git

it

objectives

By the end of this workshop, you will understand (some of)

what git does for you

behind the scenes

objectives

By the end of this workshop, you will be able to

build commits that represent meaningful changes to the codebase

objectives

By the end of this workshop, you will be able to

make the right decisions when rebasing or merging

objectives

By the end of this workshop, you will be able to

efficiently find the cause of regressions with `bisect`

not covered

- working with remote repositories

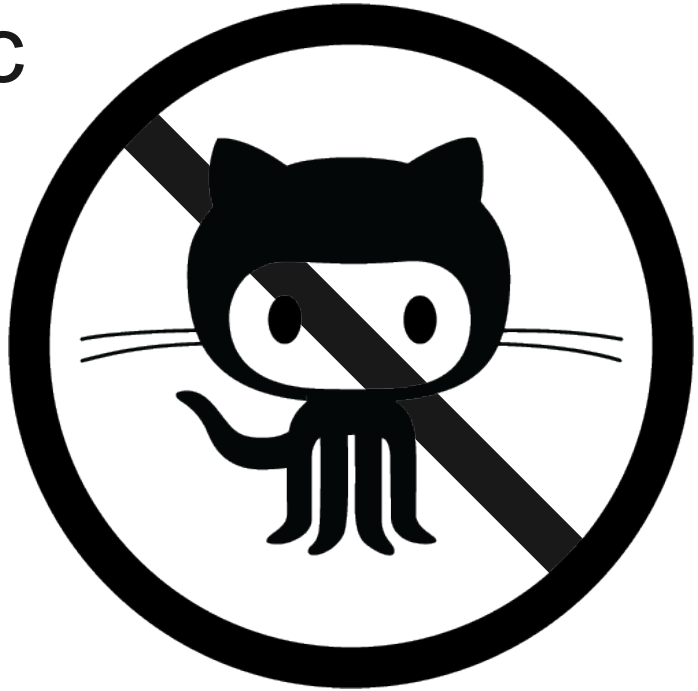
not covered

- git for mercurial / subversion / cvs users

not covered

- anything github-specific

(sorry)



not covered

- *basics* like creating branches, cloning, pushing, pulling...

topics

- the most likely reason we use git
- merging & rebasing
- conflict resolution w/ diff3
- finding the cause of bugs with bisect
- cool git tips

topics

- the most likely reason we use git
- merging & rebasing
- conflict resolution w/ diff3
- finding the cause of bugs with bisect
- cool git tips

why has git taken over open source?

- branches are first-class objects
- distributed nature lends itself well to forking
- free
- really really fast

why is git so fast?

- ~~branches are first-class objects~~
- ~~distributed nature lends itself well to forking~~
- ~~free~~
- really really fast – *people use fast things*

(often at the sacrifice of usability)

why is git so fast?

git stores and manages the data that you give it in a unique and efficient way

why is git so fast?

But first, some backg

The old-school way:

CVS, subversion, me source dungeon, etc, all fell down this trap:

Differences between en very small

therefore…

Only store the changes between revisions

“What a great idea!”

(Dick Gruene, creator of CVS)

storing diffs per-r

Storing diffs between revisions has *got* to be optimal ...after all, diffs are usually pretty small compared to the entire codebase.

SHUT THE F@#$ UP!
You've got it all wrong! Crawl back into your hole in the ground you ***caveman***!

storing diffs per-revision

While diffing between revisions is efficient on disk, it is **not efficient** for doing most of the important things that a good version control system should do.

Switching between branches takes a big performance hit because of this.

an example

What follows is an example of why storing diffs per revision storage scheme is inefficient when switching between branches.

not so fast… switching

When you do this:

`svn switch myBranch`

svn, cvs, hg, etc… do this:

finds the common ancestor of `myBranch` and your current branch, walks down to from your current branch, then walks to `myBranch`, recording the changes along the way, then applies those changes to your current working directory. – *O(n)*

When you do this:

`git checkout myBranch`

git does this:

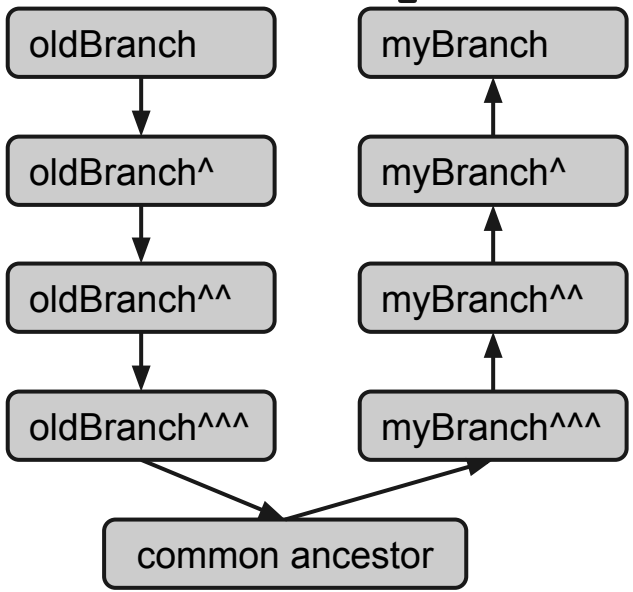
finds files **that already exist** in .git/objects that are in myBranch and uses them to replace files in your current working directory. – *O(1)*

way faster! awesome!

fast switching!

When you do this:

`svn switch myBranch`



When you do this:

`git checkout myBranch`

in other words:

git doesn't need to care about where your revision came from in order to switch to it.

how does git pull this off?

git does not need to walk the history graph of your repository to apply changes.

git organizes the files in your repository so it will have near-immediate access to every revision of every file.

how does git pull this off?

Files are keyed by their SHA-1 hash in `.git/objects`

Commits as stored by git (indirectly) store the path and SHA-1 hash of each file in your repository.

what about space?

That is very wasteful of disk space; I'd never design a version control system that worked like that!

what about space?

It turns out that with some clever pattern matching, all of your repository's data can be stored as efficiently *without* basing the diffs on the history graph.

what about space?

git uses a structure called a “packfile” to efficiently store similar files.

packfiles are stored in `.git/objects` alongside all the other files in your repository, but you don’t (usually) interact with them directly.

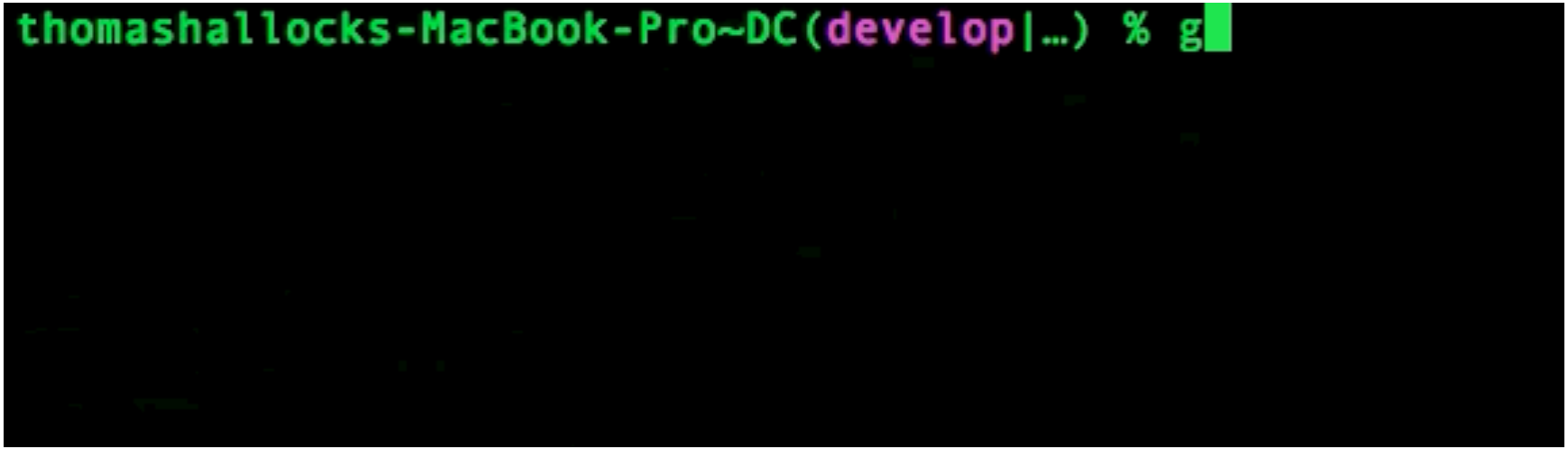
packfiles!

packfiles are usually created when git does its garbage collection.

You've probably noticed garbage collection happening occasionally when you `git pull` or `git fetch` (or run `git gc` manually.)

packfiles!

packfile creation and optimization is one of the things that is happening when you see this:



packfiles!

Have a look at the “internals” chapter of the git book. It explains in very clear language how git keeps track of absolutely everything:

https://git-scm.com/book/en/Git-Internals

ok, git is fast. now what?

git’s speed opens up a slew of great features:

- rebasing
- branches as first-class objects
- commit --amend
- its distributed nature
- quickly comparing files across arbitrary revisions

ok, git is fast. now what?

git's speed opens up a slew of great features:

- rebasing
- branches as first-class objects
- commit --amend
- its distributed nature
- quickly comparing files across arbitrary revisions

rebasing

At the very least, you’re probably familiar with the term “rebasing.”

Most of you have been using it in some capacity:

- git pull --rebase
- git rebase origin/master

rebasing

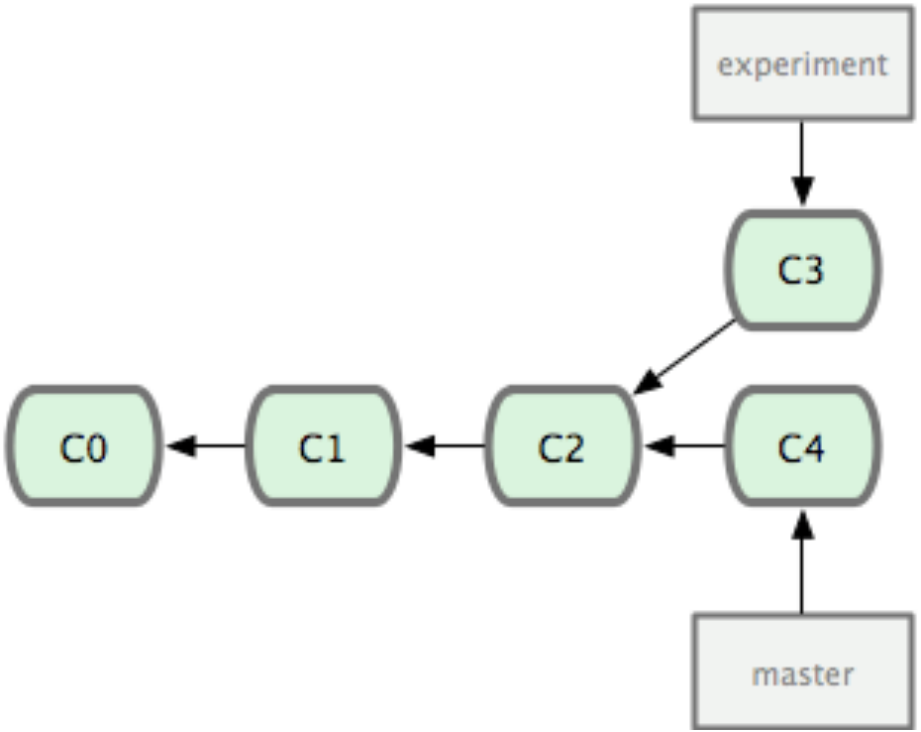
...but what exactly is “rebasing”?

re – change

base – a structure on which something draws or depends

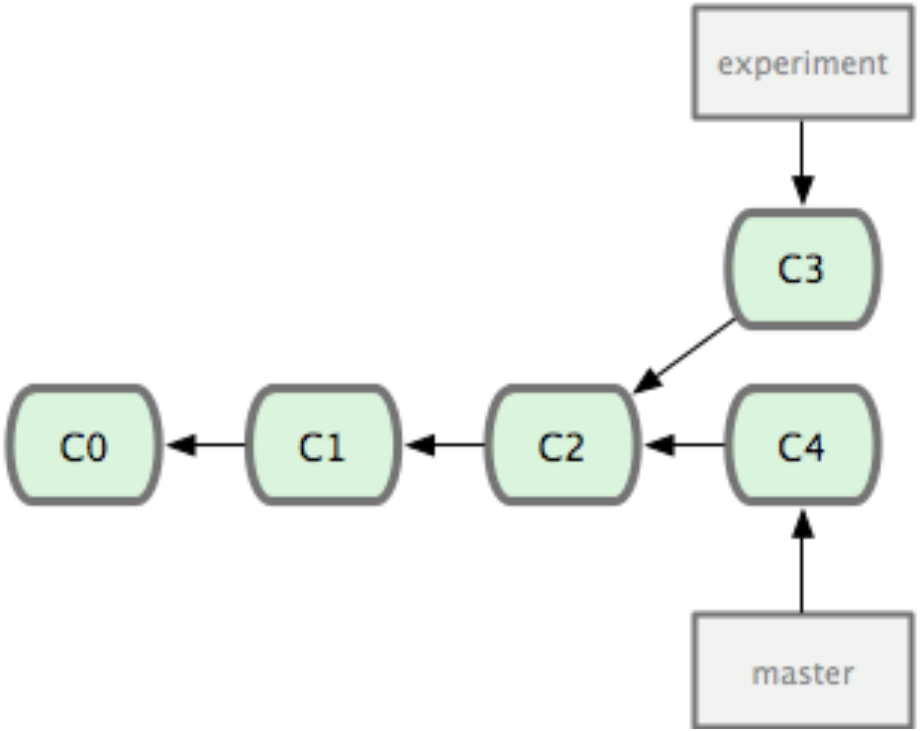
When you rebase, you are changing the parent commit of your branch. In order to accommodate this change, all the commits after the parent commit must be reconstructed.

rebasing (and merging)



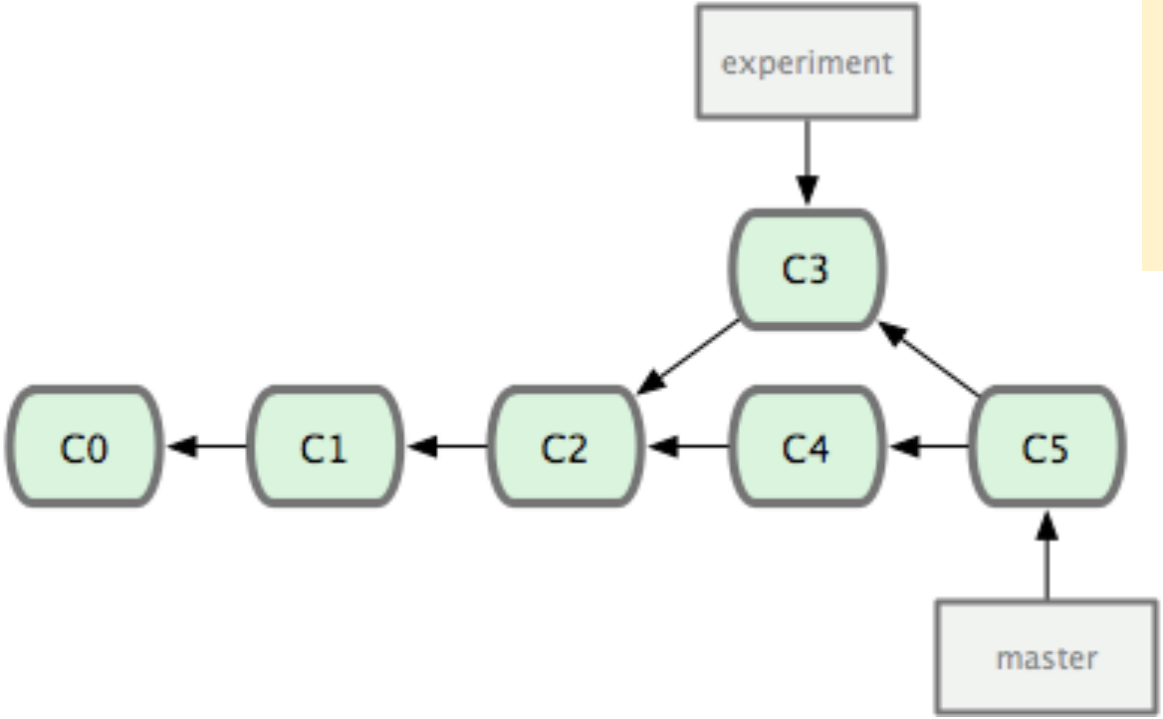
Consider this situation

rebasing (and merging)



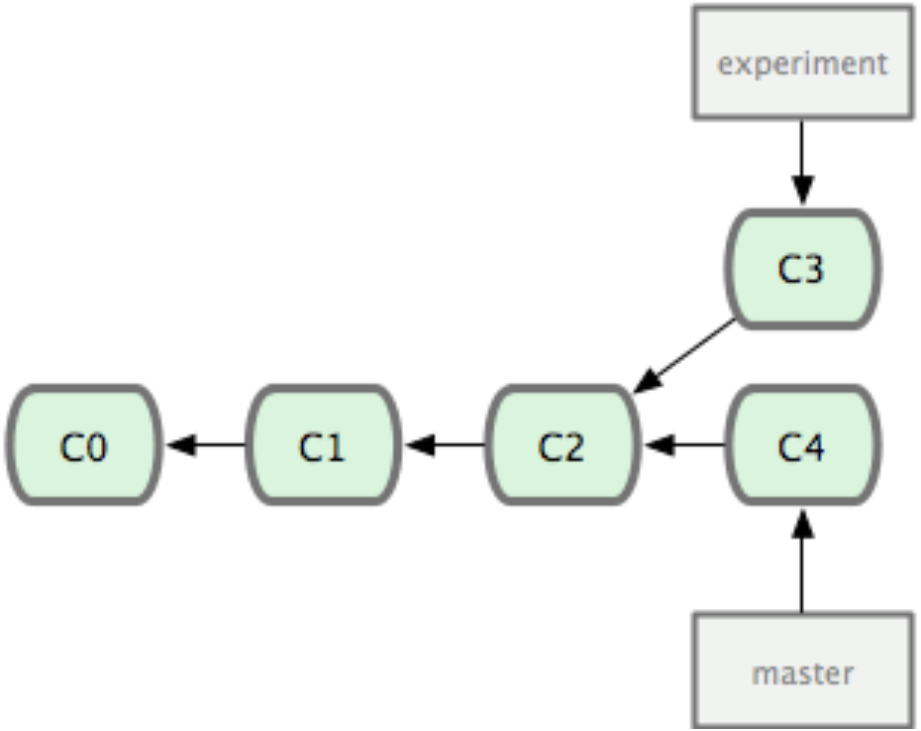
The easiest way to integrate `experiment` into `master` is to use the `merge` command. Merge performs a three-way merge between the two latest branch snapshots (C3 and C4) and the most recent common ancestor of the two (C2), creating a new snapshot (and commit)

rebasing (and merging)



C5 is the merge commit

rebasing (and merging)

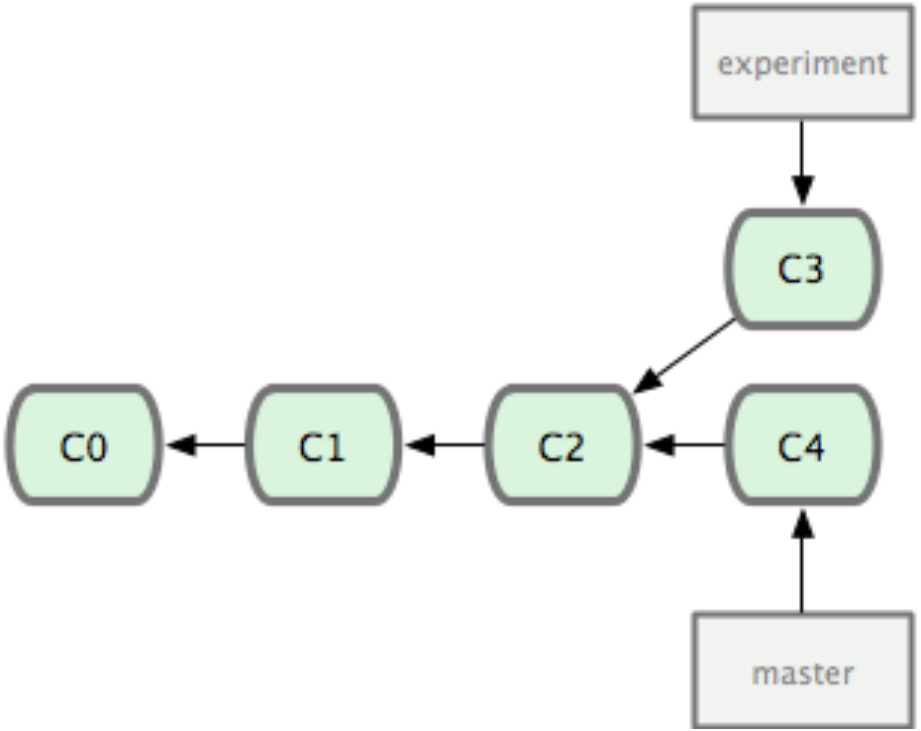


However, there is another way:

You can take the *patch* of the change that was introduced in C3 and re-apply it *on top of* C4. In Git, this is called *rebasing*.

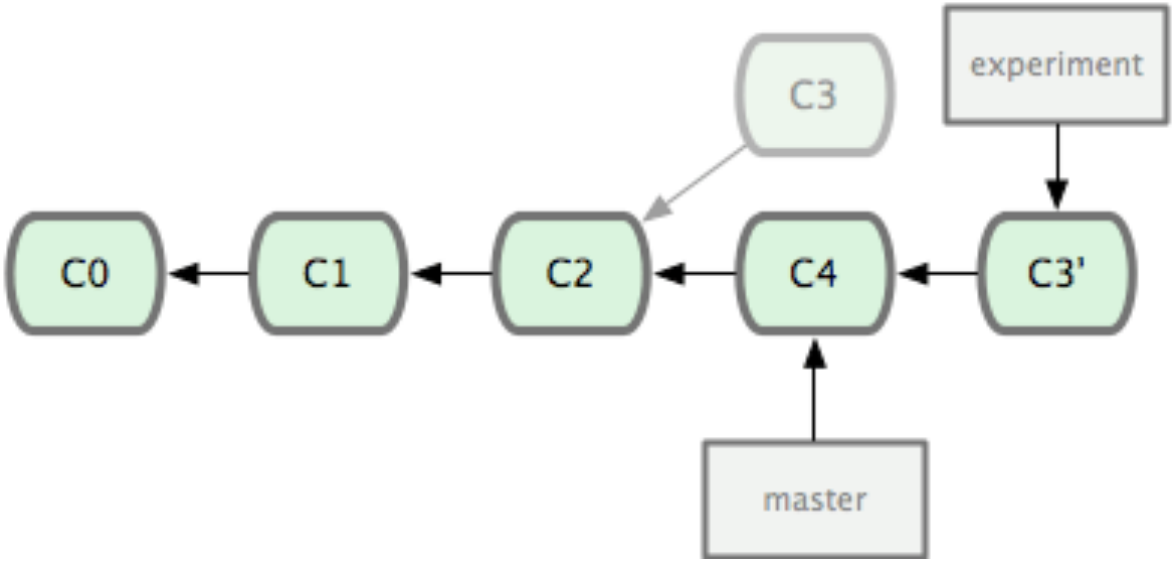
With the `rebase` command, git will take all unique changes on one branch and replay them on another one.

rebasing (and merging)



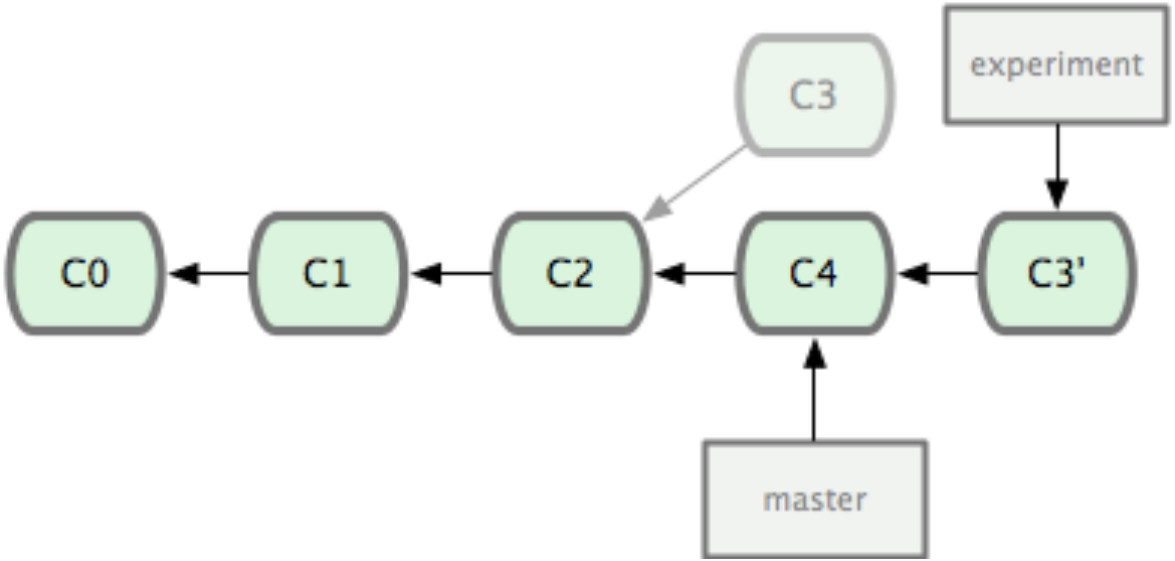

```
$ git checkout experiment

$ git rebase master

First, rewinding head to
replay your work on top
of it...
Applying: added staged
command
```

rebasing (and merging)



There is no difference between the code in the rebased version and the code in the merged version in the prior slide.

rebasing (and merging)



However, the history is cleaner in the rebased version because it does not contain a merge commit.

changing the parent branch

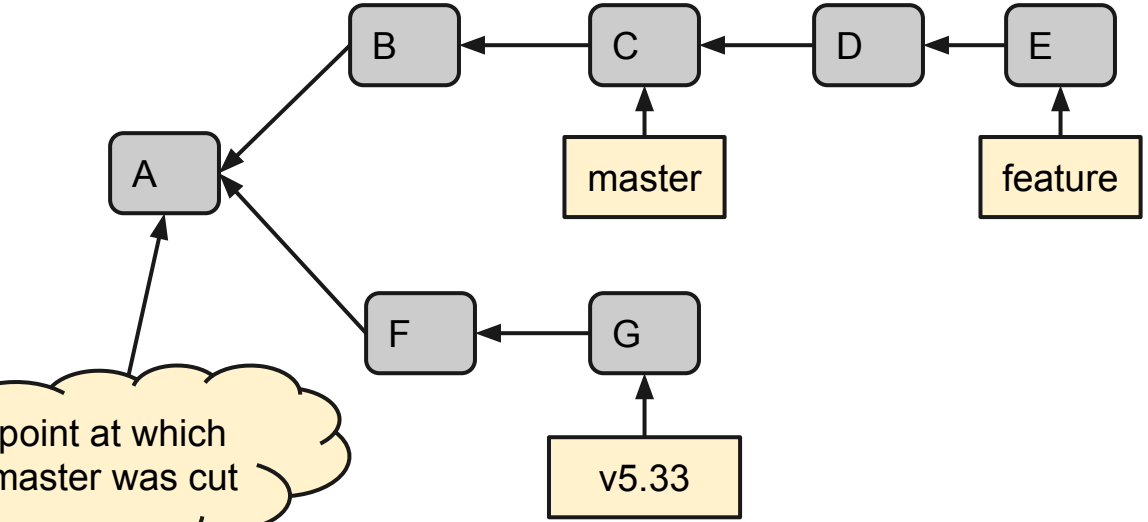
Rebasing is great for producing clean history, but it can also take your branch places that a merge can not.

Change the parent of your branch like this:

```
git rebase --onto <new parent> <old parent>
```

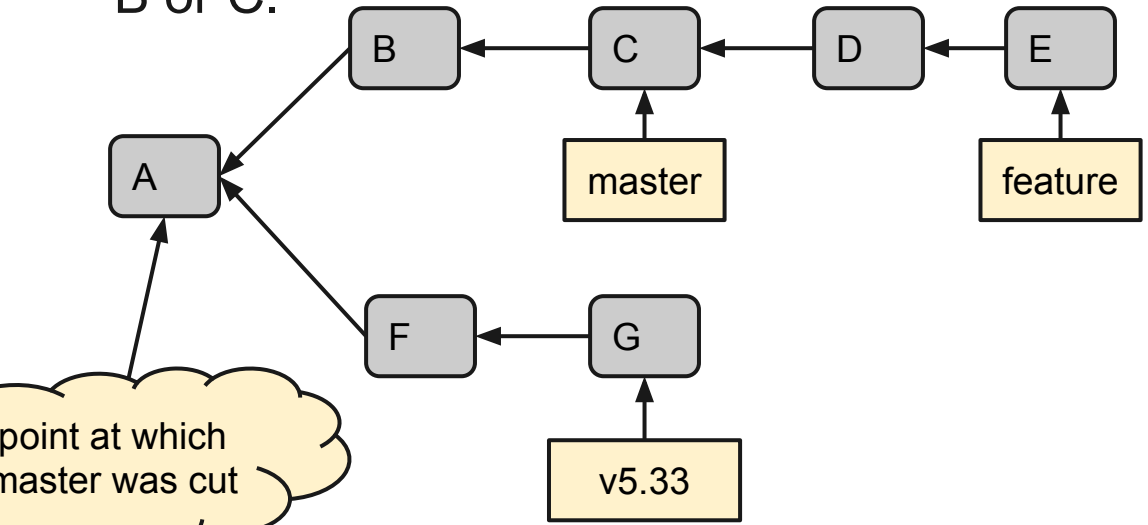
moving to the release branch

Here's a relevant scenario for us at 1stdibs. Consider that you've been keeping your `feature` branch up-to-date with `master` on bunsen:



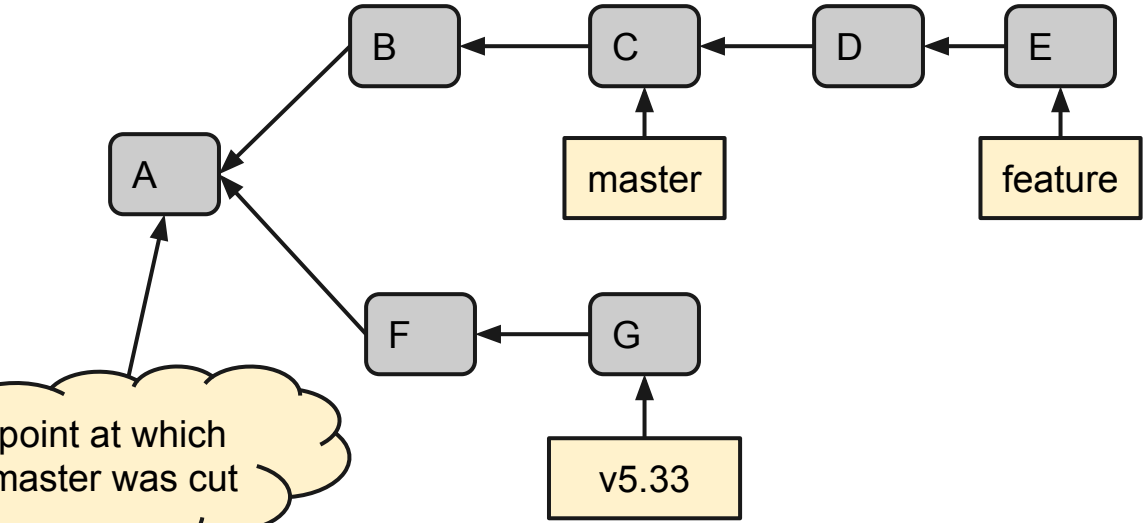
moving to the release branch

You let it slip that you've finished `feature` and product decides to release it early, so now just the commits unique to `feature` have to be included in `v5.33`. In the graph below, that would be D and E, but *not* B or C.



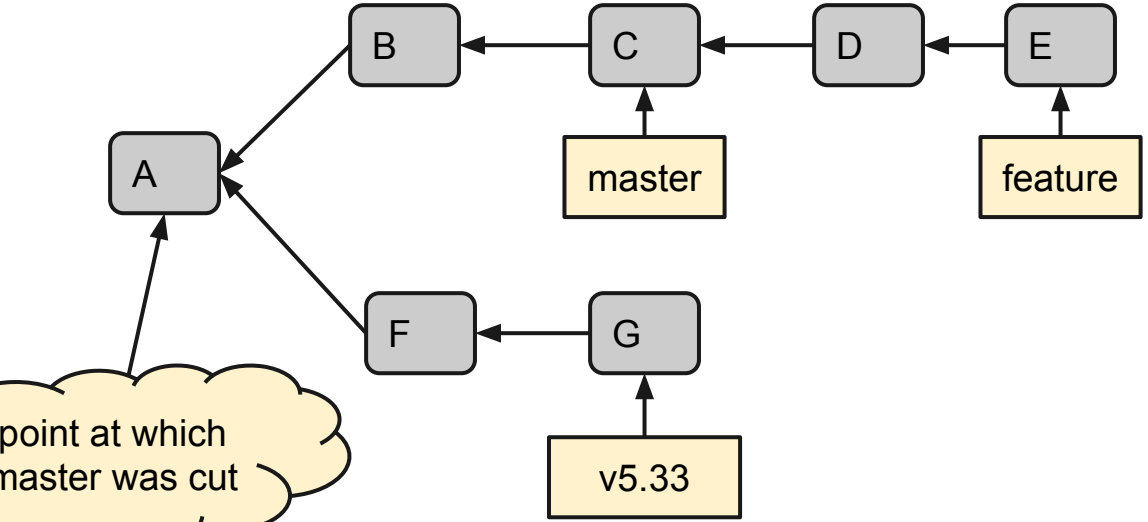
moving to the release branch

You can’t just merge `feature` as it stands into `v5.33` because doing so would also include commits B and C, which aren’t scheduled for `v5.33`.



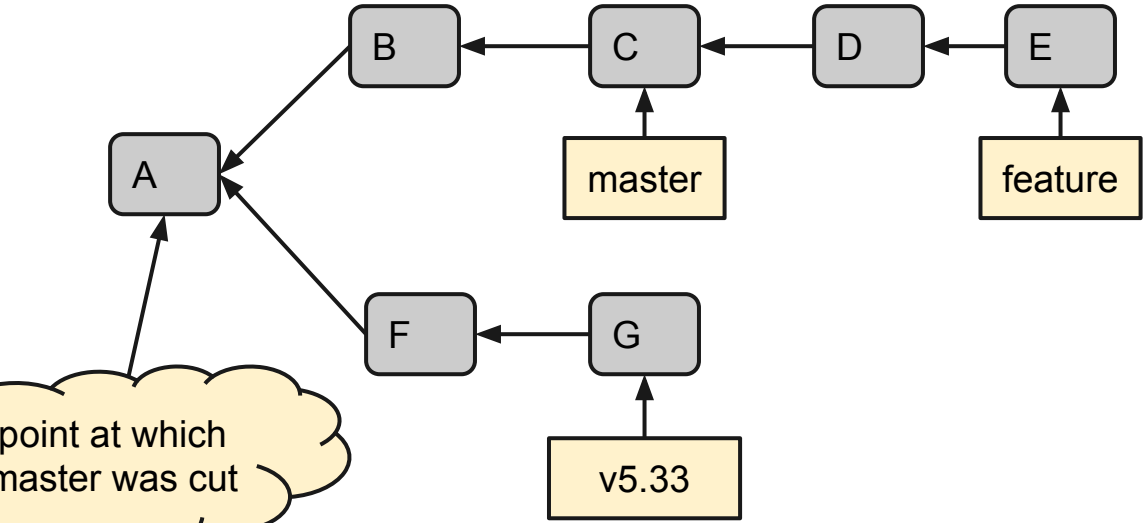
moving to the release branch

So, you just want to get D and E into v5.33. In a simple scenario like this, you could get away with cherry-picking D and E into a new branch of `v5.33`, but if `feature` had 30 unique commits, you'd have a lot of repetitive work to do.



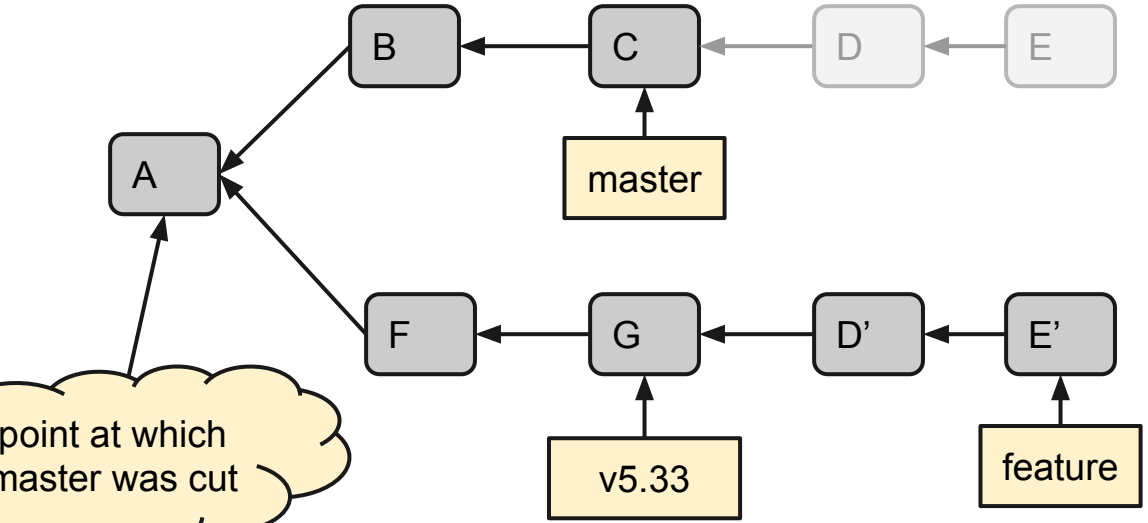
moving to the release branch

```
git rebase --onto v5.33          master
```



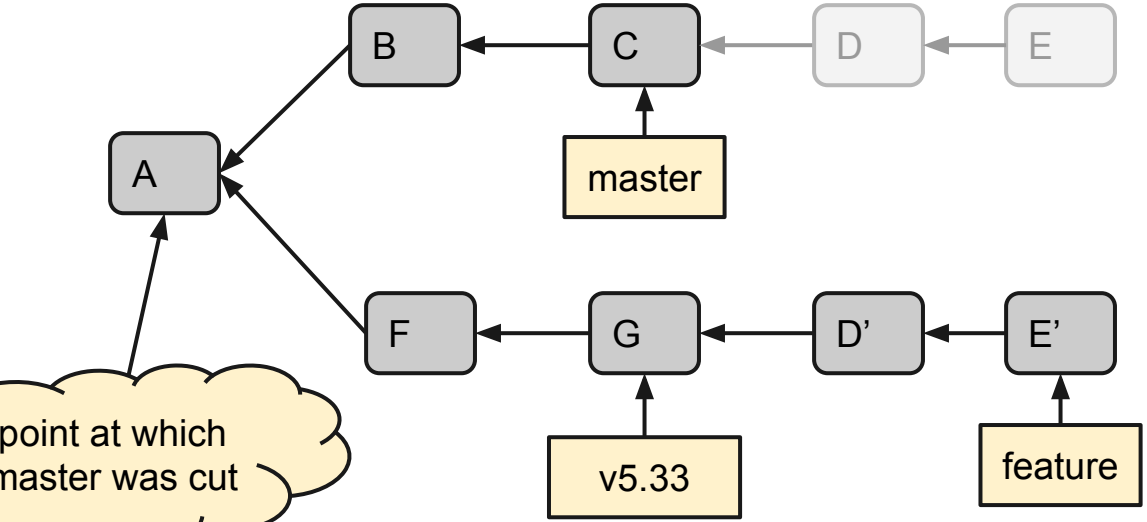
moving to the release branch

```
git rebase --onto v5.33          master
```



don’t rebase shared branches

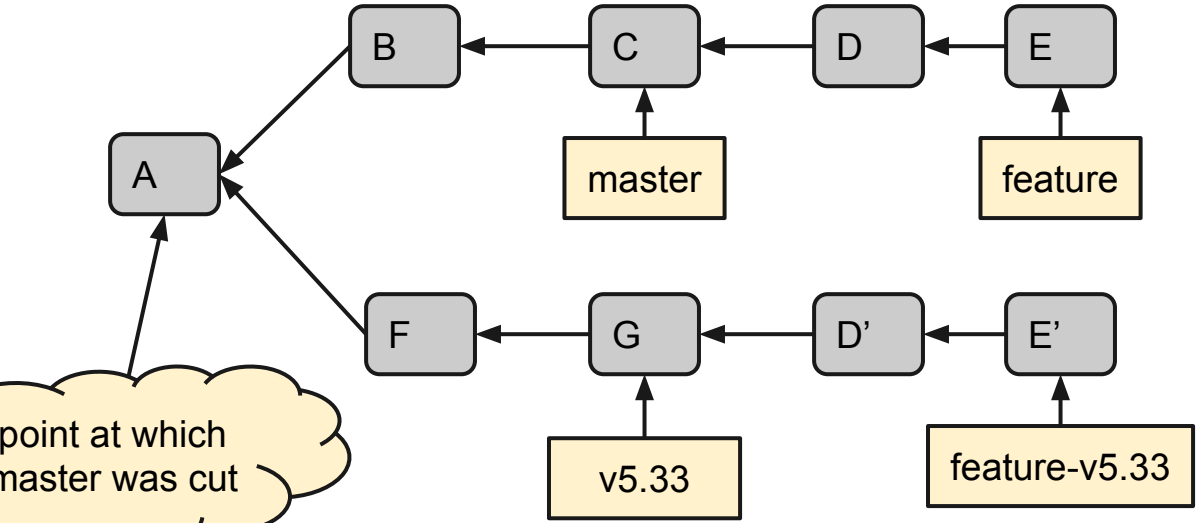
If you’re sharing this branch with someone else, do not push your rebased branch to your remote under the same name.



don’t rebase shared branches

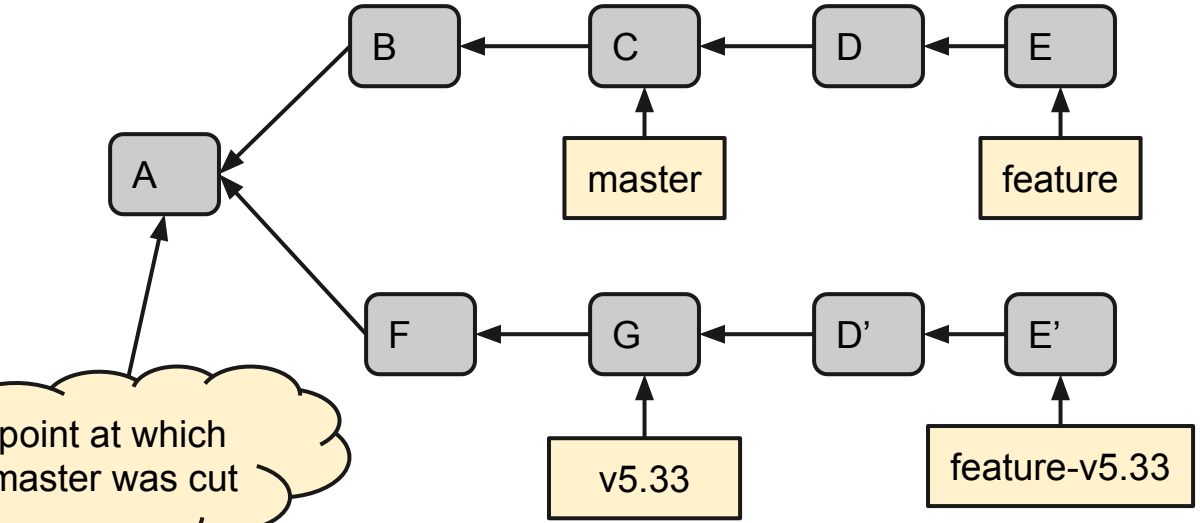
Instead, rename your rebased branch before you push it:

```
git branch -m feature-v5.33
```



moving to the release branch

When your coworker is good and ready, they can move their commits from `feature` to `feature-v5.33` in the same way that you moved `feature`'s commits from `master` onto `v5.33` with `rebase --onto`



the perils of rebasing

The git book presents a common scenario that arises when multiple people work off rebased branches:

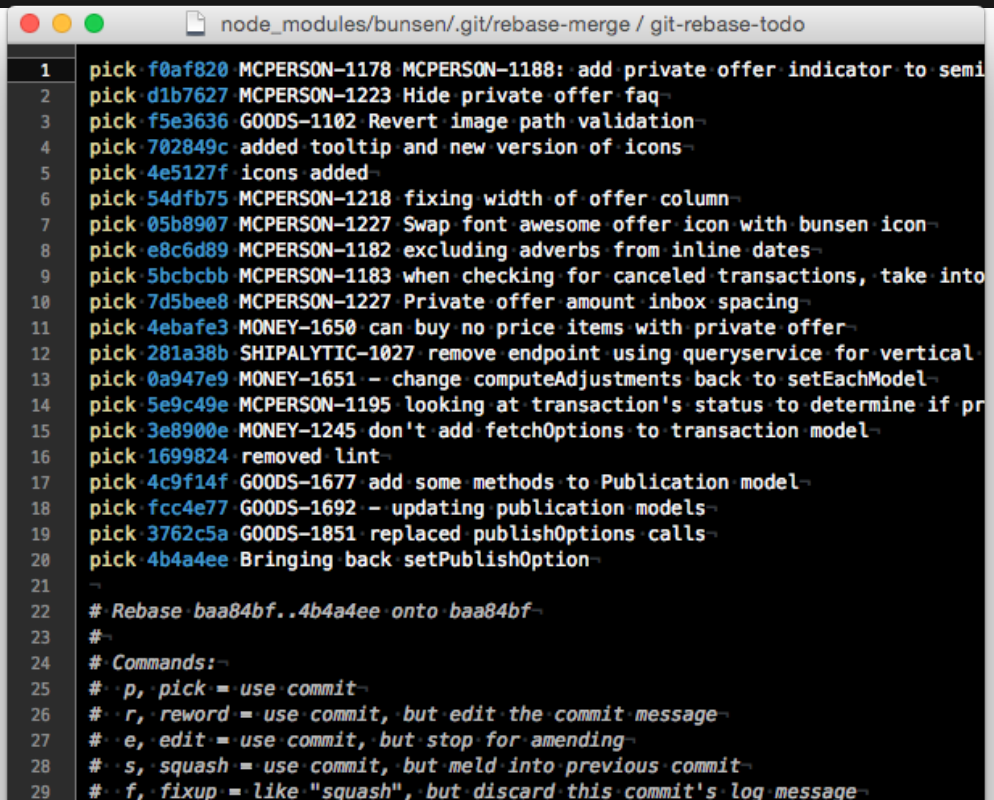
http://git-scm.com/book/en/v2/Git-Branching-Rebasing#The-Perils-of-Rebasing

git rebase -i <parent>

run `rebase` with the `-i` flag and git will show you in your editor the commits that it plans to rebase.

You can edit this plan...

git rebase -i <parent>



You can remove, reorder or even add more commits to this list.

You can also meld a commit with the one above it by changing the `pick` to `fixup`.

If you want to keep the commit message of both messages, use `squash` instead.

git rebase -i <parent>

...live demo in progress...

recommended git configs

Have a look at my .gitconfig:

github.com/antialias/dotfiles/blob/master/.gitconfig

.gitconfig

```
[rebase]
  autosquash = true
```

`--autosquash` by default when doing interactive rebase

.gitconfig

```
[rerere]
  enabled = true
```

or just run this command right now (I'll wait)

```
git config --global rerere.enabled true
```

.gitconfig

```
[rerere]
  enabled = true
```

reuse ***re***corded ***re***solution of conflicted merges

Git will remember how you resolved every conflict, and will apply that same resolution if it sees the same conflict in the future.

.gitconfig

```
[alias]
    lg = log --color --graph --pretty=format:'%Cred%h%
Creset -%C(yellow)%d%Creset %s %Cgreen(%cr)%C(bold blue)<%
an>%Creset' --abbrev-commit
```

aliases git lg to show a really nicely formatted log instead of the overly-verbose one that git shows by default.

.gitconfig

```
thomashallocks-MacBook-Pro~B(demo-fixup|✔) % git lg -30
* 4b4a4ee - (HEAD, tom/demo-fixup, demo-fixup) Bringing back setPublishOption (38 minutes ago)<prestifidibs>
* 3762c5a - GOODS-1851 replaced publishOptions calls (38 minutes ago)<Rich>
* fcc4e77 - GOODS-1692 - updating publication models (38 minutes ago)<prestifidibs>
* 4c9f14f - GOODS-1677 add some methods to Publication model (38 minutes ago)<donfusilli>
* 1699824 - removed lint (38 minutes ago)<Thomas Hallock>
* 3e8900e - MONEY-1245 don't add fetchOptions to transaction model (38 minutes ago)<Augusto Corvalan>
* 5e9c49e - MCPERSON-1195 looking at transaction's status to determine if private offer transaction show be consider
* 0a947e9 - MONEY-1651 - change computeAdjustments back to setEachModel (4 hours ago)<Rob Richard>
* 281a38b - SHIPALYTIC-1027 remove endpoint using queryservice for vertical categories (4 hours ago)<Joseph Ahn>
* 4ebafe3 - MONEY-1650 can buy no price items with private offer (4 hours ago)<Rob Richard>
* 7d5bee8 - MCPERSON-1227 Private offer amount inbox spacing (4 hours ago)<Peter Jodogne>
* 5bcbcbb - MCPERSON-1183 when checking for canceled transactions, take into account case where item was marked sold
r items sold through checkout (4 hours ago)<Brett Ashford>
* e8c6d89 - MCPERSON-1182 excluding adverbs from inline dates (4 hours ago)<Thomas Hallock>
* 05b8907 - MCPERSON-1227 Swap font awesome offer icon with bunsen icon (4 hours ago)<Peter Jodogne>
* 54dfb75 - MCPERSON-1218 fixing width of offer column (4 hours ago)<Thomas Hallock>
* 4e5127f - icons added (4 hours ago)<Liam Dickson>
* 702849c - (refs/bisect/good-702849ca1563a2d38683e6df6c6b88d0083de3fc) added tooltip and new version of icons (4 ho
* f5e3636 - GOODS-1102 Revert image path validation (4 hours ago)<takeWarning>
* d1b7627 - MCPERSON-1223 Hide private offer faq (4 hours ago)<Peter Jodogne>
* f0af820 - MCPERSON-1178 MCPERSON-1188: add private offer indicator to semi-collapsed message view and apply inacti
* baa84bf - MONEY-1633 buyer offer takes a financial amount (4 hours ago)<Rob Richard>
* 5183f40 - MCPERSON-1155 move normalizeCurrencyFormat() into bunsen/numberConversions.js so it can be used in admi
* f0f8f73 - MCPERSON-1220 showing actual status for private offer headsup when we don't have any predefined logic. (
* d0afbf0 - GOODS-1572 upcoming arrow should have white fill, not default black fill (4 hours ago)<donfusilli>
* b3356f3 - MONEY-1245 added disabled-action color var (4 hours ago)<Augusto Corvalan>
```

.gitconfig

```
thomashallocks-MacBook-Pro~B(master|✔) % git lg -30
*   954e251 - (HEAD, origin/master, origin/HEAD, master) Merge pull request #1112 from jakemccloskey/mixin-helper (31 minutes ago)<dale tan>
|\
| * 6aebd27 - Added mixin helper for ES2015 classes (8 days ago)<Jake McCloskey>
* | 6412847 - Bringing back setPublishOption (24 hours ago)<prestifidibs>
* |   f4f317e - Merge pull request #1110 from RichLandy/feature-publishing-statuses (25 hours ago)<prestifidibs>
|\ \
| * | 10139fa - GOODS-1851 replaced publishOptions calls (25 hours ago)<Rich>
* | |   f5d73bf - Merge pull request #1141 from prestifidibs/feature-listings (25 hours ago)<prestifidibs>
|\ \ \
| |/ /
|/| |
| * | f34fc62 - GOODS-1692 - updating publication models (26 hours ago)<prestifidibs>
|/ /
* |   68d0b9f - Merge pull request #1127 from donfusilli/feature-publication-model-additions (28 hours ago)<prestifidibs>
|\ \
| * | 1cafdd3 - GOODS-1677 add some methods to Publication model (28 hours ago)<donfusilli>
|/ /
* | b66cba3 - removed lint (4 days ago)<Thomas Hallock>
* |   6a2226d - Merge branch 'v5.30' of https://github.com/1stdibs/bunsen (4 days ago)<Rob Richard>
|\ \
| * \   7fef954 - (origin/v5.30, origin/integration-private-offer-status, bre/integration-private-offer-status, v5.30, integration-private-offe
ider-status-awaiting-manual-review (5 days ago)<brettjashford>
| |\ \
| | * | 15dc972 - (tom/bugfix-consider-status-awaiting-manual-review, bugfix-consider-status-awaiting-manual-review) MCPERSON-1195 looking at t
 show be considered 'accepted' (5 days ago)<Thomas Hallock>
| | * | bc387a1 - fixup! MCPERSON-1220 showing actual status for private offer headsup when we don't have any predefined logic. (5 days ago)<Th
| |/ /
* | |   8234fbd - Merge pull request #1126 from augustocorvalan/feature-saved-address (5 days ago)<Rob Richard>
|\ \ \
| * | | ab307cb - MONEY-1245 don't add fetchOptions to transaction model (5 days ago)<Augusto Corvalan>
```

.gitconfig

```
[merge]
  conflictstyle = diff3
```

shows the common ancestor between the “ours” and “theirs” blocks of a conflict.

.gitconfig

```
cauliflower
<<<<<<< HEAD
peas
potatoes
||||||| merged common ancestors
peas
=======
>>>>>>> topic
tomatoes
```

.gitconfig

```
cauliflower
<<<<<<< HEAD
peas
potatoes
=======
>>>>>>> topic
tomatoes
```

.gitconfig

```
[merge]
  conflictstyle = diff3
```

PHPStorm doesn’t show the common ancestor block, so you’ll have to resolve the conflict directly in your source file. Learn to read it.

.gitconfig

```
[merge]
  conflictstyle = diff3
```

More information here:

http://psung.blogspot.com/2011/02/reducing-merge-headaches-git-meets.html

end of slides - *demo time!*

bisect

John Rodriguez, Sr. Front-End Engineer

will now demonstrate how to find the cause of regressions by using a clever git tool called

`bisect`

caution: extra slides ahead

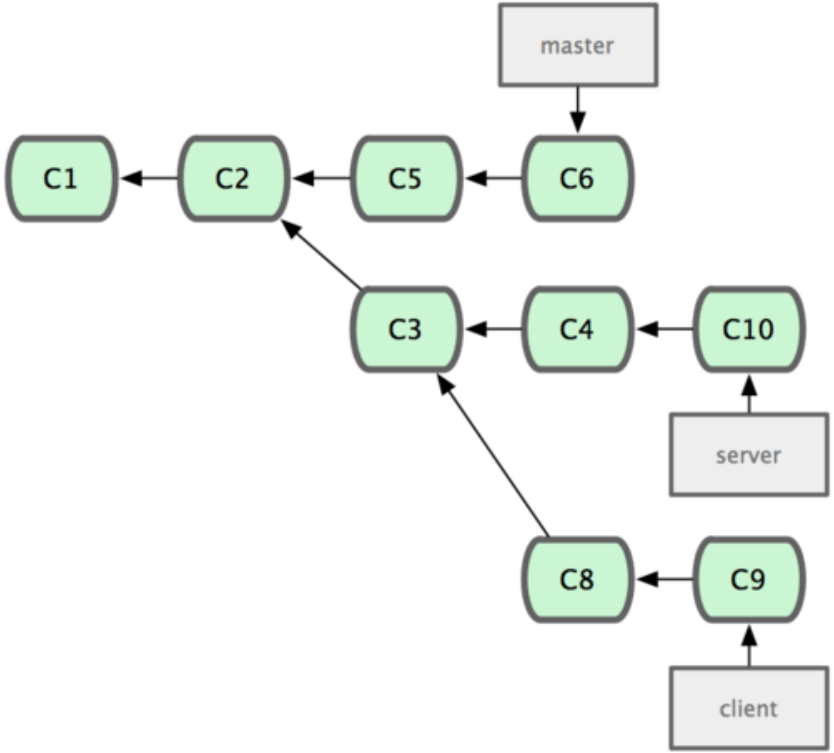
boredom surely awaits. you have been warned

packfiles!

git's algorithm for creating a packfile goes something like this:

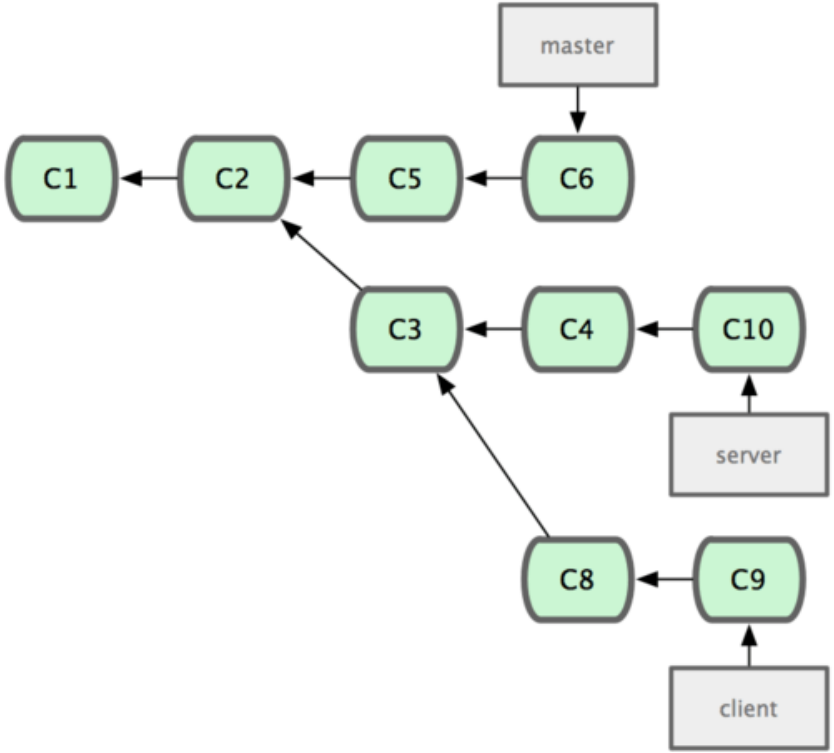
1. find similar files in `.git/objects`
2. store the differences between them in a special file
3. remove all but the base file from which the differences are computed.
4. list the files and the diff files in the packfile, with the order of the files indicating how the files relate to each other.

changing the parent branch



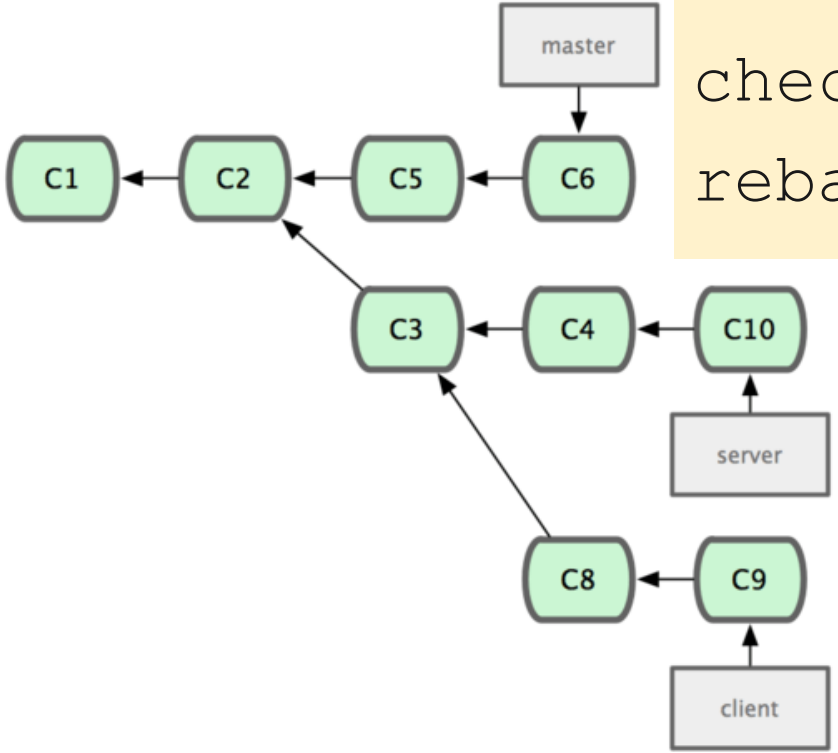
Suppose that the code for `client` is ready to be integrated with `master`, but the code for `server` is not.

changing the parent branch



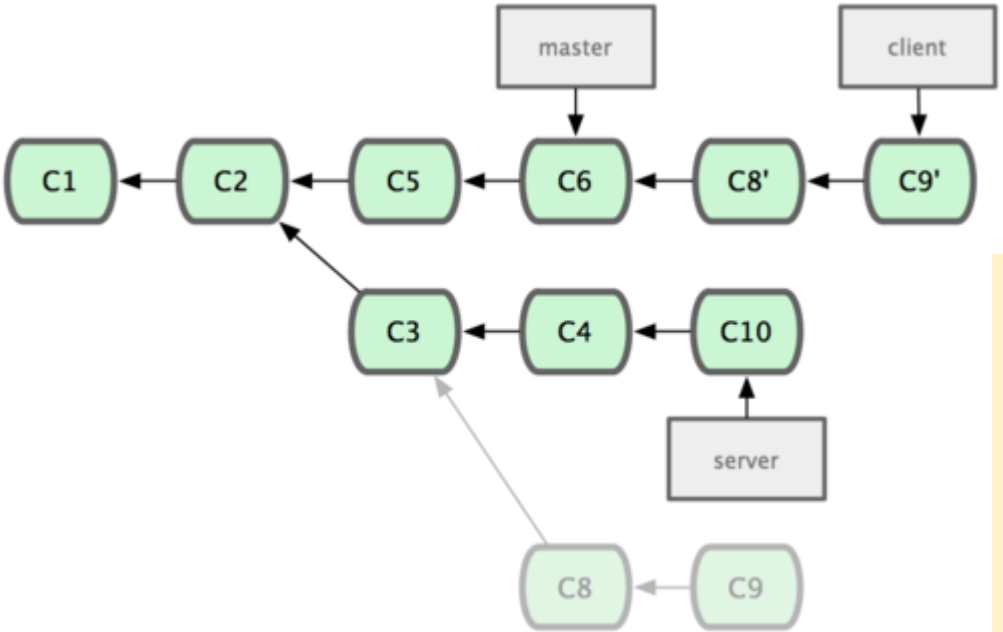
`rebase --onto` can be used to integrate the code for `client` into `master` while excluding the code for `server`.

changing the parent branch




```
checkout client
rebase --onto master server
```

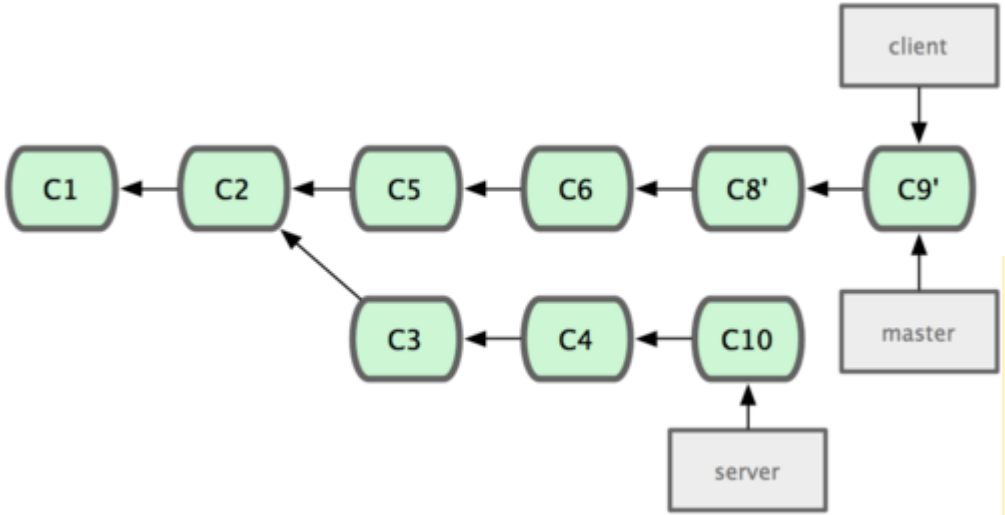
time for a fast-forward



Now we can fast-forward master up to client with

```
checkout master
merge client
```

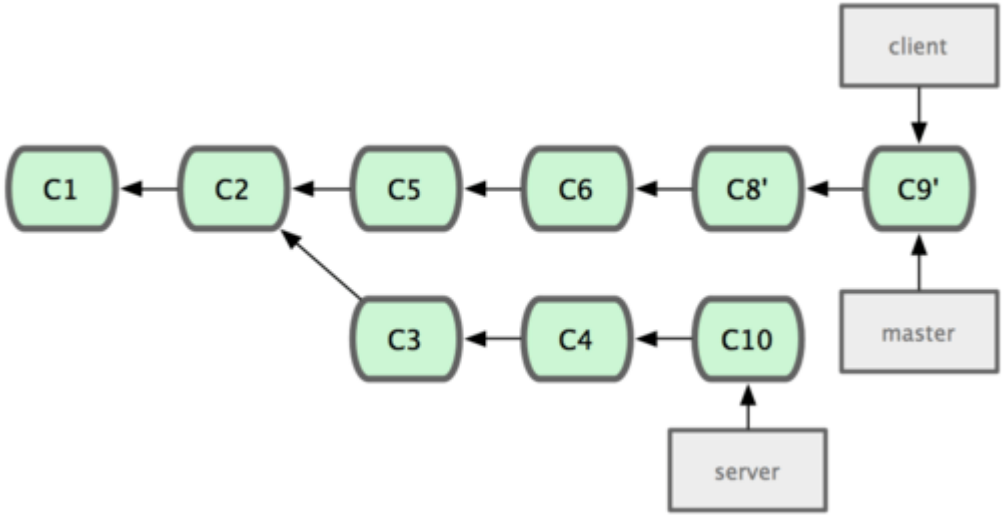
time for a fast-forward



Now we can fast-forward master up to client with

```
checkout master
merge client
```

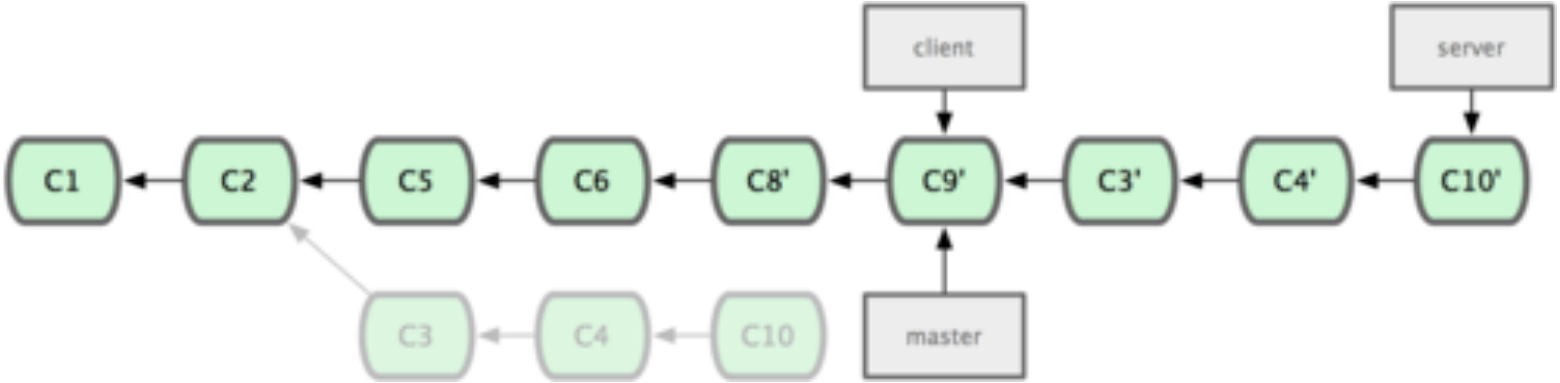
going all the way



Suppose `server` is now ready to be integrated into master. We can rebase it on top of `master` like this:

```
checkout server
rebase master
```

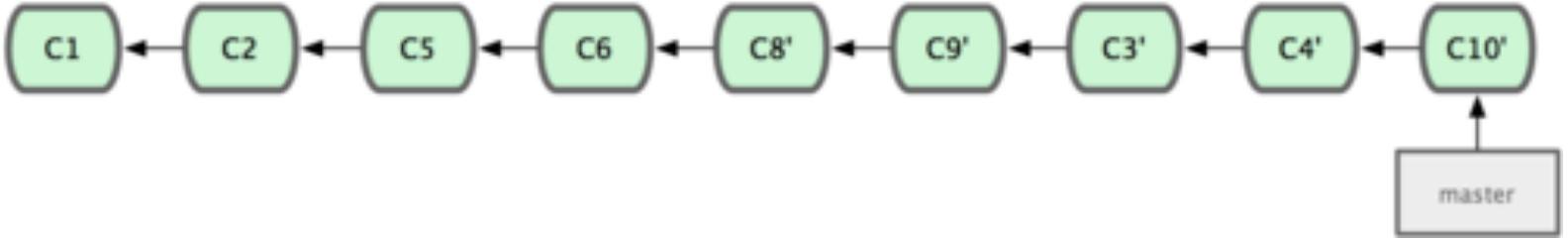
completely linear history



Now, we can merge server into master with:

```
git checkout master
git merge server
```

completely linear history



Again, git figures out that no merge commit is needed so it does a fast-forward merge of `server` into `master`.