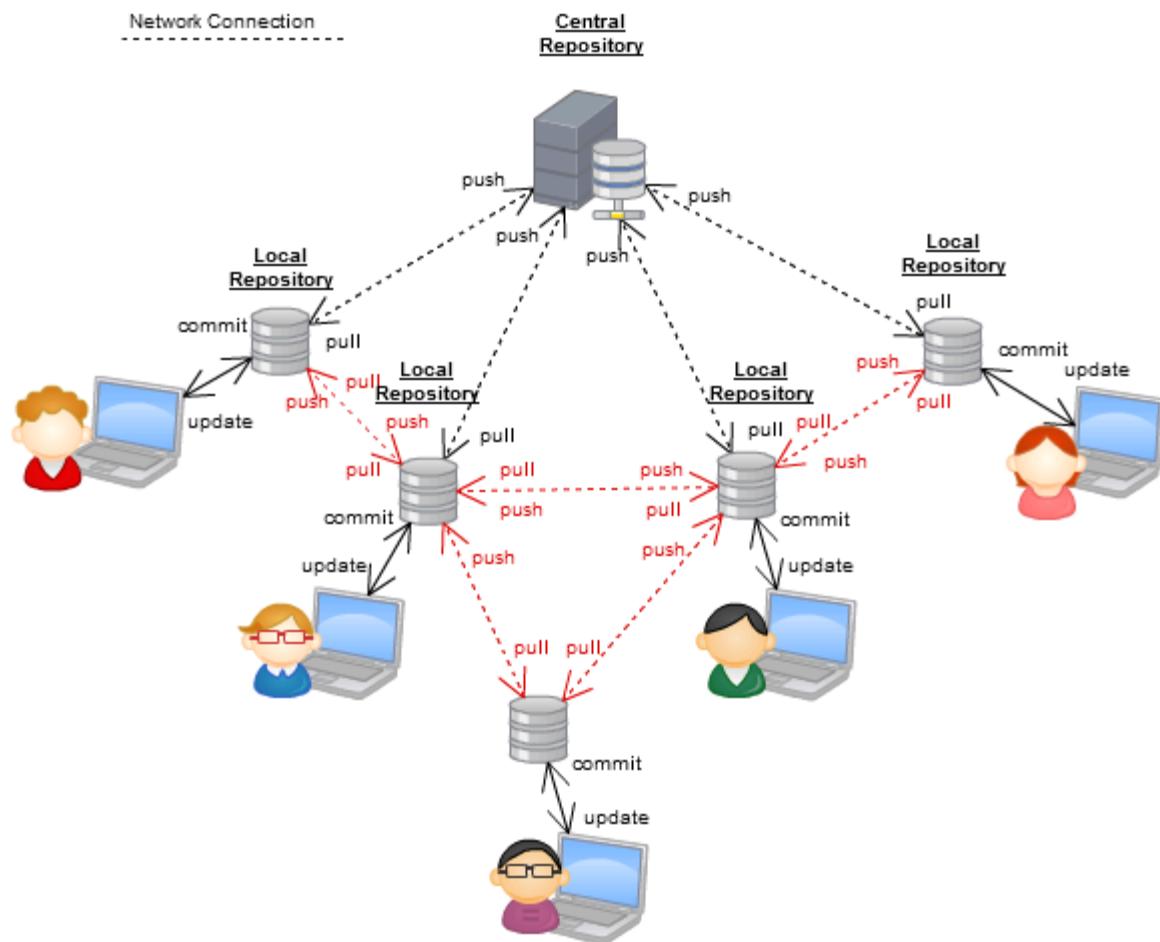
```
~/repo$ git log  
commit b9bdb0261ee9bcaa31d7eb062c0bcafee3e530f0  
Author: jdoe <john@doe.com>  
Date:   Thu Jan 29 11:27:14 2026 -0400  
  
Added documentation
```

A Git commit hash is a checksum of *everything* in that repository.
Every file, every character, every commit message, etc.

LOCAL REPOS

Every local instance of a git repo has ALL of the data.
That means when GitHub goes down, you keep working...
...even with other people / machines.

A DISTRIBUTUED VCS



LOGS



Image credit: <http://thedailyomnivore.net/2012/02/20/ren-stimpy/>

THE BASIC LOG

```
~/repo$ git log  
commit d42c38691161f42fcc07d806f7df4579e8cd189e  
Author: jakerella <me@example.com>  
Date:   Fri Jan 30 13:57:36 2026 -0400
```

Added new API route to readme docs

```
commit b9bdb0261ee9bcaa31d7eb062c0bcafee3e530f0  
Author: jakerella <me@example.com>  
Date:   Thu Jan 29 09:14:16 2026 -0400
```

Various grammar edits to documentation

LOG OPTIONS

```
~/repo$ git log --oneline
```

```
d42c3869 Added new API route to readme docs  
b9bdb026 Various grammar edits to documentation  
c2cb87e3 Remove unused files from test harness  
2e8a119c Merge pull request #186 from jakerella/new-feature  
79eec03a Reorganized documentation sections and renamed  
...
```

FORMATTING AND GRAPHING

```
~/repo$ git log --pretty=format:"%h %as %an: %s" --graph
* 714e0dd9 2026-01-30 jakerella: Merge pull request #220 from foobar
|\ \
| * c8bcfc14 2026-01-29 roro: last update for feature two
| * 3e3a8fdf 2026-01-28 jakerella: Merge branch 'feature-two-tweak'
| |\ \
| | * 46f4cf12 2026-01-28 drnick: tweak for feature two
| * | 1471f2b8 2026-01-27 drnick: one more thing on feature two
| |\ /
| * 5ce150bc 2026-01-26 drnick: feature two initial work
| /
* 062b1f01 2026-01-25 roro: some previous work
```

MAKE IT EASY
Add an alias to your bash or gitbash profile:

```
alias gl="git log --pretty='tformat:%h %as %an: %s' --graph"
```

FILTERING THE LOG

```
~/repo$ git log --no-merges
```

```
~/repo$ git log --author="jakerella"
```

```
~/repo$ git log --before="2026-02-01"
```

```
~/repo$ git log -- source/routes
```

```
~/repo$ git log --grep=4\.31\.[0-9]+ -E
```

POINTING BLAME
(Possibly at yourself...)

BLAME GAME

```
1. function divide(x, y) { ... }
2. function multiply(x, y) { ... }
3. function add(x, y) {
4.     return x + x;
5. }
6. function subtract(x, y) { ... }
```

```
~/repo$ git blame source/math.js
```

ca5dc559 (roro	2025-03-13	1) function divide(x, y) { ... }
ca5dc559 (roro	2025-03-13	2) function multiply(x, y) { ... }
d35241e6 (drnick	2026-01-22	3) function add(x, y) {
27cc7225 (jakarella	2026-01-29	4) return x + x;
d35241e6 (drnick	2026-01-22	5) }
d35241e6 (drnick	2026-01-22	6) function subtract(x, y) { ... }

BLAME GAME

Know where to look?

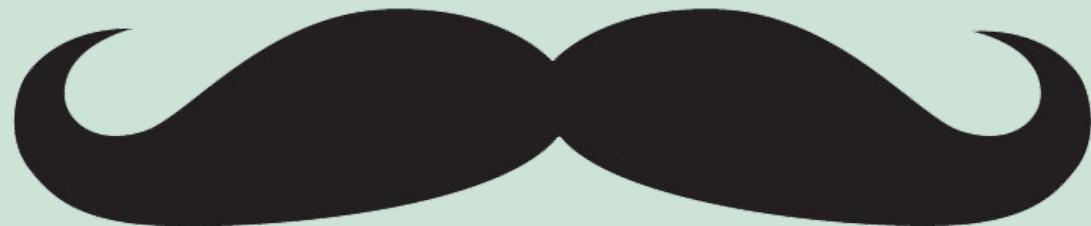
```
~/repo$ git blame -L3,5 source/utilities.js  
d35241e6 (drnick      2026-01-22  3) function add(x, y) {  
27cc7225 (jakerella   2026-01-29  4)     return x + x;  
d35241e6 (drnick      2026-01-22  5) }
```

```
~/repo$ git log --oneline source/utilities.js
```

```
27cc7225 Tweaked math add to double input  
d35241e6 Expanded common math ops for new feature use  
ca5dc559 Created helper file for common math ops
```

My boss just told me to pivot...

Using `git stash`



THE SITUATION

You've made some changes on a feature branch,
but they are not ready to commit yet...

```
~/repo$ git checkout -b hot-fix-123
```

```
~/repo$ git status
On branch hot-fix-123
Changes not staged for commit:
  modified:  borked-code.js
  modified:  some-unrelated-file.js
```

```
Untracked files:
  new-code.js
```

SAVING YOUR STASH

Before switching branches...

```
~/repo$ git stash --include-untracked  
Saved working directory and index state WIP on new-feature: 3c927dd  
HEAD is now at 3c927dd Add a README
```

```
~/repo$ git status  
On branch new-feature  
nothing to commit, working tree clean
```

MANAGING YOUR STASH

```
~/repo$ git stash list  
stash@{0}: WIP on new-feature: 3c927dd Add a README
```

But what if you **stash** multiple times?!

```
~/repo$ git stash list  
stash@{0}: WIP on new-feature: 3c927dd Add a README  
stash@{1}: WIP on new-feature: 3c927dd Add a README  
stash@{2}: WIP on new-feature: 3c927dd Add a README  
stash@{3}: WIP on hot-fix-456: a46c33b Replaced old lib with new one
```

NAMING YOUR STASH

```
~/repo$ git stash save 'work on the API for new feature'
```

```
~/repo$ git stash list
stash@{0}: On new-feature: work on the API for new feature
stash@{1}: On new-feature: trying out some async behavior
stash@{2}: On hot-fix-456: attempt to fix bug #456, but incomplete
```

GREAT, BUT HOW DO I USE IT?

APPLYING YOUR STASH

```
~/repo$ git checkout new-feature  
~/repo$ git stash apply
```

On branch new-feature

Changes not staged for commit:

```
  modified: borked-code.js  
  modified: some-unrelated-file.js
```

Untracked files:

```
  new-code.js
```

```
~/repo$ git stash list  
stash@{0}: On new-feature: work on the API for new feature  
stash@{1}: On new-feature: trying out some async behavior  
stash@{2}: On hot-fix-456: attempt to fix bug #456, but incomplete
```

DROPPING YOUR STASH

```
~/repo$ git stash drop stash@{0}  
Dropped refs/stash@{0} (a52d9c3ba1121dd94eb3925ba60d3f8ef30540c8)
```

Make sure you know which stash you're dropping!

POPPING YOUR STASH

Apply and drop in one command!

```
~/repo$ git stash pop
```

On branch new-feature

Changes not staged for commit:

 modified: borked-code.js

 modified: some-unrelated-file.js

Untracked files:

 new-code.js

Dropped refs/stash@{0} (a52d9c3ba1121dd94eb3925ba60d3f8ef30540c8)

