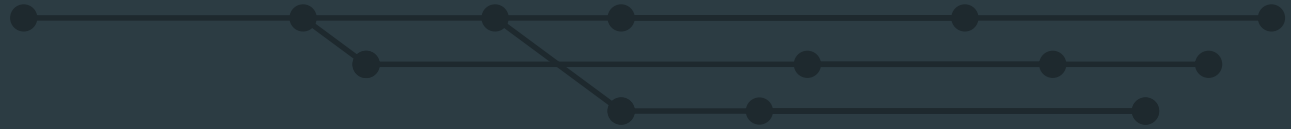
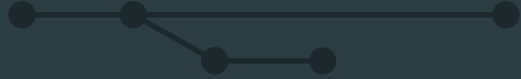




How Git Works

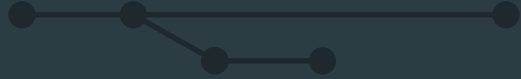


The .git folder

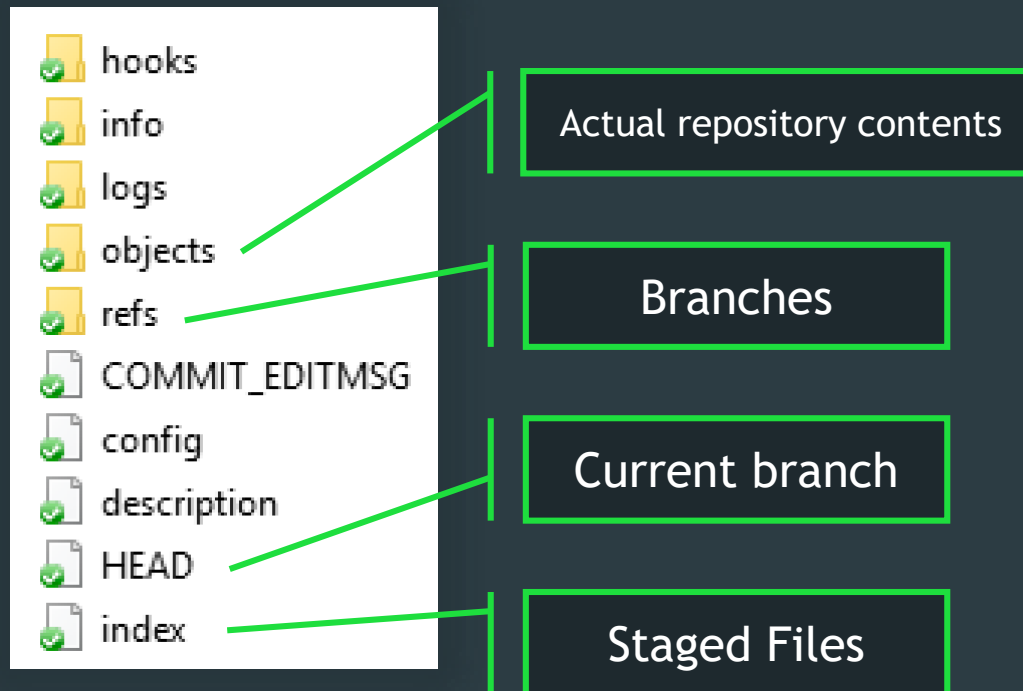


- Contains all of git's "state"
 - File contents
 - Staged files
 - Commit structure
 - Configuration options
 - Current branch
 - Branches/tags
 - etc.

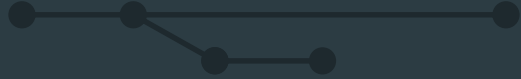
The .git folder



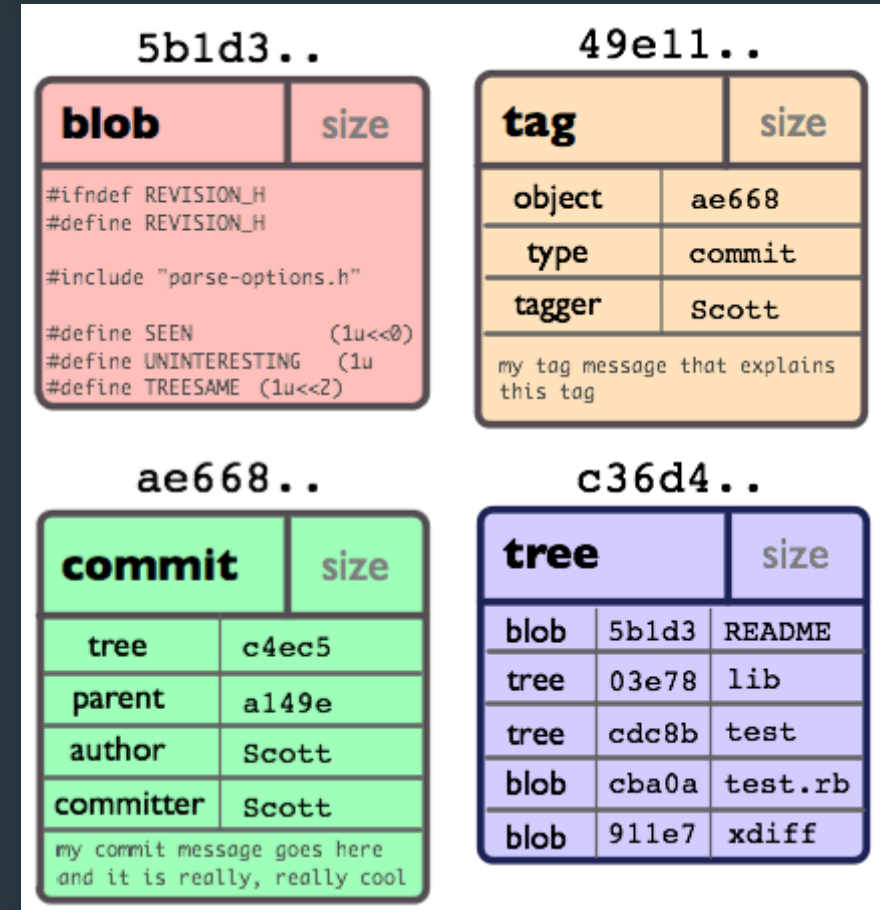
- Contains all of git's "state"



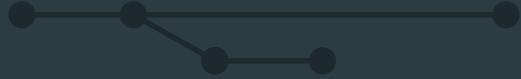
objects directory



- Repository contents are stored as objects
- Object types:
 - blob
 - tree
 - commit
 - tag



objects directory



- Objects are referred to by hash
- Print objects with:
`git cat-file -p [hash]`
- Print tree objects with:
`git ls-tree [hash]`
- Git commit hash with:
`git rev-parse [branch name]`

5b1d3..

blob	size
#ifndef REVISION_H #define REVISION_H #include "parse-options.h" #define SEEN (1u<<0) #define UNINTERESTING (1u #define TREESAME (1u<<2)	

49e11..

tag	size
object	ae668
type	commit
tagger	Scott
my tag message that explains this tag	

ae668..

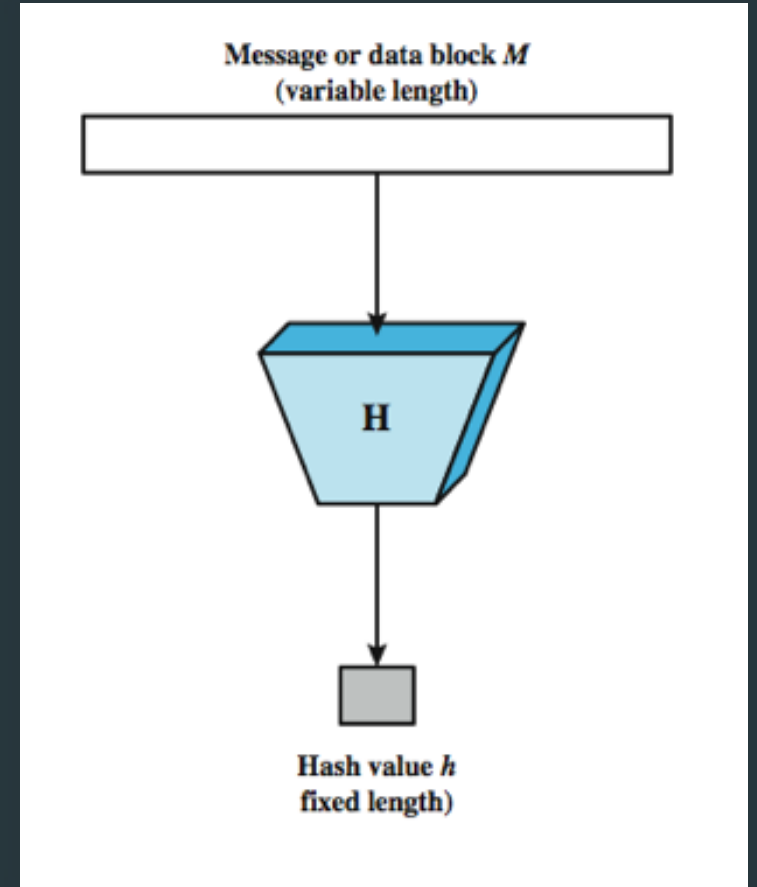
commit	size
tree	c4ec5
parent	a149e
author	Scott
committer	Scott
my commit message goes here and it is really, really cool	

c36d4..

tree	size	
blob	5b1d3	README
tree	03e78	lib
tree	cdc8b	test
blob	cba0a	test.rb
blob	911e7	xdiff

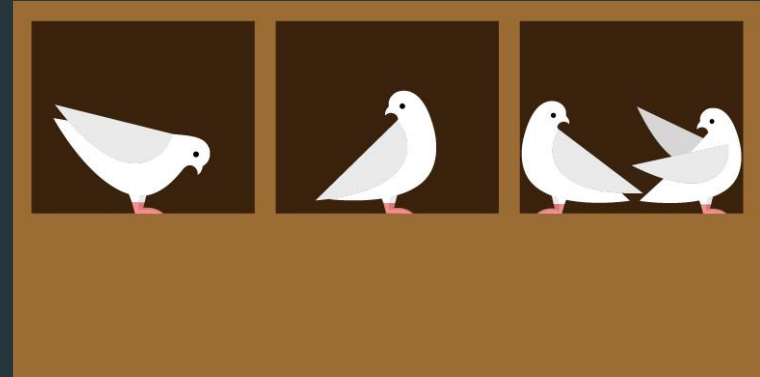
Hash functions

- A hash function maps data of arbitrary size to a "hash" of fixed size
- Ideally, similar inputs are mapped to very different outputs
- Git uses SHA-1
 - 160 bits (40 characters in hex)
- Objects folder maps hashes to data





Hash collisions

- What if two files hash to the same value?
 - There are more than 10^{48} unique hashes
 - But there are an infinite number of files...
- SHA-1 is a cryptographic hash meaning no one knows how to reverse it
 - But Google did generate a collision



SHattered

The first concrete collision attack against SHA-1
<https://shattered.io>





Marc Stevens
Pierre Karpman

Elie Bursztein
Ange Albertini
Yarik Markov

SHattered

The first concrete collision attack against SHA-1
<https://shattered.io>



Marc Stevens
Pierre Karpman

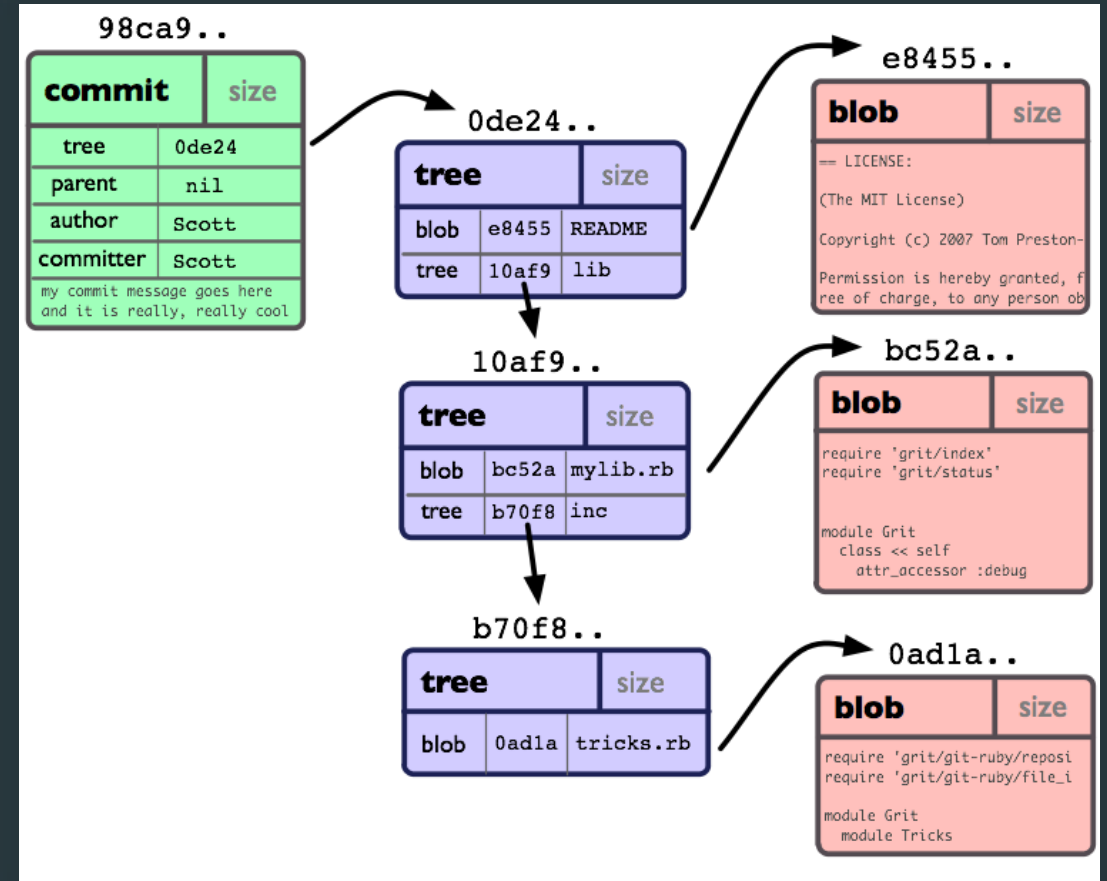
Elie Bursztein
Ange Albertini
Yarik Markov

```
sha1sum *.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a 1.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a 2.pdf
/tmp/sha1
sha256sum *.pdf
2bb787a73e37352f92383abe7e2902936d1059ad9f1ba6daaa9c1e58ee6970d0 1.pdf
d4488775d29bdef7993367d541064dbdda50d383f89f0aa13a6ff2e0894ba5ff 2.pdf
```

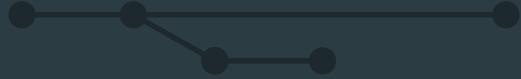
0.64G 8-11h

Using hashes to save space

- Each commit points to a tree object and a list of parent commits
- If two subtrees or blobs are shared by a commit, they only need to be defined once
- When diffing two commits, git can skip a subtree if the hashes match



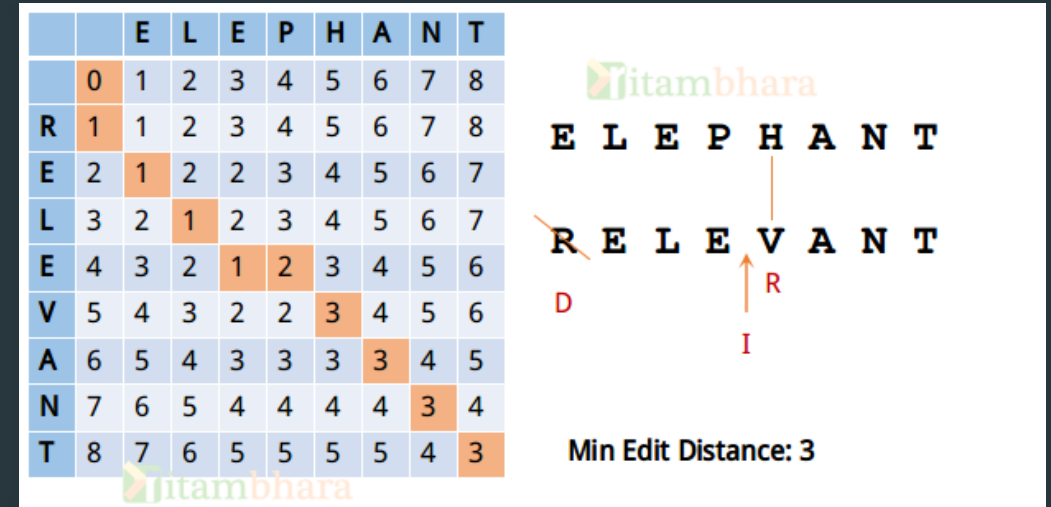
Packfiles



- Large files create a new blob each time they are modified
- This uses a lot of space
- Git can compress "loose" objects into a packfile that removes the redundancy
- Run `git gc` to create a packfile

git diff

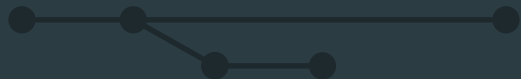
- Git compares files line by line
- Attempts to find the *minimum edit distance* between the files
- Uses the Myers diff algorithm
- Provides a *context* for changed lines



```
234 apstring operator + ( char ch, const apstring & str ) const {
235 // precondition: returns concatenation of ch & str
236 {
237     apstring result; // make string equivalent
238     result = ch;
239     result += str;
240     return result;
241 }
242
243 apstring operator + ( const apstring & str, char ch ) const {
244 // precondition: returns concatenation of str & ch
245 {
246     apstring result(str);
247     result += ch;
248     return result;
249 }
250
251
252 apstring apstring::substr(int pos, int len) const {
253 //description: extract and return the substring
254 //              at index pos
255 //precondition: this string represents c0, c1, ..., cn-1
256 //              0 <= pos <= pos + len - 1 < n.
```

```
43
44 // indexing
45
46 char operator[] (int k) const {
47     char &operator[] (int k);
48
49 // modifiers
50
51 const apstring &operator += ( const apstring & str );
52 const apstring &operator += ( char ch );
53
54
55 private:
56     int myLength; // len
57     int myCapacity; // cap
58     char * myCString; // str
59 };
60
61 // The following free (non-member) functions are
62 //
63 // I/O functions
64
```

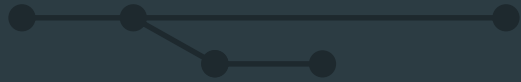
git merge



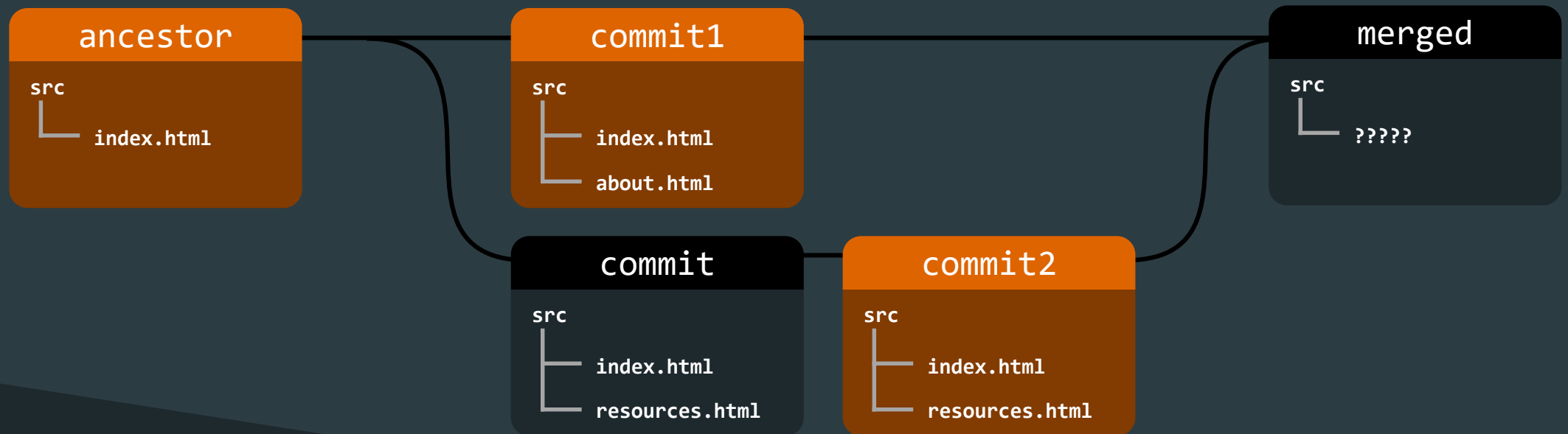
- How does git merge two commits?



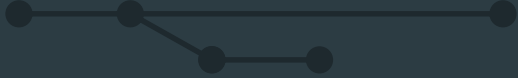
git merge



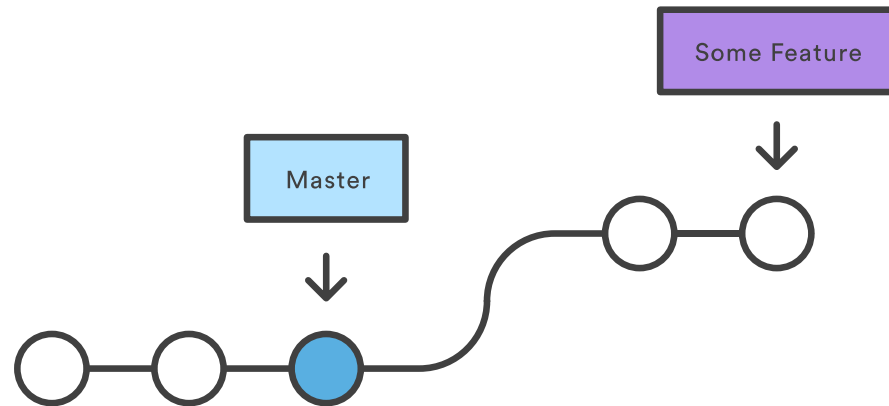
- How does git merge two commits?
- We need the least common ancestor



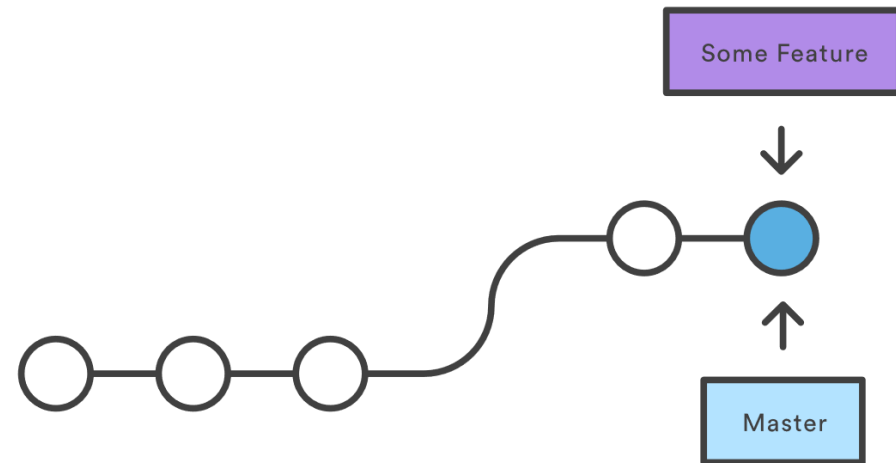
Fast-forward merge



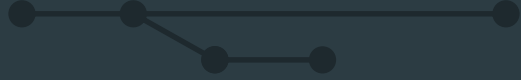
Before Merging



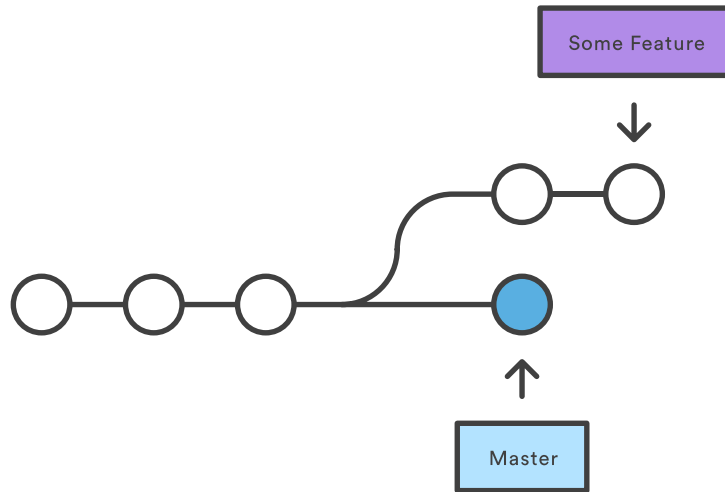
After a Fast-Forward Merge



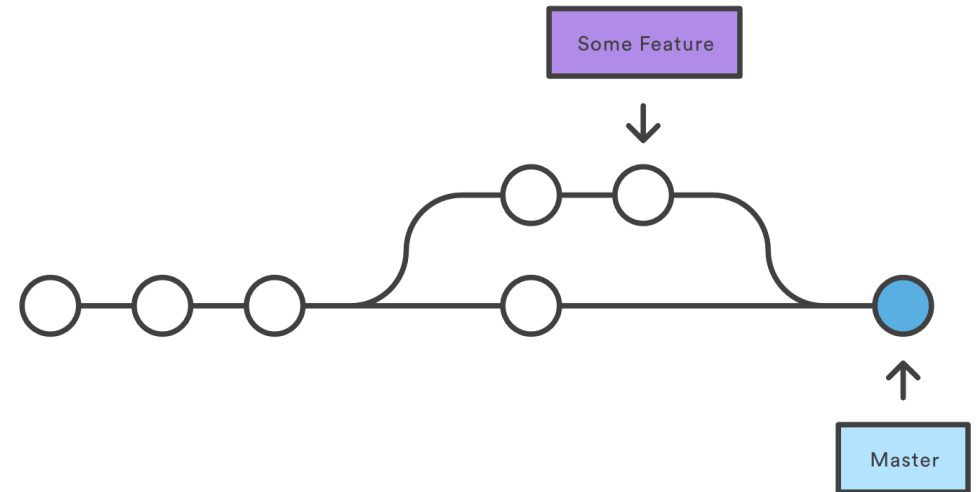
3-way merge



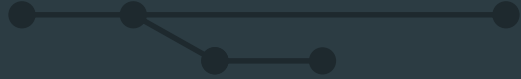
Before Merging



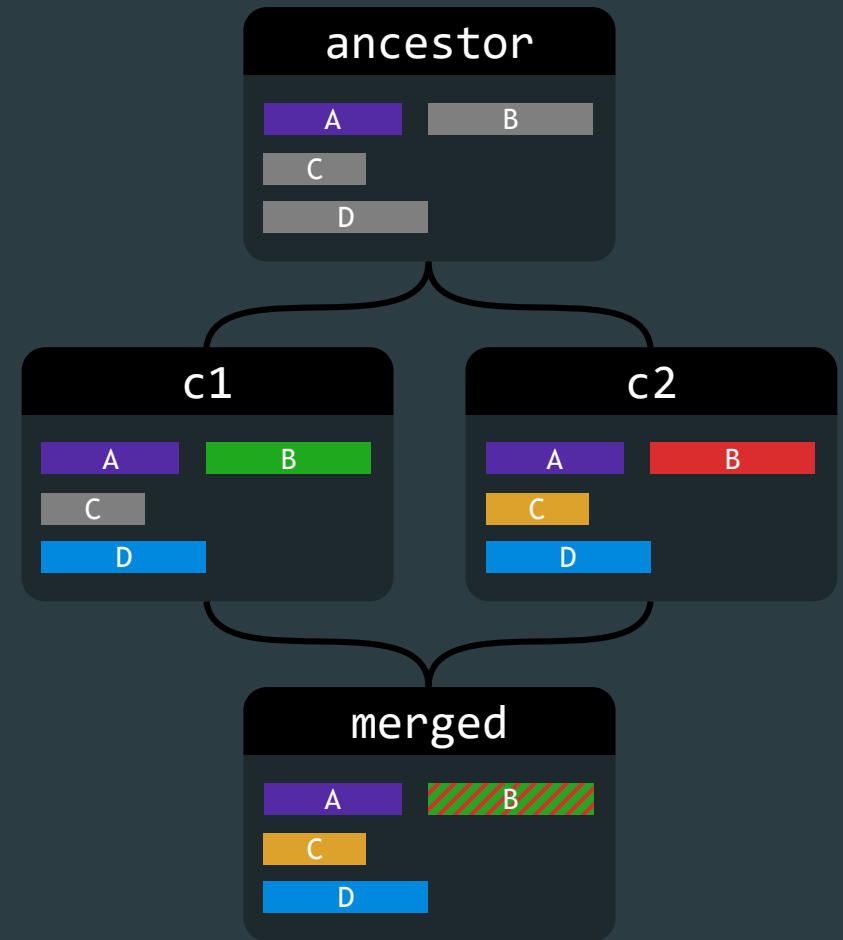
After a 3-way Merge



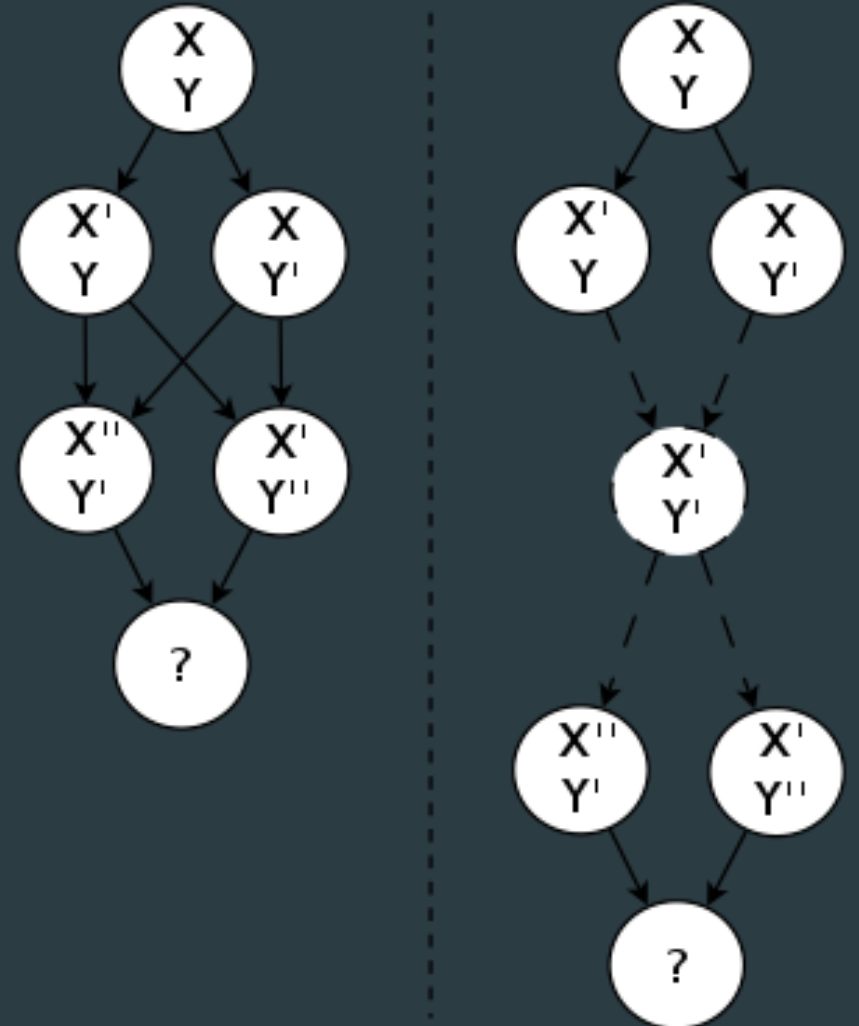
Merging files



- Git diffs both files against the least common ancestor
- Regions that are the same across all three are kept
- If regions are the same in 2 of the three, the version not in the ancestor is used
- If all three differ, there is a conflict

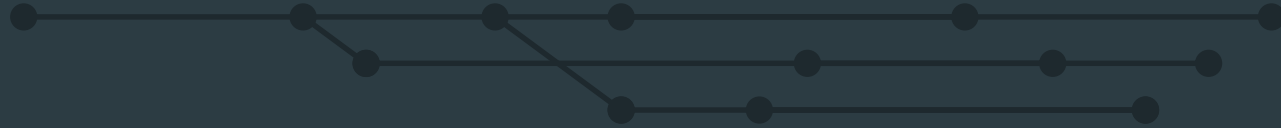


- Sometimes there are multiple least common ancestors!
- In this case, git creates a "virtual ancestor" by merging the ancestors first
- But they might also have multiple ancestors...

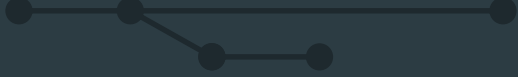




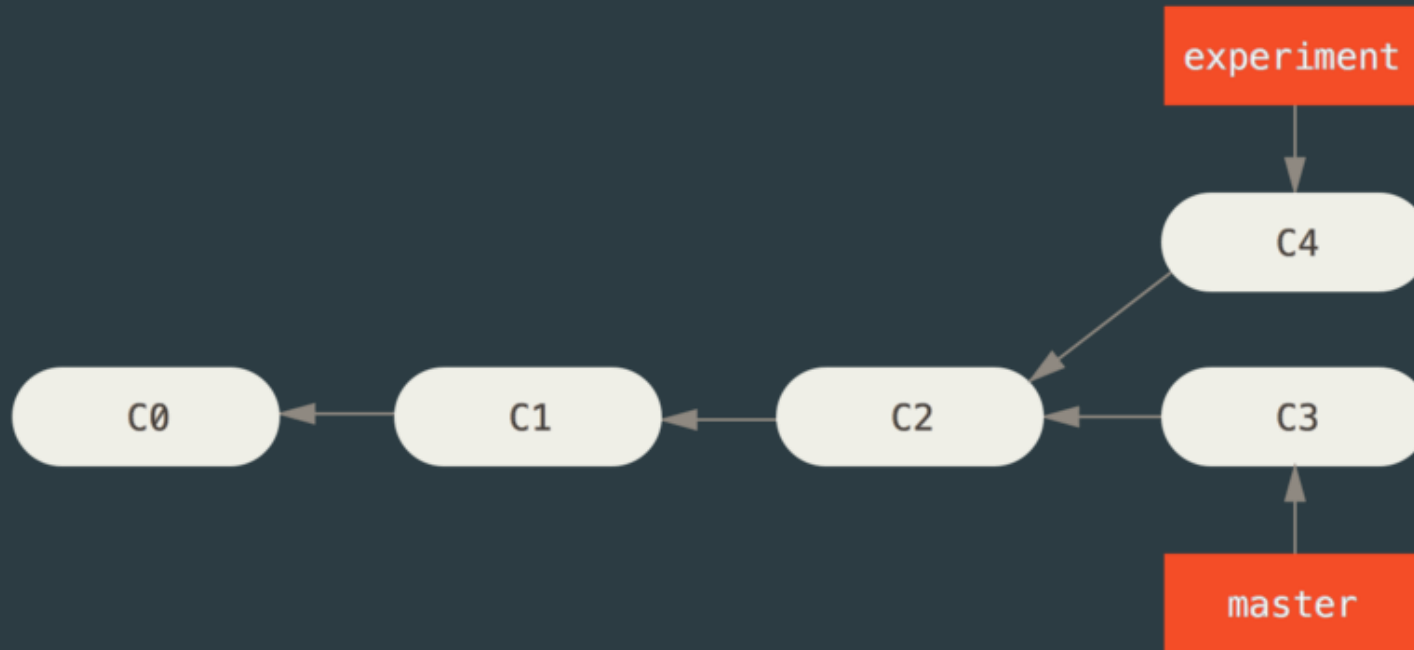
More Git Tricks



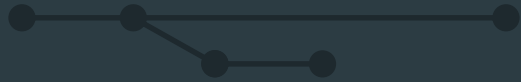
Rebasing



- Lets you move one timeline onto another
- Useful for reordering commits

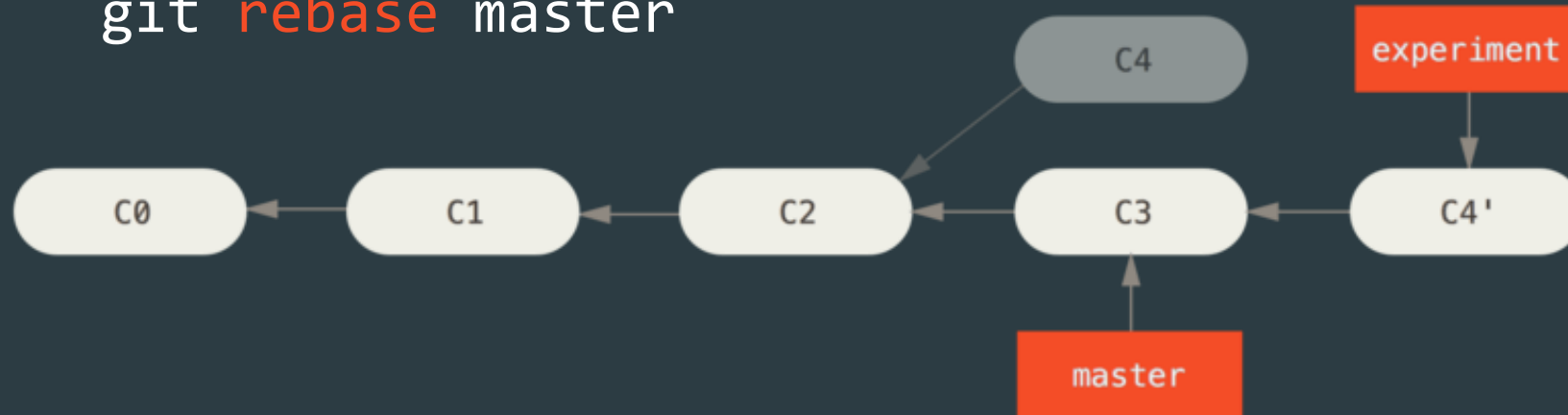


Rebasing

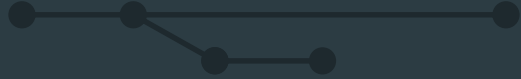


- Lets you move one timeline onto another
- Useful for reordering commits

```
git checkout experiment  
git rebase master
```



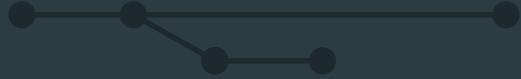
Interactive Rebase



- Triggered with `git rebase -i`
- Lets you choose which commits to include

```
#
# 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
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

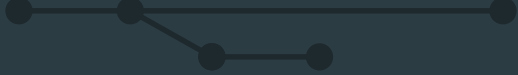
Squashing Commits



- Combine multiple commits into one
- Accessed from interactive rebase

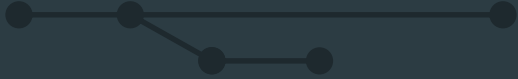
```
pick f7f3f6d changed my name a bit
squash 310154e updated README formatting and added blame
squash a5f4a0d added cat-file
```

.gitignore



- Tells git to ignore certain files
- Can be placed in any subdirectory
 - Matching is relative to the directory
- Uses glob patterns to exclude files
- `.git/info/exclude` can be used for exclusions that are stored locally

filter-branch



- Lets you apply an operation to all commits
- This changes the commit hash of every commit!
- Useful for removing an accidentally committed file

```
git filter-branch --tree-filter 'rm -f passwords.txt' HEAD
```



</presentation>

