GITTING MORE OUT OF GIT

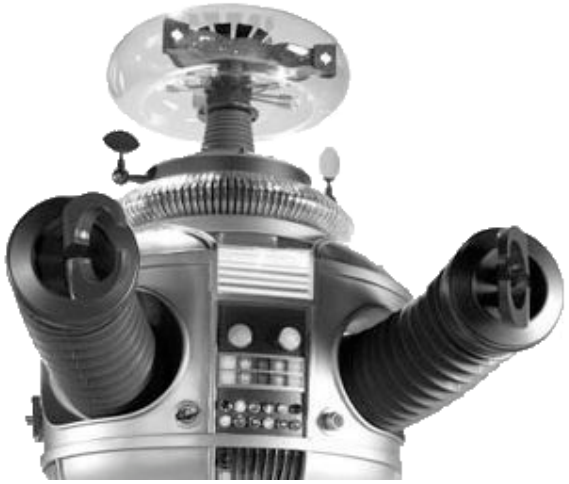
Jordan Kasper

@jakerella

StrongLoop™

Express

LoopBack



DANGER!

GIT IS DISTRIBUTED

A CENTRALIZED VCS

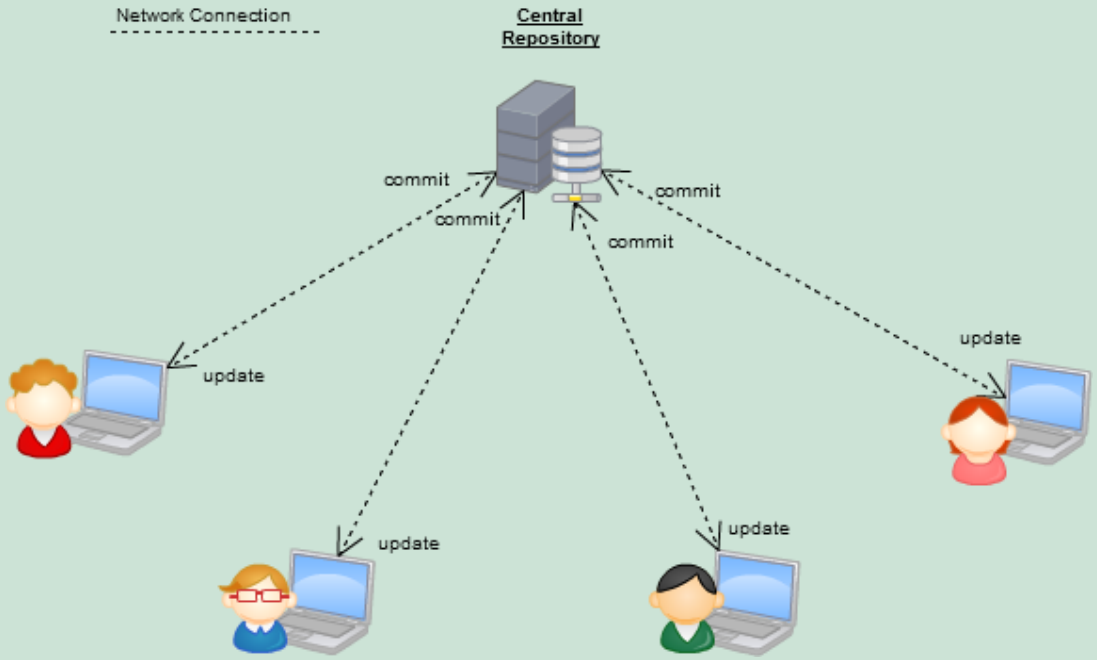


Image credit: https://docs.joomla.org/Dvcs

A DISTRIBUTUED VCS

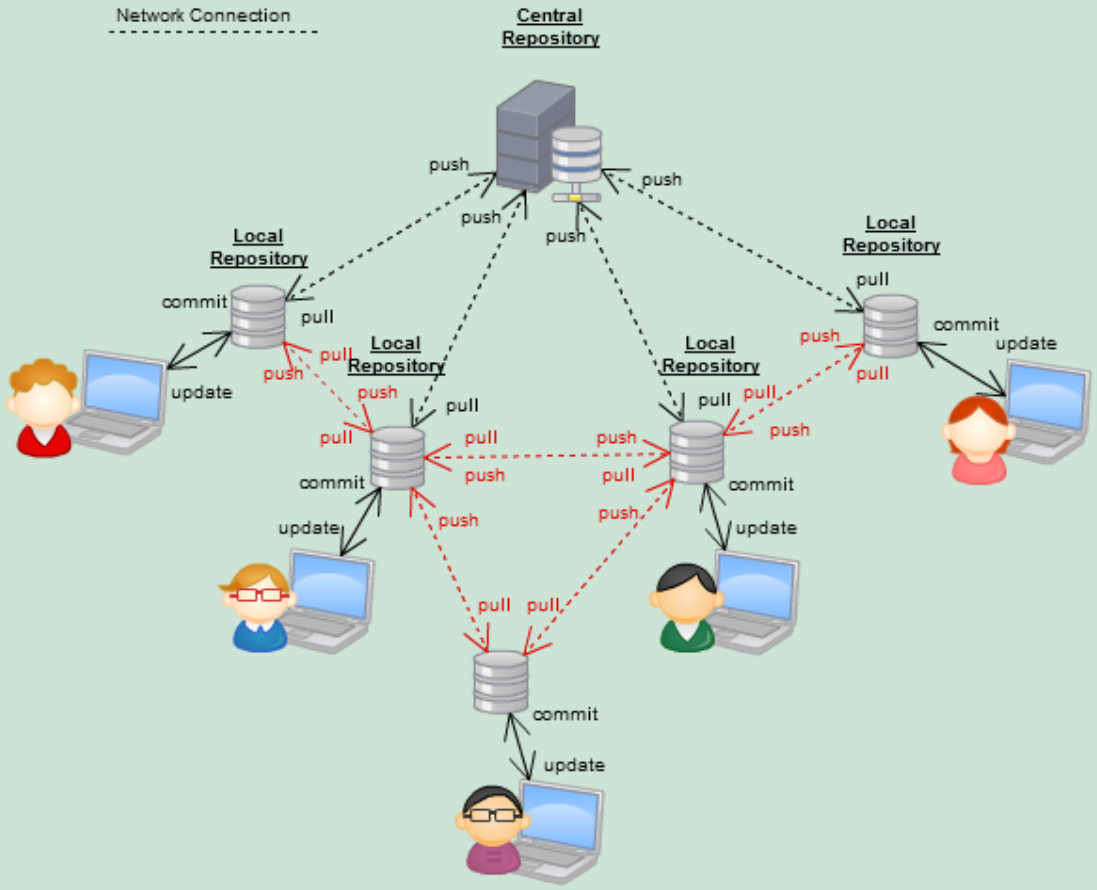


Image credit: https://docs.joomla.org/Dvcs

A DISTRIBUTUED VCS (IN PRACTICE)

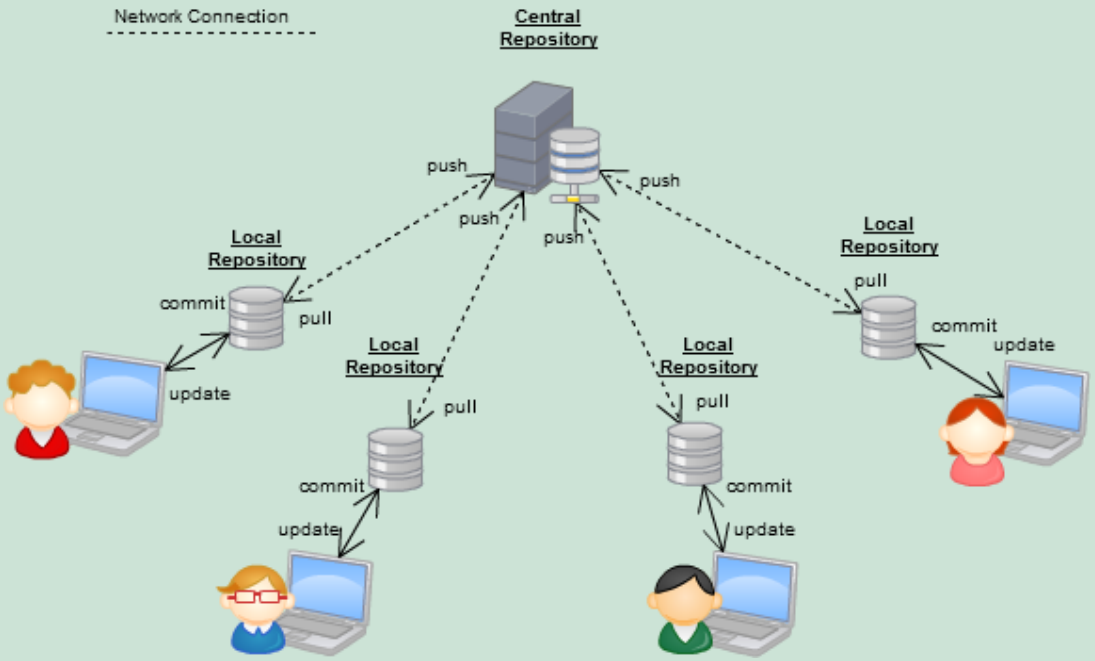


Image credit: https://docs.joomla.org/Dvcs

REMOTES

Just like it sounds, a "remote" is a repo outside your environment.

CLONING

```
~$ git clone http://some.repo.com someDirectory
```

```
~$ git branch -a
* master
  remotes/origin/master
```

Cloning creates a remote called "origin"

TRACKING

A local branch may "track" a remote URL...

```
~$ git branch -vv
* master              b956c45 [origin/master] Initial commit
  some-local-branch   a74b295 Implemented that cool feature
```

WHEREZITAT?

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

TRACKING

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

```
~$ git checkout -b new-feature
~$ git push -u origin new-feature
```

The `-u` option sets the "upstream" remote repo to track.

```
~$ git branch -vv
  master            b956c45 [origin/master] Initial commit
* new-feature       b956c45 [origin/new-feature] Initial commit
  some-local-branch a74b295 Implemented that cool feature
```

OTHER REMOTES

What about other repos? How is this distributed?

```
~$ git remote add jquery-source https://github.com/jquery/jquery.git
```

```
~$ git remote -v
origin         https://github.com/jakerella/jquery.git (fetch)
origin         https://github.com/jakerella/jquery.git (push)
jquery-source  https://github.com/jquery/jquery.git (fetch)
jquery-source  https://github.com/jquery/jquery.git (push)
```

```
~$ git fetch jquery-source
...
~$ git merge jquery-source/master
```

MOAR BRANCHES

LISTING BRANCHES

```
~$ git branch
  hot-fix
* master
  new-feature
  stuff
```

```
~$ git branch --no-merged
stuff
```

BRANCH DIFFERENCES

```
~$ git diff master stuff
```

```
diff --git a/README.md b/README.md
index a1e4bac..e69de29 100644
--- a/README.md
+++ b/README.md
@@ -1 +0,0 @@
-This is the readme
diff --git a/STUFF.md b/STUFF.md
new file mode 100644
index 0000000..e69de29
```

This is ALL changes between `master` and `stuff` (regardless of what branch those came from).

SINGLE-BRANCH DIFFERENCES

```
~$ git diff master..stuff
diff --git a/STUFF.md b/STUFF.md
new file mode 100644
index 0000000..e69de29
```

SINGLE-BRANCH DIFFERENCES

```
~$ git diff b9bdb02 b86e65d
diff --git a/STUFF.md b/STUFF.md
new file mode 100644
index 0000000..e69de29
```

GIT AND DATA INTEGRITY

Git uses snapshots

(versus file diffs)

FILE DIFFS

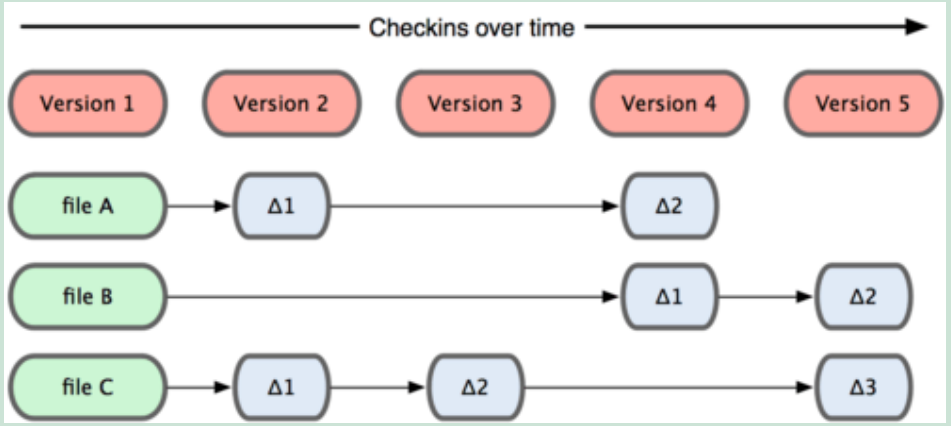


Image credit: http://git-scm.com/book/en/Getting-Started-Git-Basics

SNAPSHOTS

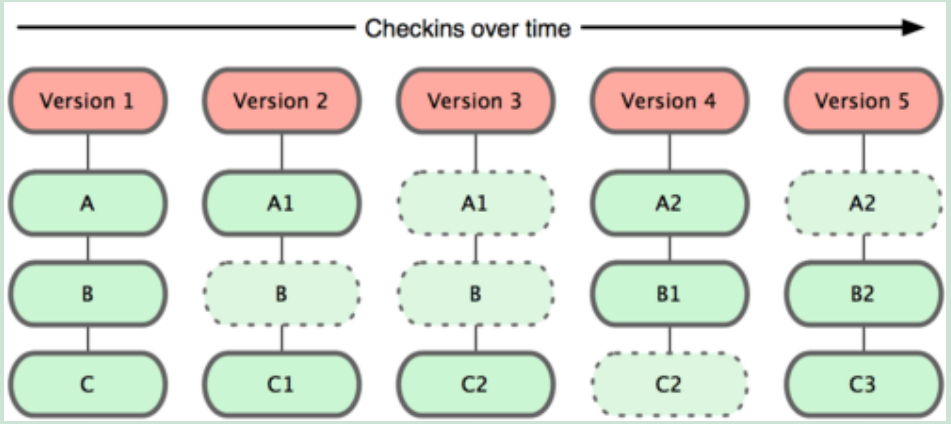


Image credit: http://git-scm.com/book/en/Getting-Started-Git-Basics

COMMIT HASHES

```
~$ git commit -m "Added documentation"
[master (root-commit) b9bdb02] Added documentation
```

```
~$ git log
commit b9bdb0261ee9bcaa31d7eb062c0bcafee3e530f0
Author: jdoe <john@doe.com>
Date:   Fri Jul 11 17:27:14 2014 -0500

Added documentation
```

WHEN THINGS GO WRONG...

FIXING COMMIT MESSAGES

```
~$ git commit -m "This is teh best."
[feature-branch fd0091e] This is teh best.
1 file changed, 1 insertion(+)
```

```
~$ git commit --amend -m "This is the best."
[feature-branch 0486a7d] This is the best.
1 file changed, 1 insertion(+)
```

FIXING COMMIT MESSAGES

```
~$ git commit -m "This is teh best."
[feature-branch fd0091e] This is teh best.
1 file changed, 1 insertion(+)
```

```
~$ git commit --amend -m "This is the best."
[feature-branch 0486a7d] This is the best.
1 file changed, 1 insertion(+)
```

WHAT ABOUT THE LOG?

```
~$ git log
commit 0486a7d61ee9bcaa31d7eb062c0bcafee3e530f0
Author: jdoe <john@doe.com>
Date:   Fri Jul 11 17:27:14 2014 -0500

This is the best.

commit 72267fc26d88fa977d24760252da63b46ca3b81a
Author: fbar <foo@bar.com>
Date:   Fri Jul 10 14:31:36 2014 -0500

I did a thing!
```

NEVER REALLY GONE...

```
~$ git reflog
0486a7d HEAD@{0}: commit (amend): This is the best.
fd0091e HEAD@{1}: commit: This is teh best.
72267fc HEAD@{2}: commit: I did a thing!
...
```

WHAT IF I FORGOT A FILE?

```
~$ git add forgotten.js
~$ git commit --amend
```

AND WHAT IF I NEED TO MODIFY AN OLDER COMMIT?

You'll need to use `git rebase --interactive`

INTERACTIVE REBASING

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

```
pick 72267fc I did a thing!
pick fd0091e This is the best.

# Rebase 7e10e41..fd0091e onto 7e10e41
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
...
```

INTERACTIVE REBASING

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

```
edit 72267fc I did a thing!
pick fd0091e This is the best.

# Rebase 7e10e41..fd0091e onto 7e10e41
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
...
```

```
~$ git rebase --interactive HEAD^^^
Stopped at 72267fc26d88fa977d24760252da63b46ca3b81a... I did a thing!
...
```

Now alter the commit just like before...

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

Careful! You've just changed
***ALL history** from that point forward.*

```
~$ git reflog

5e83670 HEAD@{0}: rebase -i (finish): returning to refs/heads/new-feature
5e83670 HEAD@{1}: rebase -i (pick): This is the best
bcb51f9 HEAD@{2}: commit (amend): I did a thing!
72267fc HEAD@{3}: cherry-pick: fast-forward
7e10e41 HEAD@{4}: rebase -i (start): checkout HEAD^^^
fd0091e HEAD@{9}: commit: This is the best
72267fc HEAD@{10}: commit: I did a thing!
7e10e41 HEAD@{11}: commit: some old commit
```

UNDOING CHANGES

UNSTAGED CHANGES

```
~$ echo "foobar" >> README.md
```

```
~$ git status
# On branch new-feature
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    modified:   README.md
```

UNSTAGED CHANGES

```
~$ git checkout README.md

~$ git status
# On branch new-feature
nothing to commit (working directory clean)
```

```
~$ git checkout src/js/
```

UNSTAGED CHANGES

*Be careful! **NEW** files will not be removed!*

```
~$ git checkout content/
~$ rm NewFile.md
```

STAGED CHANGES

```
~$ echo "foobar" >> README.md
~$ git add README.md
```

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

STAGED CHANGES

```
~$ echo "foobar" >> README.md
~$ git add README.md
```

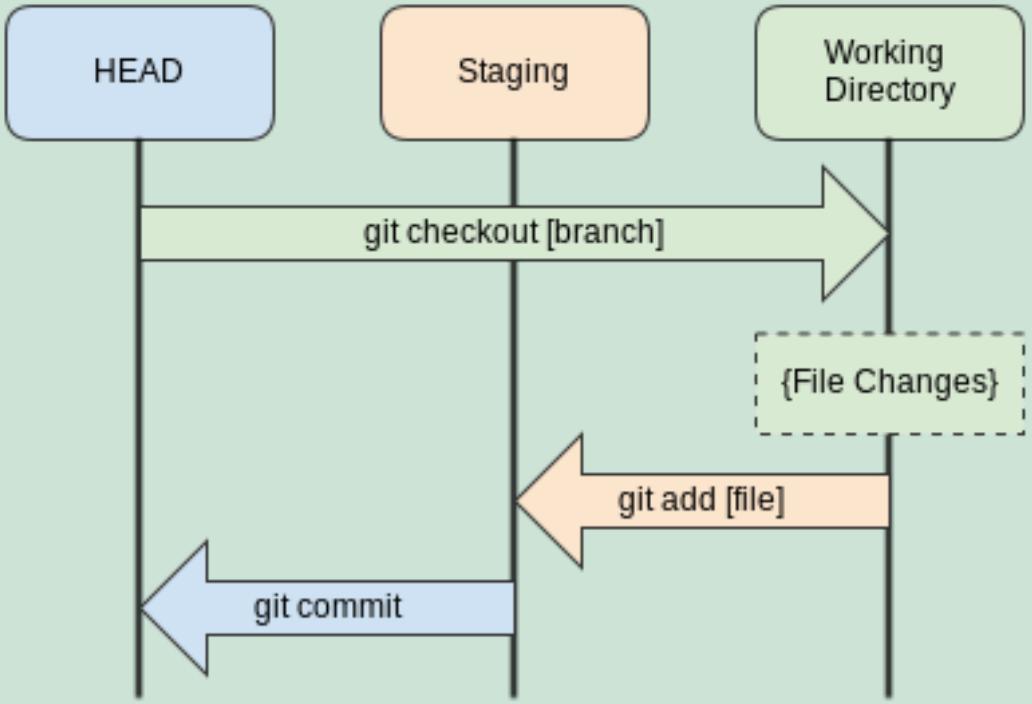
```
~$ git status
# On branch new-feature
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   README.md
```

STAGED CHANGES

```
~$ git reset HEAD README.md
```

```
~$ git status
# On branch new-feature
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   README.md
```

REMEMBER THE THREE STATES!



COMMITTED CHANGES

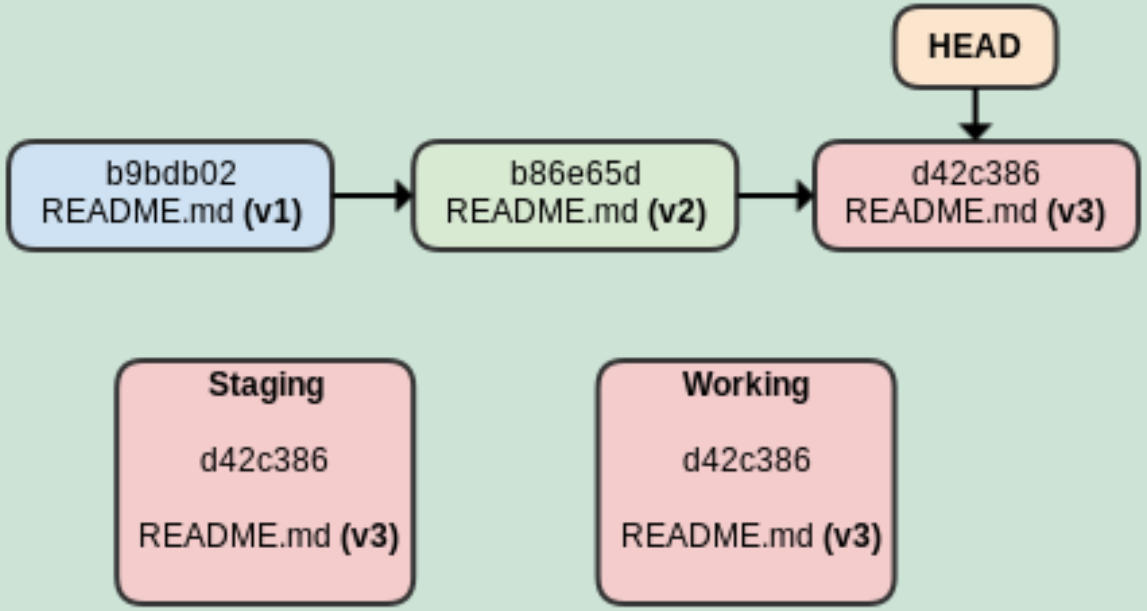
(a.k.a. Oops)

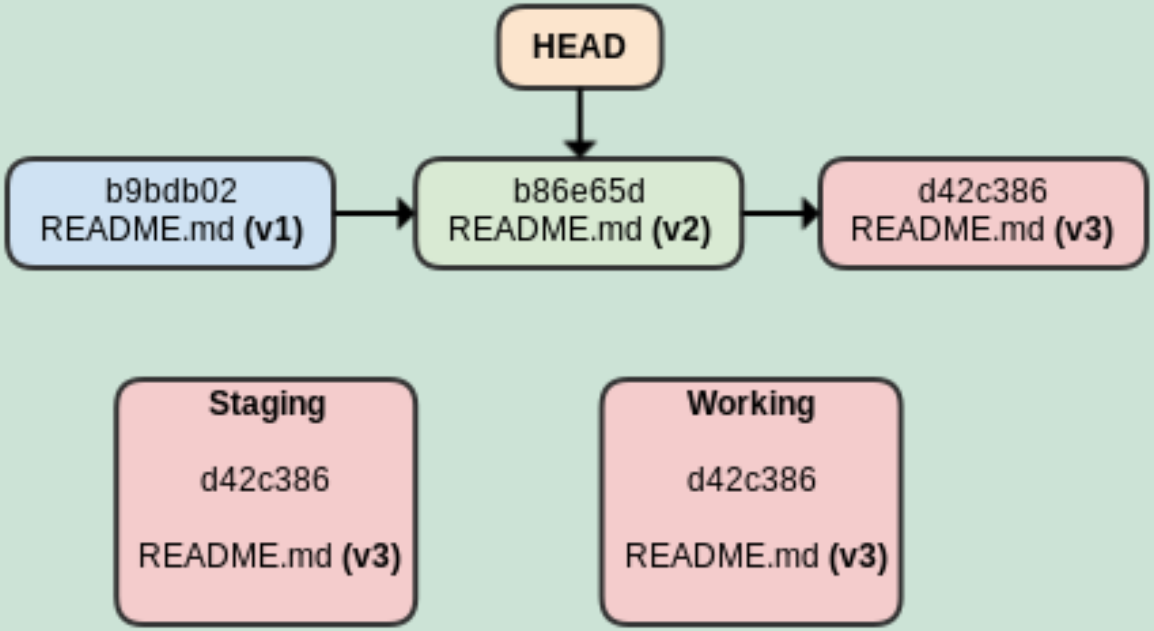
BAD COMMIT

```
~$ echo "foobar" >> README.md

~$ git add README.md
~$ git commit -m "this is not a good change"
```

```
~$ git log
commit d42c38661ee9bcaa31d7eb062c0bcafee3e530f0
Author: jdoe <john@doe.com>
Date:   Fri Jul 11 17:27:14 2014 -0500

this is not a good change
```

UNDERSTANDING THE THREE SEASHELLS



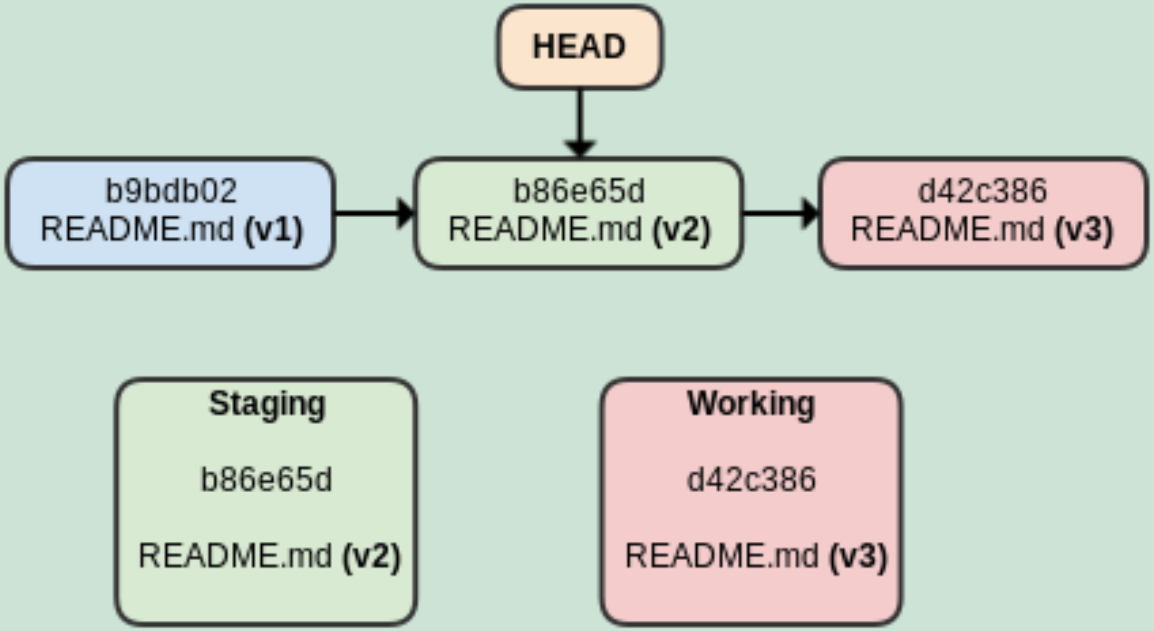



```
git reset --soft HEAD^
```

SOFT RESET

```
~$ git reset --soft HEAD^

~$ git log
commit b86e65dab535623d487e063ef1337cfdecfaf99d
Author: jdoe <john@doe.com>
Date:   Fri Jul 10 12:45:11 2014 -0500

the immediately previous change (totally normal)
```

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

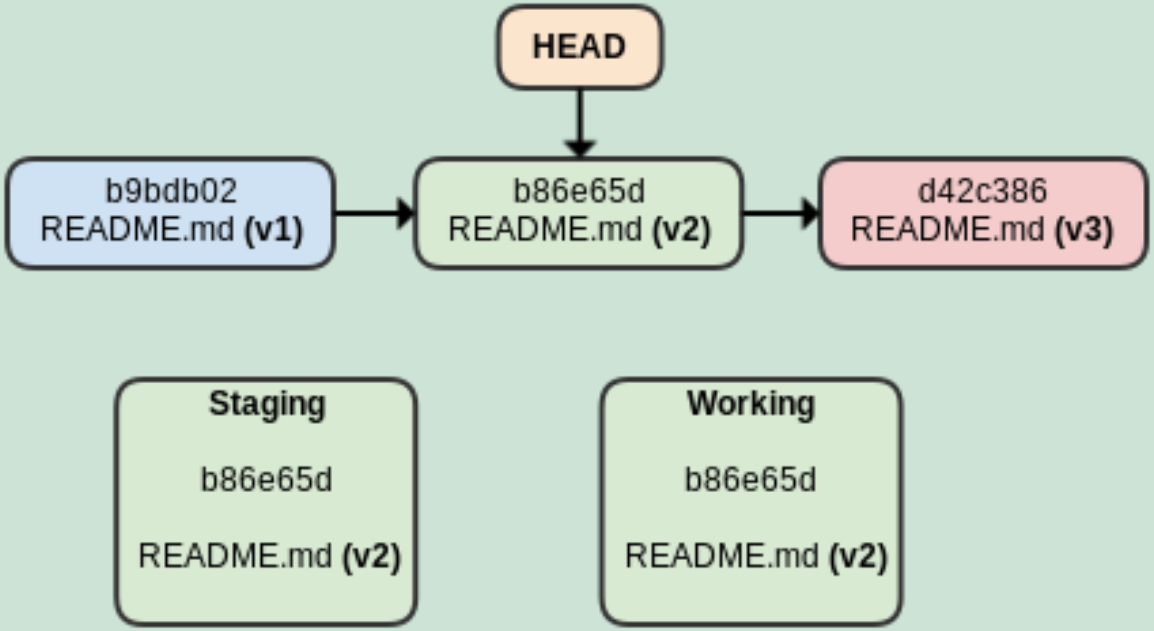



```
git reset --mixed HEAD^
```

MIXED RESET

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

~$ git log
commit b86e65dab535623d487e063ef1337cfdecfaf99d
Author: jdoe <john@doe.com>
Date:   Fri Jul 10 12:45:11 2014 -0500

the immediately previous change (totally normal)
```

```
~$ git status
# On branch new-feature
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   README.md
```




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

HARD RESET

```
~$ git reset --hard HEAD^

~$ git log
commit b86e65dab535623d487e063ef1337cfdecfaf99d
Author: jdoe <john@doe.com>
Date:   Fri Jul 10 12:45:11 2014 -0500

the immediately previous change (totally normal)

~$ git status
# On branch new-feature
nothing to commit (working directory clean)
```

OH NO!

I reset --hard and lost some work...

UNDERSTANDING THE REFLOG

```
~$ git reflog

b86e65d HEAD@{0}: reset: moving to HEAD^
d42c386 HEAD@{1}: commit: this is not a good change
b86e65d HEAD@{2}: commit: the immediately previous change (totally normal)
e098369 HEAD@{3}: checkout: moving from master to new-feature
b9bdb02 HEAD@{4}: commit: a really good change
e098369 HEAD@{5}: pull: Fast-forward
```

UNDERSTANDING THE REFLOG

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

```
~$ git reflog

d42c386 HEAD@{0}: reset: moving to HEAD@{1}
b86e65d HEAD@{1}: reset: moving to HEAD^
d42c386 HEAD@{2}: commit: this is not a good change
b86e65d HEAD@{3}: commit: the immediately previous change (totally normal)
e098369 HEAD@{4}: checkout: moving from master to new-feature
b9bdb02 HEAD@{5}: commit: a really good change
e098369 HEAD@{6}: pull: Fast-forward
```

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')

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...

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

~$ git status
# On branch hot-fix-123
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   some-unrelated-file.js
```

USING THE STASH

Before switching branches...

```
~$ git stash --include-untracked
Saved working directory and index state WIP on new-feature: 3c927dd Added text to reac
HEAD is now at 3c927dd Added text to readme
```

```
~$ git status
# On branch new-feature
nothing to commit (working directory clean)
```

MANAGING YOUR STASH

```
~$ git stash list
stash@{0}: WIP on new-feature: 3c927dd Added text to readme
```

But what if you `stash` multiple times?!

```
~$ git stash list
stash@{0}: WIP on new-feature: 3c927dd Added text to readme
stash@{1}: WIP on new-feature: 3c927dd Added text to readme
stash@{2}: WIP on new-feature: 3c927dd Added text to readme
stash@{3}: WIP on hot-fix-456: a46c33b Replaced old lib with new one
```

NAMING YOUR STASH

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

~$ 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

```
~$ git stash apply
```

```
# On branch new-feature
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   some-unrelated-file.js
```

```
~$ 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

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

Make sure you know which stash you're dropping!

POPPING YOUR STASH

```
~$ git stash pop
```

```
# On branch new-feature
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   some-unrelated-file.js
...
Dropped refs/stash@{0} (a52d9c3ba1121dd94eb3925ba60d3f8ef30540c8)
```

```
~$ git stash list
stash@{0}: On new-feature: trying out some async behavior
stash@{1}: On hot-fix-456: attempt to fix bug #456, but incomplete
```

LOGS

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

THE BASIC LOG

```
~$ git log
commit d42c38691161f42fcc07d806f7df4579e8cd189e
Author: jakerella <jordan@jordankasper.com>
Date:   Sat Jul 5 15:57:36 2014 -0500

Added text to readme

commit b9bdb0261ee9bcaa31d7eb062c0bcafee3e530f0
Author: jakerella <jordan@jordankasper.com>
Date:   Sat Jul 5 15:36:30 2014 -0500

Added documentation
...
```

LOG OPTIONS

```
~$ git log --oneline

d42c386 Added text to readme
b9bdb02 Added documentation
c2cb87e removed unused files from test harness
2e8a119 Merge pull request #186 from jakerella/feature-async-response
79eec03 Reorganized 'Documentation' section and renamed
...
```

LOG GRAPHS

```
~$ git log --oneline --graph

*   714e0dd Merge pull request #220 from appendto/feature-two
|\
| * c8bcfc1 last update for feature two
| *   3e3a8fd Merge branch 'feature-two-tweak' into feature-two
| |\
| | * 46f4cf1 tweak for feature two
| * | 1471f2b one more thing on feature two
| |/
| * 5ce150b feature two initial work
|/
* 062b1f0 some previous work
```

FILTERING THE LOG

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

~$ git log --author="jakerella"

~$ git log --before="2014-07-01"

~$ git log -- src/js/

~$ git log -S foobar
```

POINTING BLAME

(Many times at yourself...)

BLAME GAME

```
/* utility functions */

function sum(x, y) {
    return x * y;
}
```

```
~$ git blame src/js/utilities.js
d35241e6 (Ryan      2014-03-13   1) /* utility functions */
d35241e6 (Ryan      2014-03-13   2)
d35241e6 (Ryan      2014-03-13   3) function sum(x, y) {
27cc7225   (Jordan      2014-03-15     4)     return x * y;
d35241e6 (Ryan      2014-03-13   5) }
```

BLAME GAME

```
~$ git L3,5 blame src/js/utilities.js
d35241e6 (Ryan      2014-03-13   3) function sum(x, y) {
27cc7225 (Jordan    2014-03-15   4)     return x * y;
d35241e6 (Ryan      2014-03-13   5) }
```

DISSECTING THE LOG USING `GIT BISECT`

USING BISECT

First, switch to a broken branch...

```
~$ git bisect start
```

```
~$ git bisect bad
```

```
~$ git bisect good b9bdb02
Bisecting: 3 revisions left to test after this (roughly 2 steps)
[7e3f9393a716b1bf5c935031bae679249352726d] implemented some cool new thing
```

```
~$ git status
# Not currently on any branch.
nothing to commit (working directory clean)
```

USING BISECT

Test the current commit...

```
~$ git bisect good b9bdb02
Bisecting: 3 revisions left to test after this (roughly 2 steps)
[7e3f9393a716b1bf5c935031bae679249352726d] implemented some cool new thing
```

```
~$ grunt test
```

```
~$ git bisect [good|bad]
Bisecting: 1 revision left to test after this (roughly 1 step)
[b86e65d056ea750f1f882df7c10dc5e1bf2deeb2] risky change that may break things
```

USING BISECT

```
~$ git bisect start
~$ git bisect bad
~$ git bisect good b9bdb02
Bisecting: 3 revisions left to test after this (roughly 2 steps)
[7e3f9393a716b1bf5c935031bae679249352726d] implemented some cool new thing

...
```

```
~$ git bisect good
7e3f9393a716b1bf5c935031bae679249352726d is the first bad commit
commit 7e3f9393a716b1bf5c935031bae679249352726d
Author: jakerella
```

NOW YOU KNOW...

Remember, we're no longer on a branch!

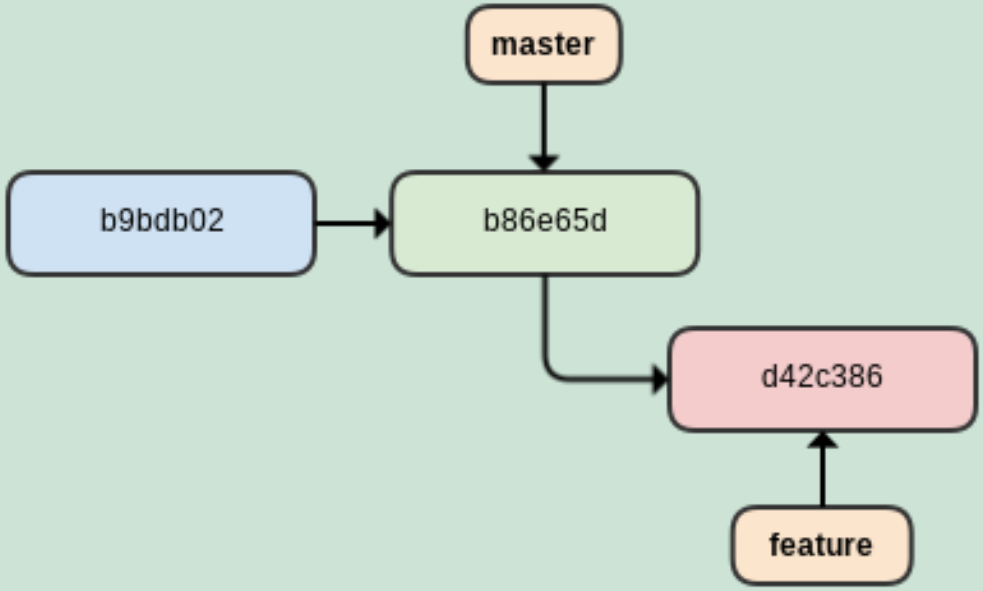
```
~$ git bisect reset
Previous HEAD position was b86e65d... Added some stuff
Switched to branch 'new-feature'
```

PLAYING NICE WITH OTHERS

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

MERGING

With no divergent changes...

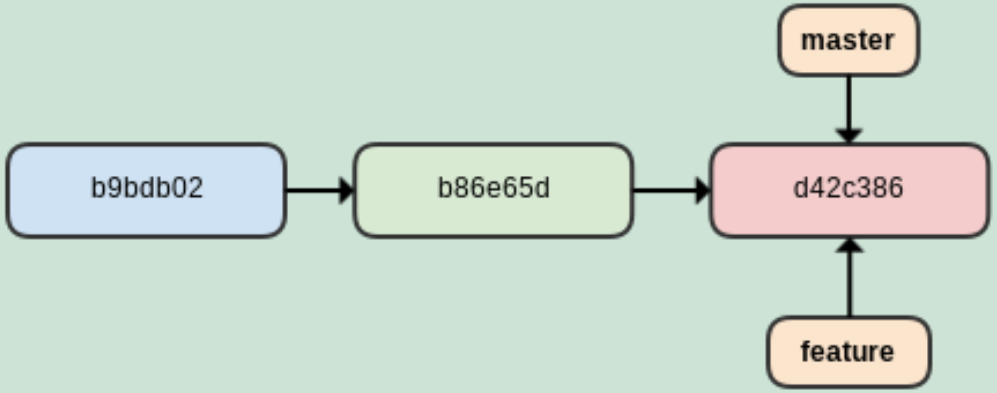


FAST FORWARD

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

```
~$ git checkout master
~$ git merge feature
```

```
Updating b86e65d..d42c386
Fast forward
src/js/some-file.js |   13 ++++--+++
1 files changed, 7 insertions(+), 2 deletions(-)
```



NO FAST FORWARD

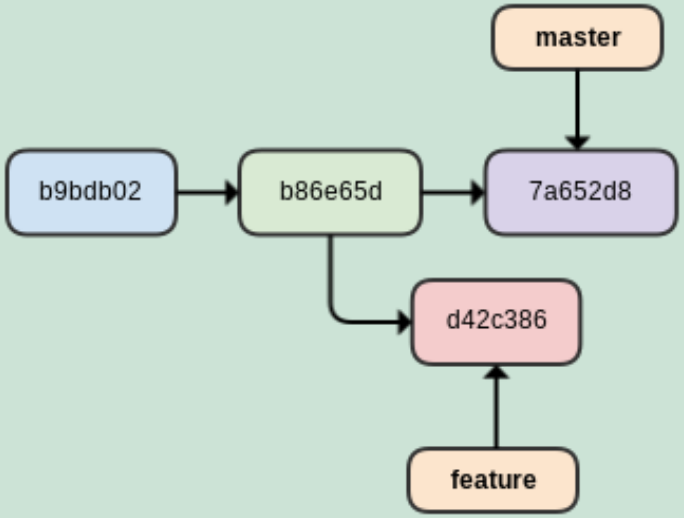
```
~$ git checkout master
~$ git merge feature --no-ff
```

WHY?

Because you lose branch/merge history with fast-forward!

DIVERGENT CHANGES

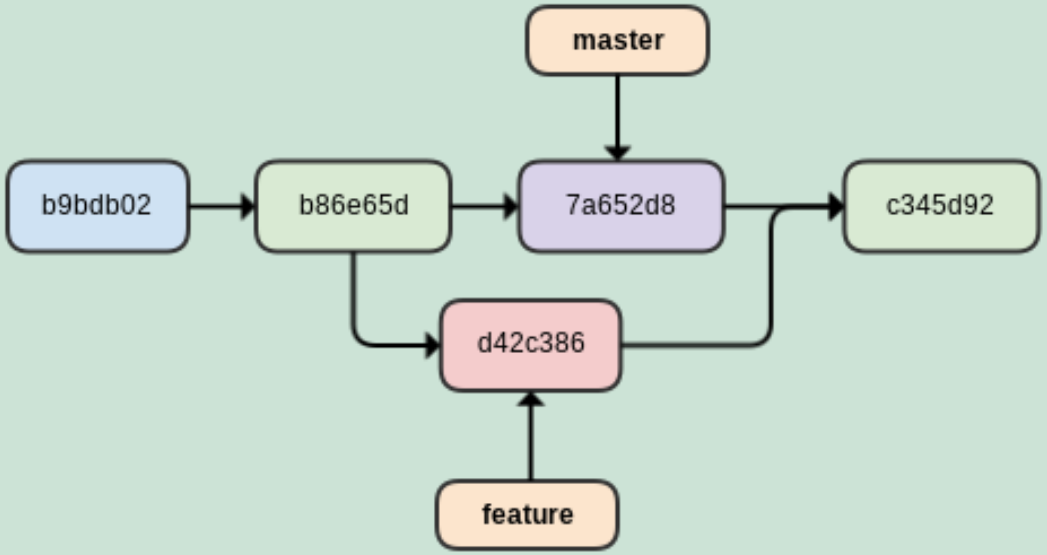
When code in `feature` diverges from the current trunk...




```
~$ git checkout master
~$ git merge feature
```

DIVERGENT CHANGES

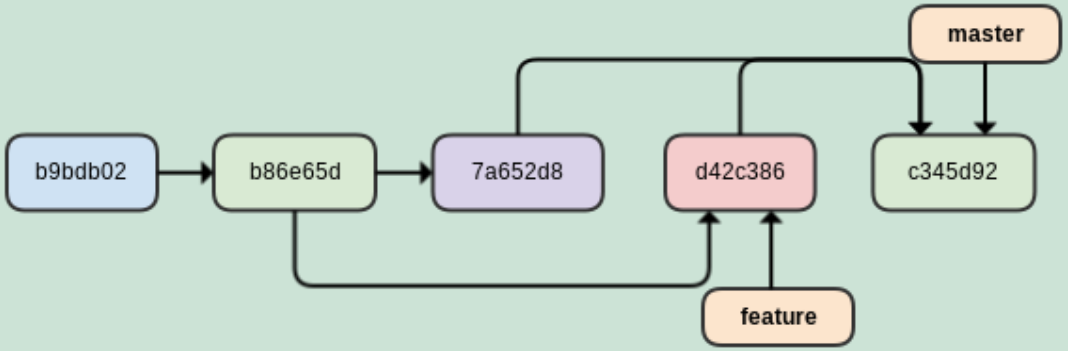
When code in `feature` diverges from the current trunk...




```
~$ git checkout master
~$ git merge feature
```

DIVERGENT CHANGES

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




```
~$ git checkout master
~$ git merge feature
```

MERGE CONFLICTS

CONFLICTS

When git can't determine how to handle multiple changes...

```
~$ git checkout master
~$ git merge feature
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

CONFLICTS

Inside the `README.md` file...

```
...some text common to both branches

<<<<<<< HEAD
text only in master
=======
same line, different text in branch
>>>>>>> feature

more common text...
```

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
   ```

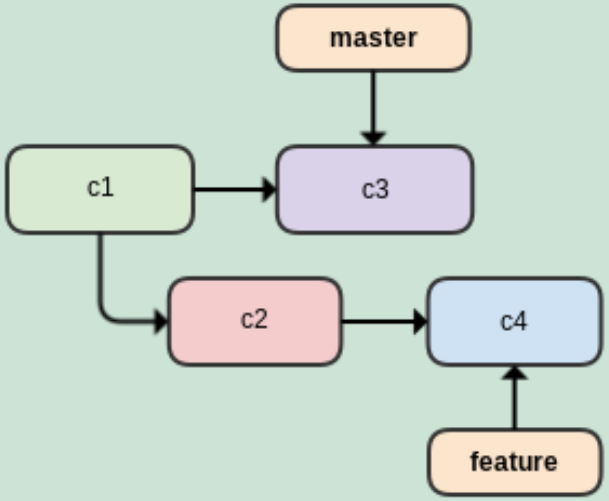
4. Remove the "orig"inal file from the branch

   ```
   ~$ rm README.md.orig
   ```

REBASING

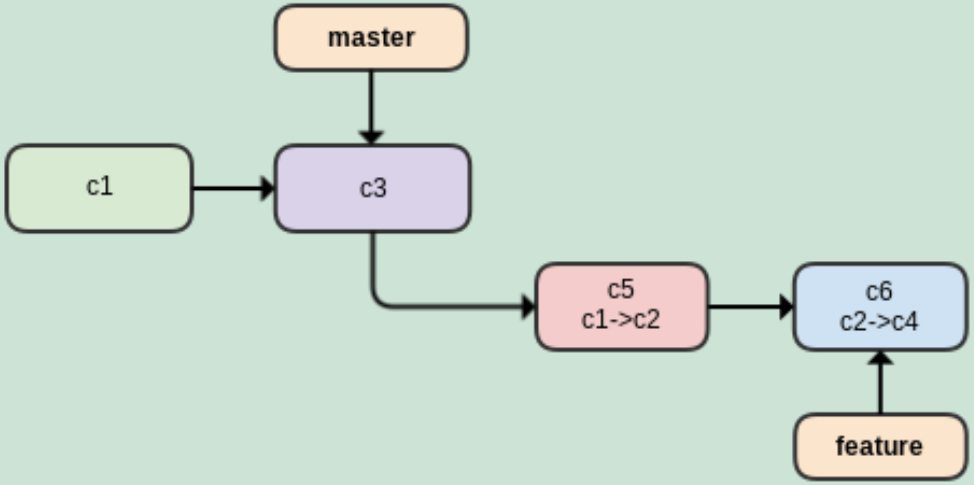
BE CAREFUL rebasing if you have already pushed to a shared branch!

DIVERGENT CHANGES



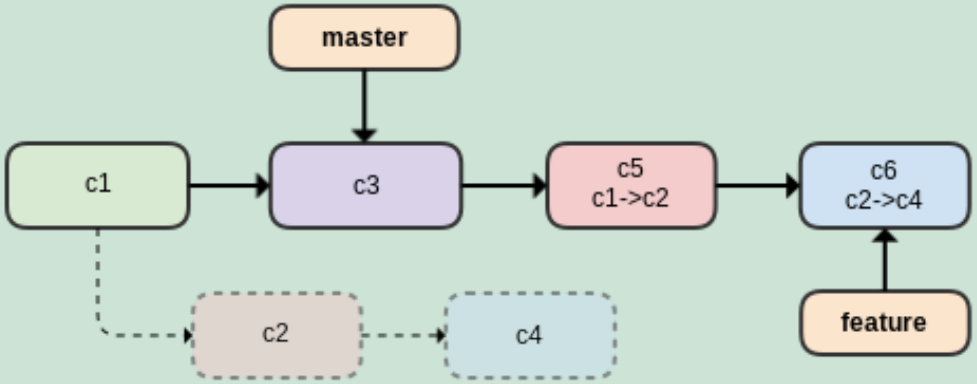

```
~$ git checkout feature
~$ git rebase master
```

REWRITING HISTORY




```
~$ git checkout feature
~$ git rebase master
```

REBASING ON MASTER




```
~$ git checkout feature
~$ git rebase master
```

YOU CAN STILL GET CONFLICTS WITH `REBASE`!

REBASE CONFLICTS

```
~$ git rebase master

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

1. Fix the conflict in the file!
2. Stage the fixed file

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

3. Continue the rebase

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

TOO MUCH CONFLICTION?

You can skip an individual conflict...

(probably not a good idea)

```
~$ git rebase --skip
```

TOO MUCH CONFLICTION?

Or you can abort the entire process...

```
~$ git rebase --abort
```

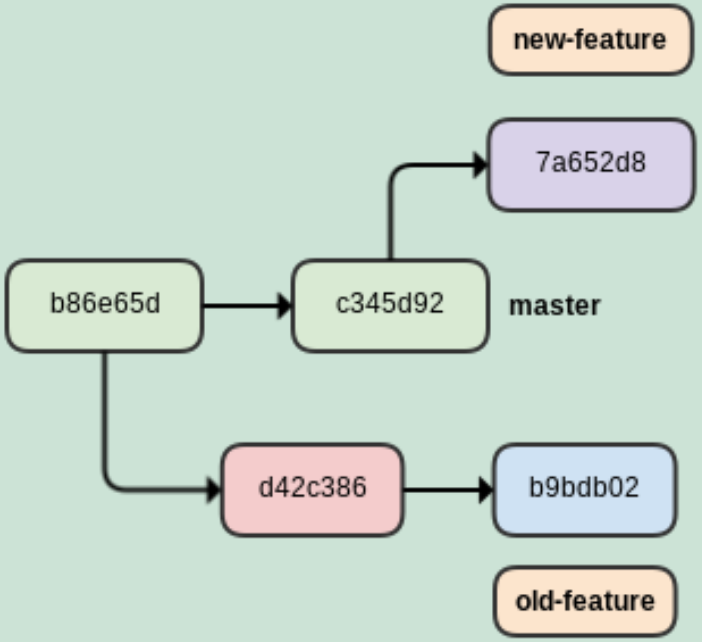
I don't always rebase, but when I do... I do it by default.

```
~$ git pull --rebase
```

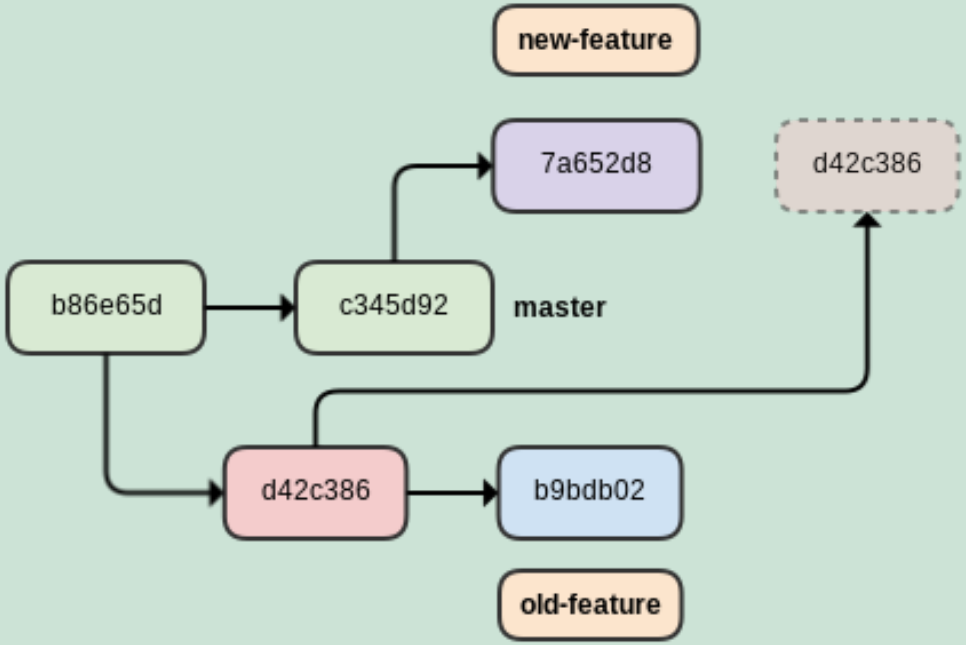
```
~$ git config branch.master.rebase true
~$ git config branch.some-other-branch.rebase true
~$
~$ git config branch.autosetuprebase always
```

CHERRY-PICKING

MULTIPLE BRANCHES

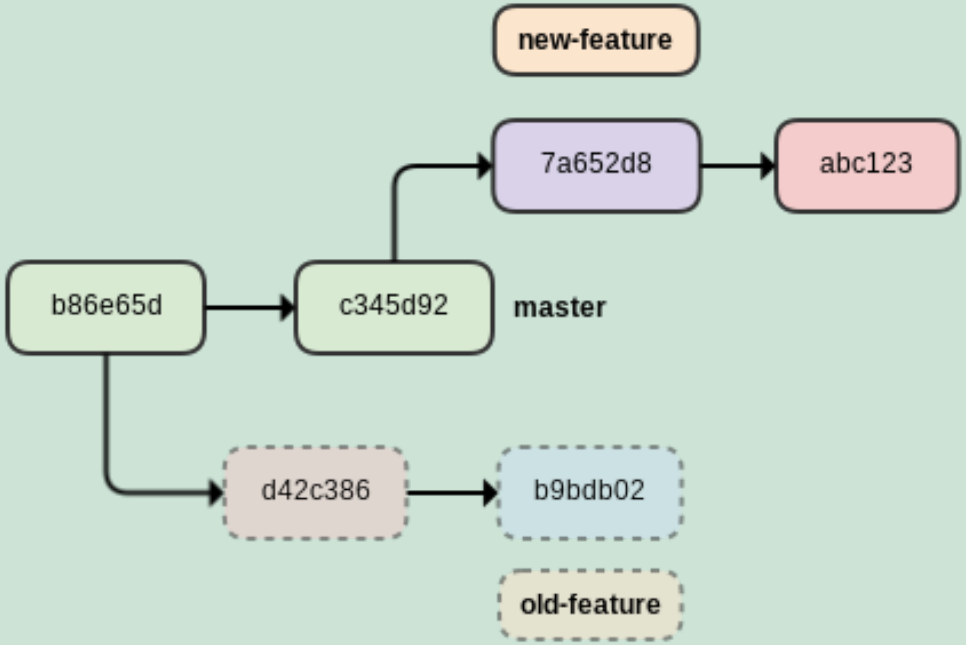


CHERRY PICK




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

CHERRY PICK




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

THANK YOU!

GITTING MORE OUT OF GIT

Jordan Kasper

@jakerella

FLUENTCONF: PLEASE RATE THIS SESSION!!

BIT.LY/GIT-REVIEW