Advanced **Git**

Manage your project's plumbing with ease

Nick Nisi

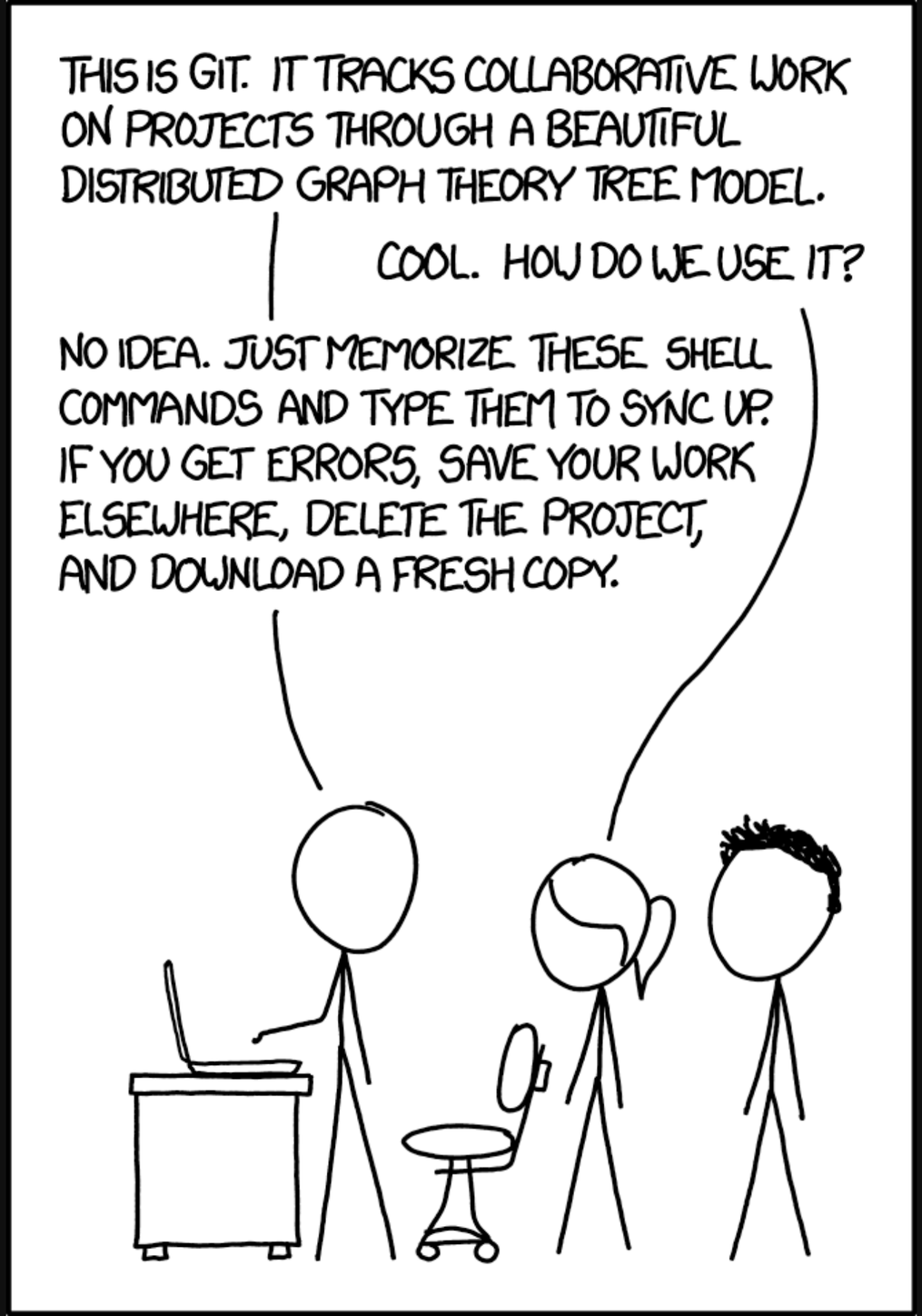
- Developer, SitePen
- Committer, Dojo
- Organizer, NEJS CONF
- Organizer, TSConf
- Organizer, NebraskaJS
- Panelist, TalkScript
- Panelist, JS Party

Overview

- Configuration
- Git Internals
- Git Areas
- Merging and Rebasing
- Keeping History Clean
- Bisect

So… Git.

- Do you use Git every day?
- Do you feel confident with Git?
- Do you change history often?
  - rebasing
  - squashing / interactive rebasing
  - amending commits
- Have you ever deleted a repo and started over?



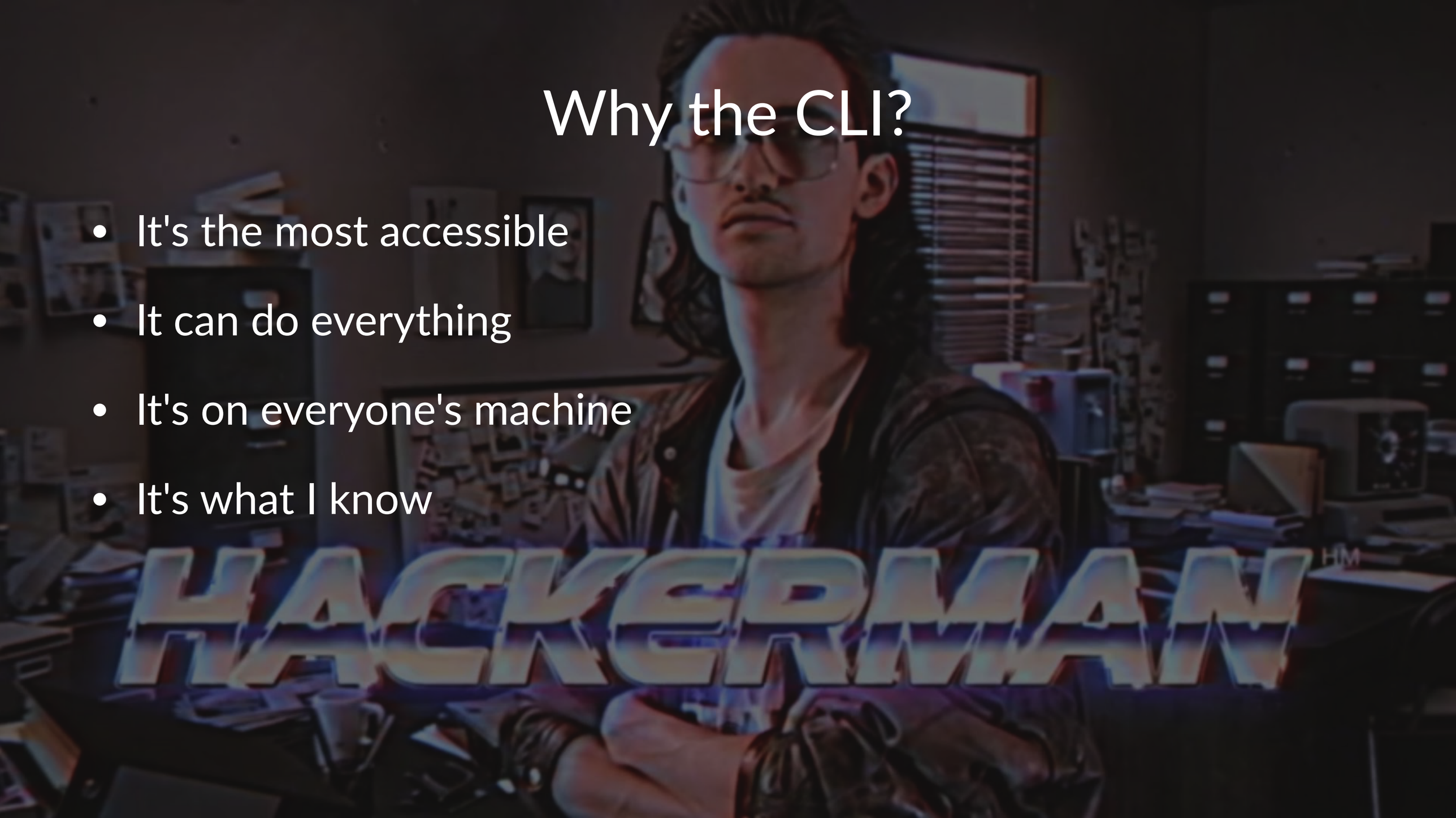
We're not going to resort to this

What is Git?

- A distributed version control system
- Each clone contains the entire history of the project
- It's better than Subversion 😱

Why the CLI?

- It's the most accessible
- It can do everything
- It's on everyone's machine
- It's what I know



Terminology

- **Repository** - The area your project is stored safely (Git)
- **Blob** - An object in Git that keeps track of file contents
- **Tree** - An object in Git that keeps track of file names and permissions
- **Commit** - An object in Git pointing to a tree of file changes with a commit message and reference to a parent
- **Staging Area** - A middle-ground between your working directory and your repository

Configuring **Git**

Configuring Git

- Global configuration goes in a `~/.gitconfig` file
- Every Git project will use these settings
- Project-specific overrides are possible by modifying `.git/config`
- `git config --global user.email`
- `git config --global user.email jlp@enterprise.tng`
- Omit the `--global` flag to modify local configuration

Tip: Identity

Keep your name and email consistent

Use a full name. Commands such as `git shortlog -sn` are useful to see who's contributing and how much

```
> git shortlog -sn
    22  rorticus
    12  Rory Mulligan
     7  Bradley Maier
     2  Nick Nisi
     2  nicknisi
     1  nick
     1  Paul
     1  Paul Shannon
```

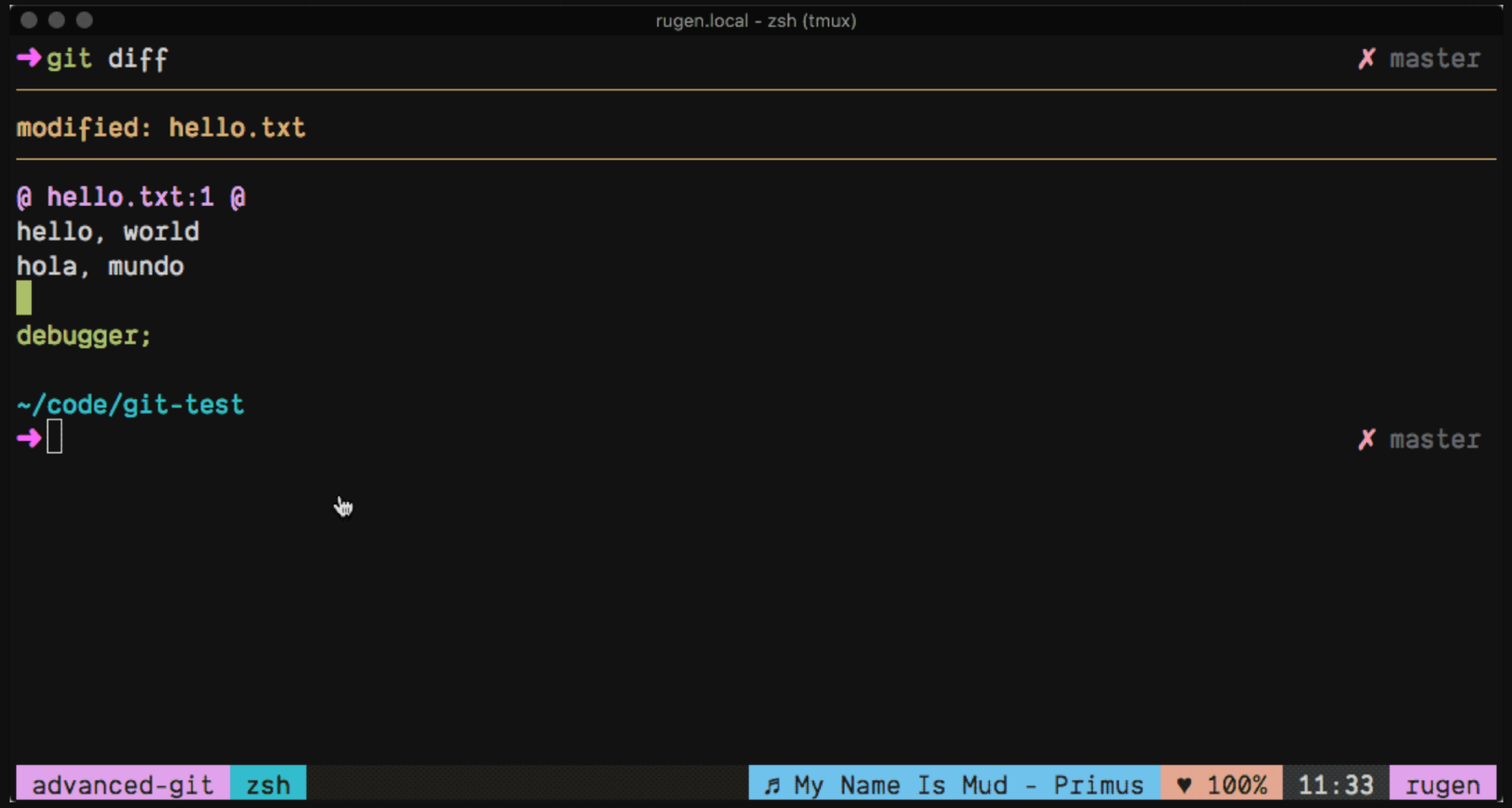
Other Useful Settings

```
[rerere]
  # Reuse Recorded Resolution - Let Git handle conflict resolution if we've already seen it
  enabled = true
[push]
  # Only push the current branch, not all branches
  default = current
[branch]
  # When running `git pull`, always rebase by default
  autosetuprebase = always
[core]
  # point to a global excludes file and keep things that you always want ignored there
  excludesfile = ~/.gitignore_global
  editor = vim # or code, mate, subl
[alias] # We'll talk about other aliases later
  # Don't show information about whitespace changes when running git diff
  diff = diff --ignore-space-at-eol -b -w --ignore-blank-lines
```

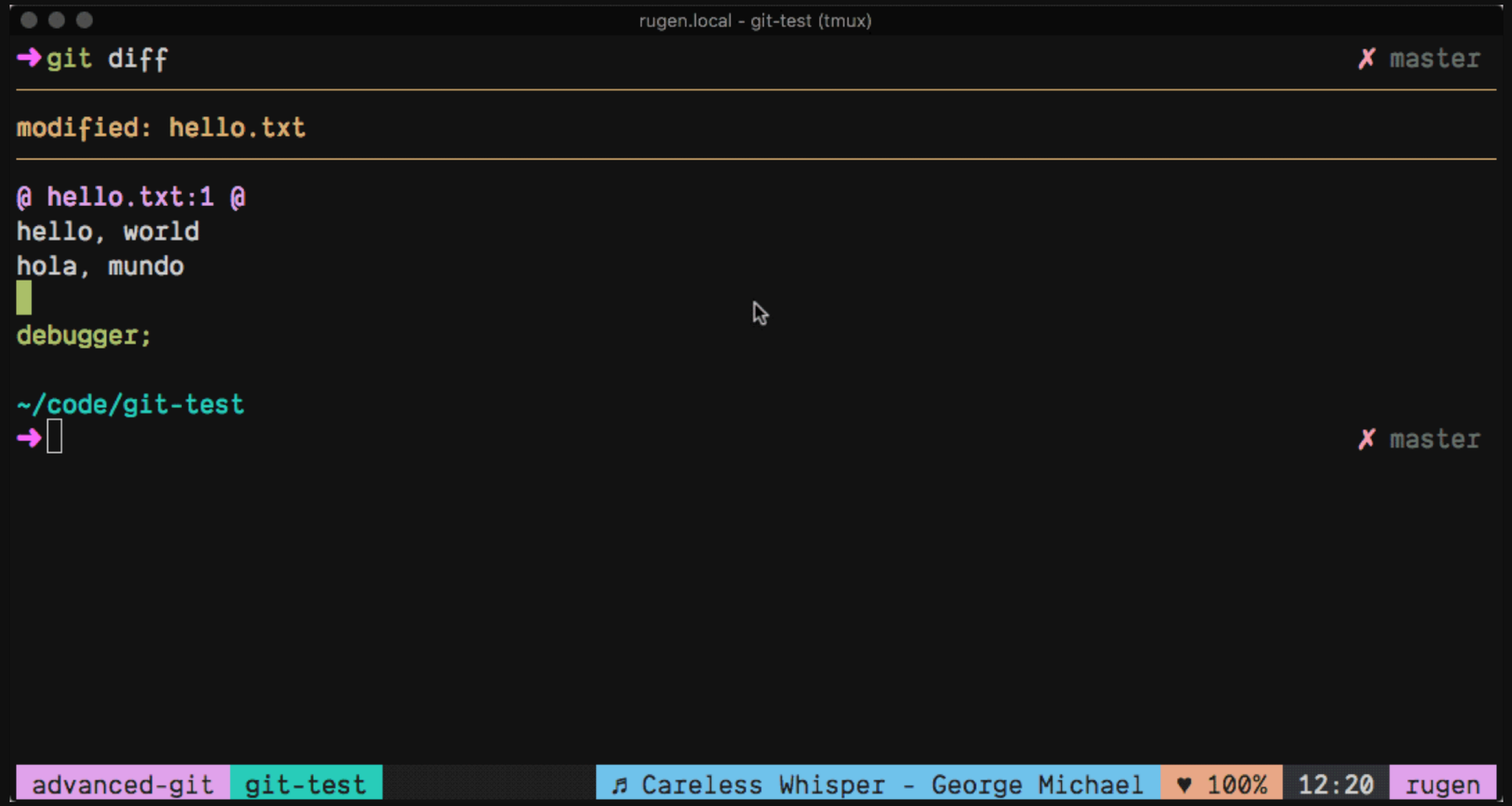
Configuring Git: Aliases

```
[alias]
    co = checkout # shorthand for git checkout
    br = branch # shorthand for git branch
    s = status --short # shortcut to short status

    # undo the last commit, but don't delete them
    undo = reset --soft HEAD~1
    # aliases for assuming files are unchanged
    assume = update-index --assume-unchanged
    unassume = update-index --no-assume-unchanged
    assumed = "!git ls-files -v | grep ^h | cut -c 3-"
    unassumeall = "!git assumed | xargs git update-index --no-assume-unchanged"
```

Alias: `git undo`

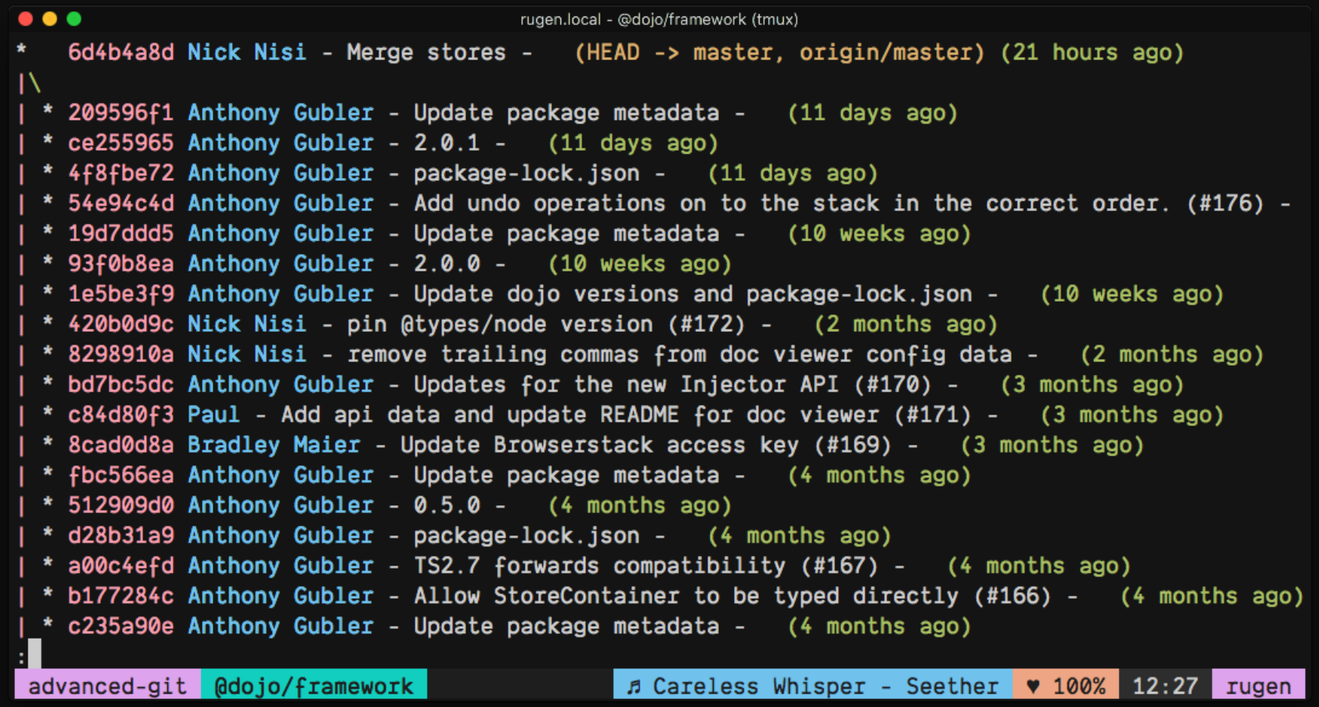


Alias: `git assume`



Alias: `git l` / `glog`

```
log --graph --pretty=format:'%Cred%h%Creset %C(bold blue)%an%C(reset) - %s - %Creset %C(yellow)%d%Creset %Cgreen(%cr)%Creset' --abbrev-commit --date=relative
```



Git Internals

Git Internals

Let's explore the internals of Git just enough to have a basic understanding of what git is doing with our files.

- Understand how Git organizes a repository (from a basic level)
- Trust that it is actually kind of difficult to lose history
- We'll start by creaitng a new repository from scratch

Creating a New Git Repository

```
> git init
Initialized empty Git repository in ~/code/git-test/.git/
```

```
> tree .git/
.git/
├── HEAD
├── config
├── description
├── hooks
|   ├── applypatch-msg.sample
|   └── commit-msg.sample
├── info
|   └── exclude
├── objects
|   ├── info
|   └── pack
└── refs
    ├── heads
    └── tags

8 directories, 7 files
```

```
> echo "hello, world" > hello.txt
> cat hello.txt
hello, world

> git status
On branch master

No commits yet

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

    hello.txt

nothing added to commit but untracked files present (use "git add" to track)
```

```
> echo "hello, world" > hello.txt
> cat hello.txt
hello, world

> git status --short
?? hello.txt
```

```
> git log
fatal: your current branch 'master' does not have any commits yet

> git add hello.txt
> git log
fatal: your current branch 'master' does not have any commits yet
```

```
> tree .git/
.git/
├── HEAD
├── config
├── description
├── hooks
├── index
├── info
│   └── exclude
├── objects
│   ├── 4b
│   │   └── 5fa63702dd96796042e92787f464e28f09f17d
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags

9 directories, 6 files
```

Internals: blob

```
├── objects
|   ├── 4b
|   |   └── 5fa63702dd96796042e92787f464e28f09f17d
```

- This is a git blob containing the contents of hello.txt
- How do I know it's a blob? `git cat-file -t 4b5f`
- The first two digits of the SHA is the directory name
- The next 38 digits are the rest of the hash

```
> cat .git/objects/4b/5fa63702dd96796042e92787f464e28f09f17d
xK .OR04f»HÕ..◊Q(œ/ I·IΣð

> git cat-file -t 4b5f
blob

> git cat-file -p 4b5f
hello, world
```

Blob Structure

A blob stores the the contents of a file with a SHA1 name made up of the following:

Type	`null`	**Content**
blob	`\0`	hello, world

```
blob 13\0hello, world
```

```
> git hash-object hello.txt
4b5fa63702dd96796042e92787f464e28f09f17d

> echo 'blob 13\0hello, world' | openssl sha1
4b5fa63702dd96796042e92787f464e28f09f17d
```

So now we have this `blob`

but we haven't committed anything yet.

This blob was created because we staged `hello.txt` with `git add`.

Git Staging

- A step before the commit process
- The working area and what Git commits are essentially decoupled
- Pick and choose specific files (or parts of files) to commit
- Staging allows commits to be built up and focused, rather than an accumulation of all current changes

```
> git status --short
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   hello.txt

> git status --short
A  hello.txt
```

```
# make further edits
> echo "hola, mundo" >> hello.txt

> cat hello.txt
hello world
hola mundo

> git status --short
AM hello.txt

# revert the changes back to what is staged
> git checkout hello.txt

> cat hello.txt
hello, world

> git status --short
A  hello.txt
```

Commitment

```
> git commit -m "initial commit"
> tree .git/objects
.git/objects
├── 2f
|   └── cf070118eafc860bb87aa933d7a3cbea1c027b
├── 4b
|   └── 5fa63702dd96796042e92787f464e28f09f17d
├── 82
|   └── ad2dff9cb502d849a8e74f6a4f8f1291c173fc
├── info
└── pack

5 directories, 3 files
```

What are these new objects?

Trees and Commits

A `tree` object

```
> git cat-file -t 82ad2dff9cb502d849a8e74f6a4f8f1291c173fc
tree
```

A `commit` object

```
> git cat-file -t 2fcf070118eafc860bb87aa933d7a3cbea1c027b
commit
```

A `tree` Object

A `tree` stores information about filenames and permissions for associated blobs.
There is a tree object for each directory in the workspace

```
> git cat-file -p 82ad2dff9cb502d849a8e74f6a4f8f1291c173fc
100644 blob 4b5fa63702dd96796042e92787f464e28f09f17d    hello.txt
```

permissions	type	SHA1	filename
100644	blob	4b5fa63702dd96796042e92787f464e28f09f17d	hello.txt

A `commit` Object

A commit points to a top-level tree of changes for this commit and contains metadata about the commit[1].

```
> git cat-file -p 2fcf070118eafc860bb87aa933d7a3cbea1c027b
tree 82ad2dff9cb502d849a8e74f6a4f8f1291c173fc
author Nick Nisi <nick@nisi.org> 1530405835 -0500
committer Nick Nisi <nick@nisi.org> 1530405835 -0500

initial commit
```

---

[1] Because this is the first commit, it does not contain a reference to a parent commit.

`commit` Contents

A commit object contains the following:

- `tree` - The top-level tree object, which points to `blob` and `tree` objects
- `parent` - The parent commit
- `author` - Name and email of the commit's author
- `committer` - Name and email of the person who commited the

---

[2] The committer and author can be different when, for example, the committer rewrites history but does not change the original author of a commit. A rebase is an example where this can happen.

Git References

A reference, or ref, is anything pointing to a commit.

- branches
- tags
- remote branches

```
> tree. git/refs
.git/refs
├── heads
|   └── master
└── tags
```

Git References

```
> cat .git/refs/heads/master
2fcf070118eafc860bb87aa933d7a3cbea1c027b

# create a new branch
> git checkout -b feature-branch

> cat .git/refs/heads/feature-branch
2fcf070118eafc860bb87aa933d7a3cbea1c027b

> git checkout master

# .git/HEAD always points to the current ref or commit
> cat .git/HEAD
ref: refs/heads/master
```

HEAD

```
> cat .git/HEAD
ref: refs/heads/master

> cat .git/refs/heads/master
6fb33c0677d495d883baf73e12d1d72597c49789
```

HEAD always points to the current branch (or top-level commit)

git reflog

The Reflog is a log of what `.git/HEAD` has pointed to and the actions taken to get there.

```
> git reflog
2fcf070 HEAD@{0}: checkout: moving from feature-branch to master
2fcf070 HEAD@{1}: checkout: moving from master to feature-branch
2fcf070 HEAD@{2}: commit (initial): initial commit
```

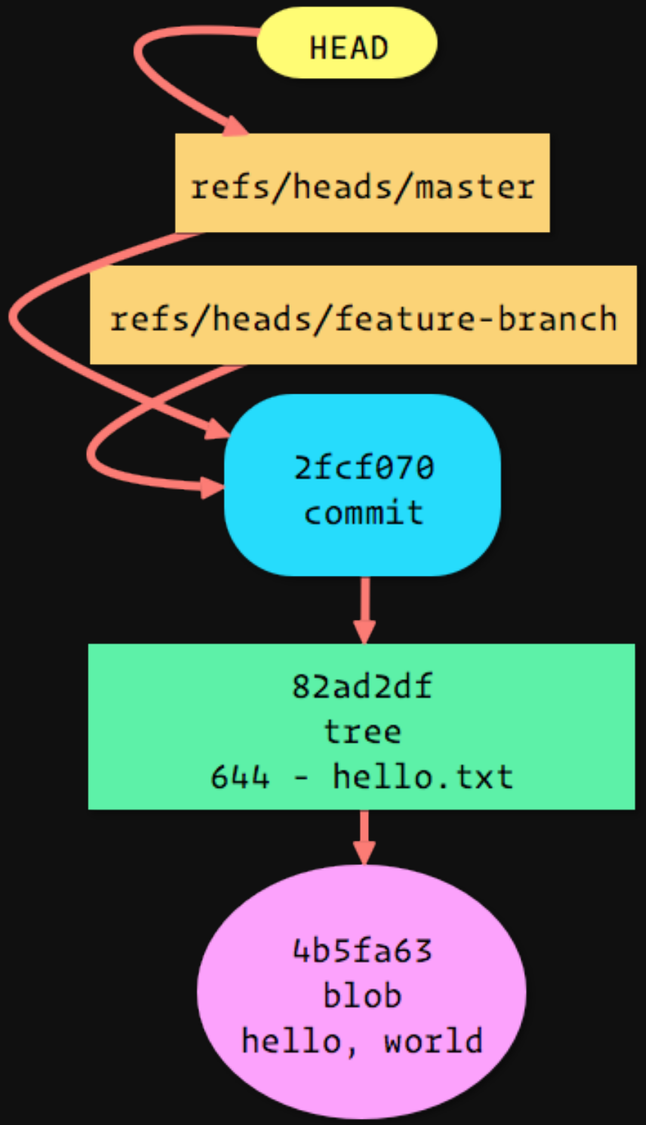
Quick Review

- `git init` creates the repository (the `.git/` directory)
- `git add` adds changes to a staging area, creates `blob` objects
- `blob` objects contain file contents
- `git commit` creates `tree` and `commit` objects
- `tree` objects point file permission and names to associated `tree` and `blob` objects
- `commit` objects contain commit information, parent pointer, and top-level `tree`

Quick Review

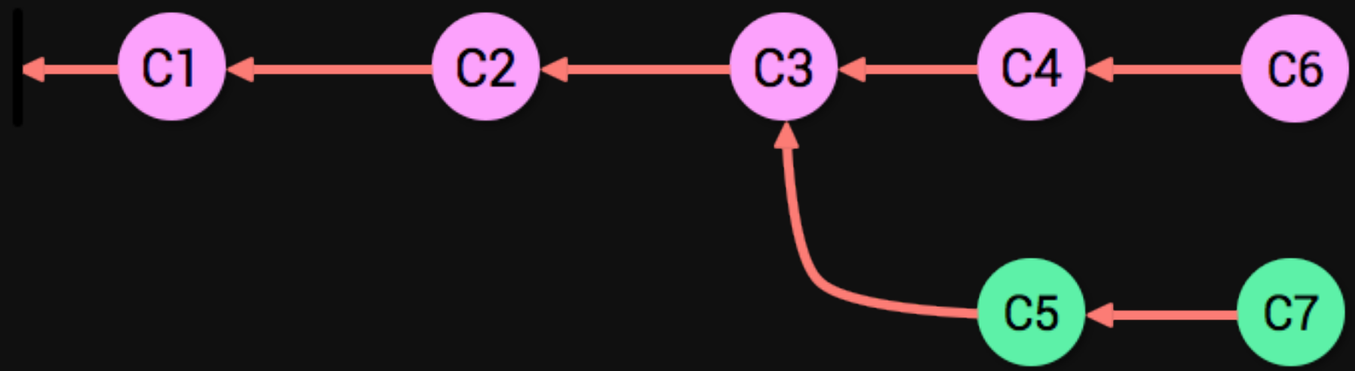
- `git cat-file` can show the type (`-t`) and contents (`-p`) of an object
- The Git staging area can be use to build up commits consisting of specific file or file contents changes
- Refs point to specific commits
- `HEAD` points to refs or to specific commits

```
> tree .git/
.git/
├── HEAD
├── objects
|   ├── 2f
|   |   └── cf070118eafc860bb87aa933d7a3cbea1c027b
|   ├── 4b
|   |   └── 5fa63702dd96796042e92787f464e28f09f17d
|   ├── 82
|   |   └── ad2dff9cb502d849a8e74f6a4f8f1291c173fc
|   ├── info
|   └── pack
└── refs
    ├── heads
    |   ├── feature-branch
    |   └── master
    └── tags
```



Commits Always Point to a Parent

This forms a linked list where each commit knows who its parent is



This makes branching cheap and easy as each branch just needs to point to a commit.

Exercise 01: Multiple Commits

Complete the first exercise.

A Quick Note

Git will eventually take the objects and compress them into a Packfile. This is an effort to reduce the size of the repo.

- Shrink duplicate objects into the Packfile
- Stores deltas between similar objects instead of complete copies

Packfiles are created automatically by Git eventually or can be manually created by running the `git gc` command.

Summary

- We now know how Git stores objects internally
- If we delete delete files, the blobs are still around until a GC happens (90 days by default)
- We saw a glimpse of the reflog, which keeps track of where `HEAD` has been

Git Areas

Git Areas

A file in Git has three main states:

- 0. Untracked
- 1. Committed
- 2. Staged
- 3. Modified

Untracked File

```
> echo "hello" > test.txt
> git status --short
?? text.txt
```

Staged File

```
> git add test.txt
> git status --short
A test.txt
```

Committed File

```
> git commit -m "test commit"
git status --short
```

Modified File

```
> echo " world" >> test.txt
> git status --short
M test.txt
```

Committed File

```
> git commit -m "make a minor change to test.txt"
> git status --short
```

Git has four main areas

If you have anything staged, commit now or stash forever

Git Areas

- 0. Stash
- 1. Working Tree
- 2. Staging Area
- 3. Repository
- 4. Remote Repository

1. Working Tree

Contains a single checkout of a version of the project.
This is where files are created, modified, and deleted.

```
> ls -l
total 8
-rw-r--r--   1 nicknisi  staff    25 Jul  8 21:47 hello.txt
drwxr-xr-x   3 nicknisi  staff    96 Jul  8 21:47 lib
```

2. Staging Area

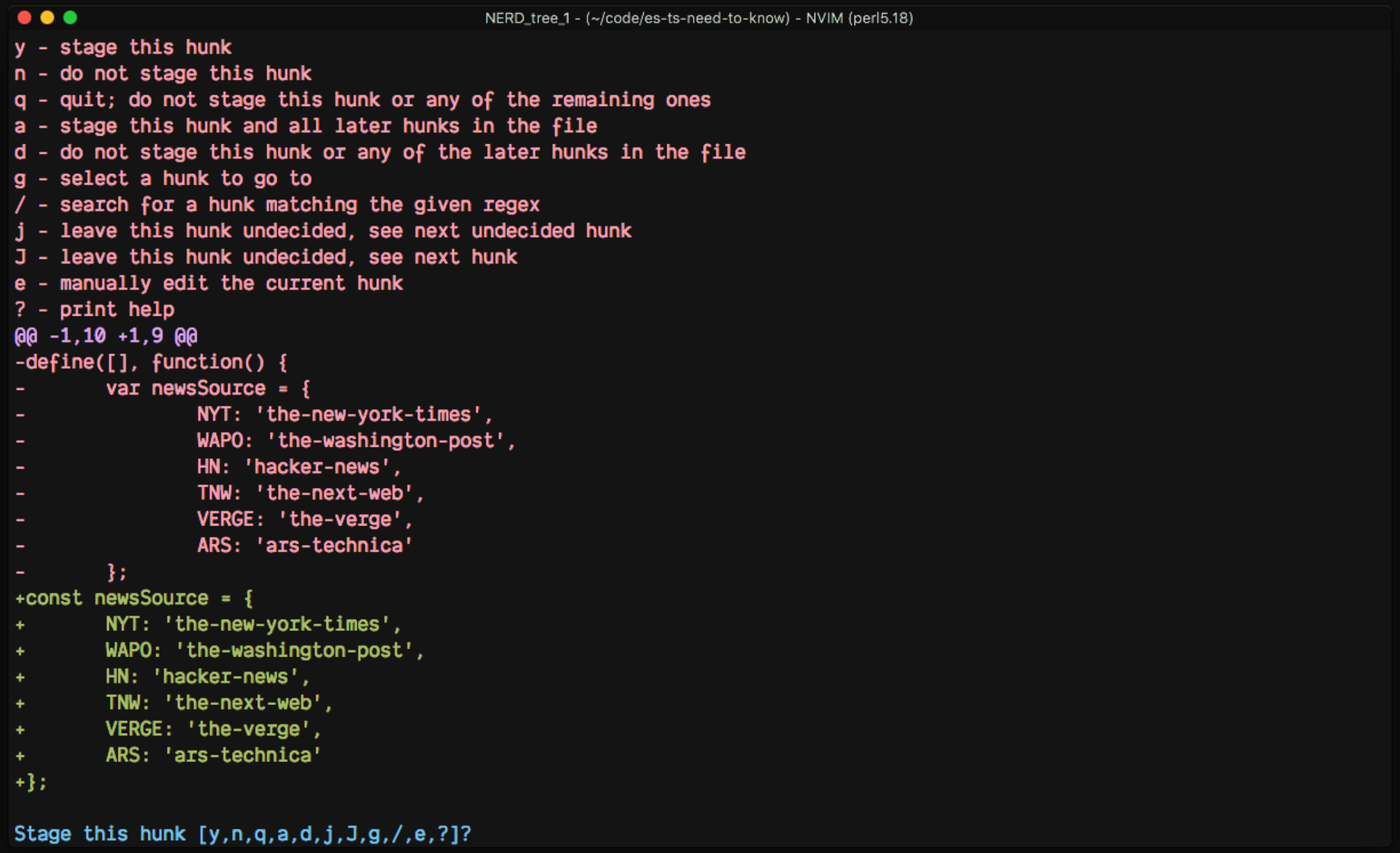
This is step **BEFORE** the commit process.

- Allows you to be as precise or as blunt as you like when crafting commits
- Commit working directory across multiple commits
- `git add .`
- `git add hello.txt`
- `git add -p` / `git add --patch`

The `-p` Flag

Your best friend

- Select specific files or parts of files to stage
- The tool interactively walks you through all hunks in your working tree
- You decide what is staged and what isn't
- Also works with `git checkout` and `git reset`


NERD_tree_1 - (~/code/es-ts-need-to-know) - NVIM (perl5.18)
y - stage this hunk
n - do not stage this hunk
q - quit; do not stage this hunk or any of the remaining ones
a - stage this hunk and all later hunks in the file
d - do not stage this hunk or any of the later hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
e - manually edit the current hunk
? - print help
@@ -1,10 +1,9 @@
-define([], function() {
-	var newsSource = {
-		NYT: 'the-new-york-times',
-		WAPO: 'the-washington-post',
-		HN: 'hacker-news',
-		TNW: 'the-next-web',
-		VERGE: 'the-verge',
-		ARS: 'ars-technica'
-	};
+const newsSource = {
+	NYT: 'the-new-york-times',
+	WAPO: 'the-washington-post',
+	HN: 'hacker-news',
+	TNW: 'the-next-web',
+	VERGE: 'the-verge',
+	ARS: 'ars-technica'
+};
Stage this hunk [y,n,q,a,d,j,J,g,/,e,?]?

3. Repository

- The repository is where all the commits are stored.
- After staging changes, commit them with `git commit`
- Commits are not yet distributed at this point
  - Commits can be lost if the local copy of the repository is destroyed
  - Commits here do not yet affect other users

4. Remote Repository

- A distributed copy of the full repository
- Remotes are set up with the `git remote` command
- Several remotes can be added for a repository
- There can be several remotes, but one is typically considered to be the canonical source
- `origin` is typically the canonical repository unless working with a fork
- If working with a fork, then `upstream` is typically the canonical

Git Remotes

```
> git remote add origin git@github.com:nicknisi/widget-core.git
> git remote add upstream git@github.com:dojo/widget-core.git

> git remote -vv
origin  git@github.com:nicknisi/widget-core.git (fetch)
origin  git@github.com:nicknisi/widget-core.git (push)
upstream    git@github.com:dojo/widget-core.git (fetch)
upstream    git@github.com:dojo/widget-core.git (push)
```

Point the master branch at the upstream repository

```
> git branch --set-upstream-to=upstream/master
```

The Stash

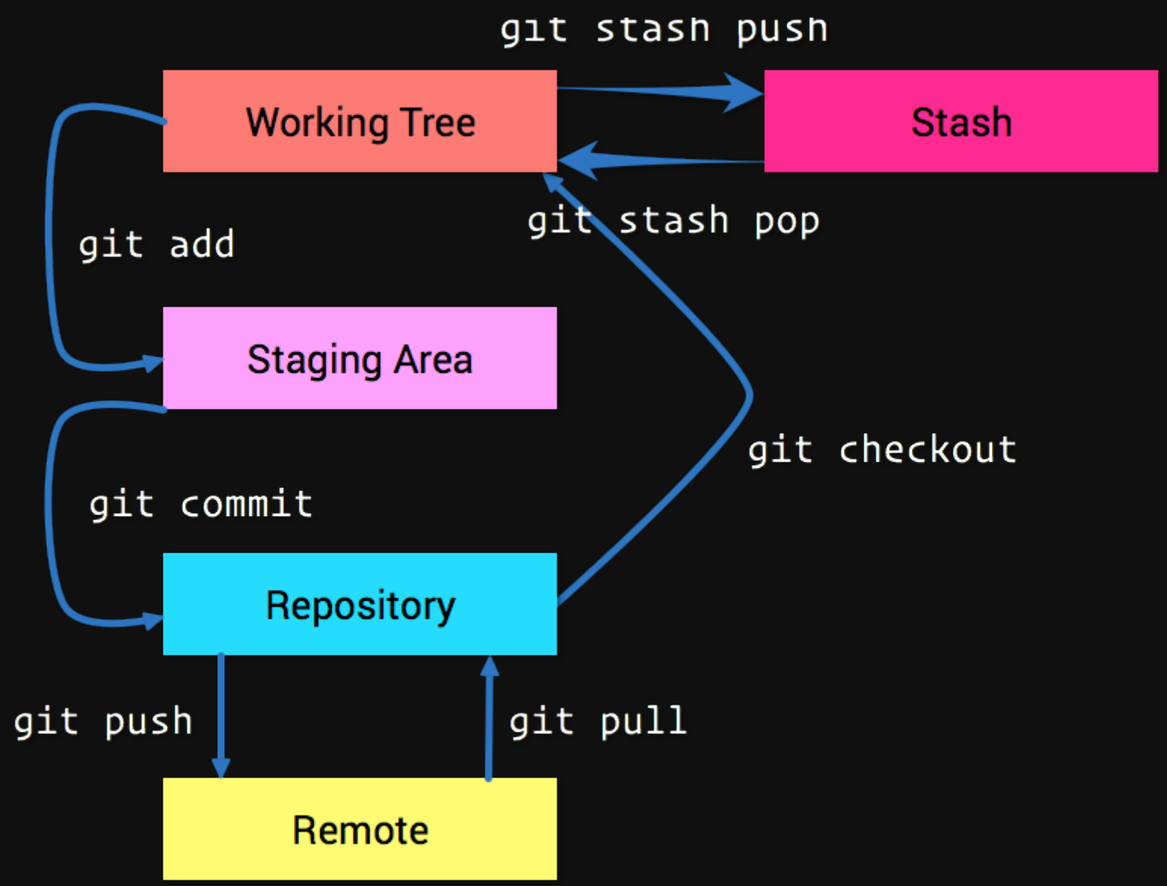
- Stash dirty changes in the working directory
- Makes the working directory clean
- A good, temporary place to store changes that aren't ready to be committed
- `git stash push` / `git stash`
- `git stash pop`, `git stash apply`, `git stash clear`, `git tash show`

Stash: Common Use Cases

- Quickly switching priorities (such as for a bug fix)
- Stashing debug statements / logs
- Stashing changes to bring other changes into the current branch
  - `rebase`, `pull`, `merge` won't work if your working directory is dirty

```
> git diff
diff --git i/hello.txt w/hello.txt
index 419b879..63a2eb8 100644
--- i/hello.txt
+++ w/hello.txt
@@ -1,2 +1,4 @@
 hello, world
 hola, mundo
+
+debugger;

> git stash
Saved working directory and index state WIP on master: 6fb33c0 add spanish hello, goodbye

> git stash pop
On branch master
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:   hello.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (814a54943a6014e9b0dcba6b07c908767f5726da)
```

Review: Git Areas



Exercise 02: Git Areas

Complete the second exercise.

Merging and Rebasing

It is easy to shoot your foot off with git, but also easy to revert to a previous foot and merge it with your current leg.

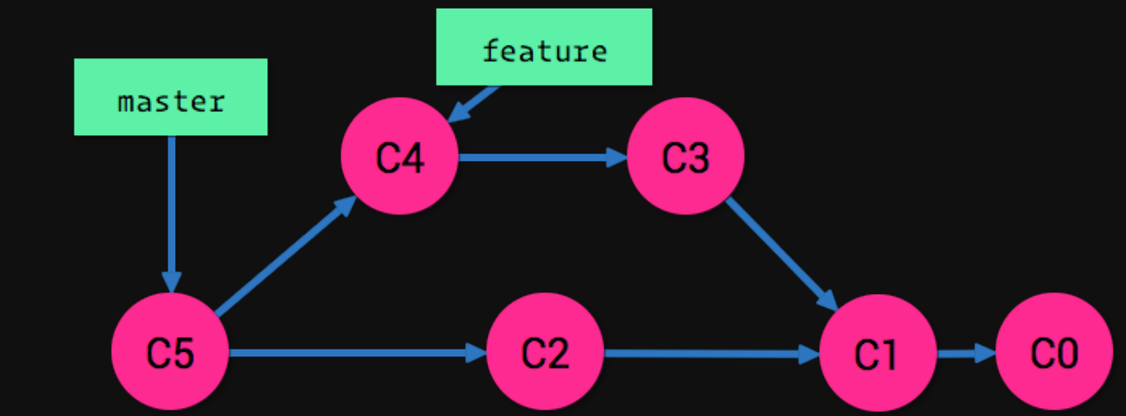
— Jack William Bell

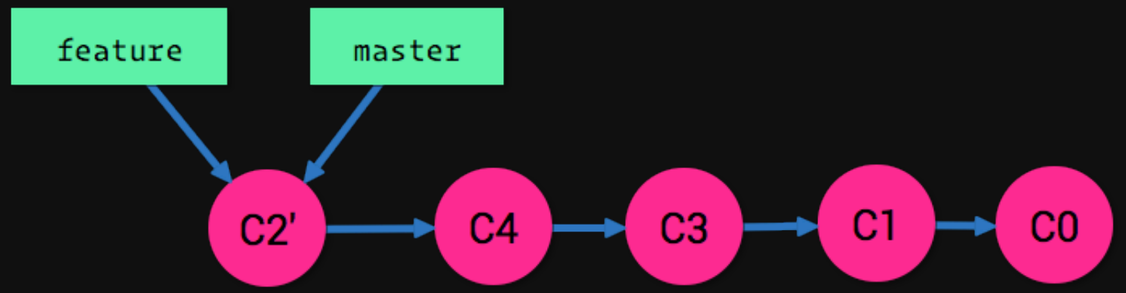
Git is great until you have to work with others.

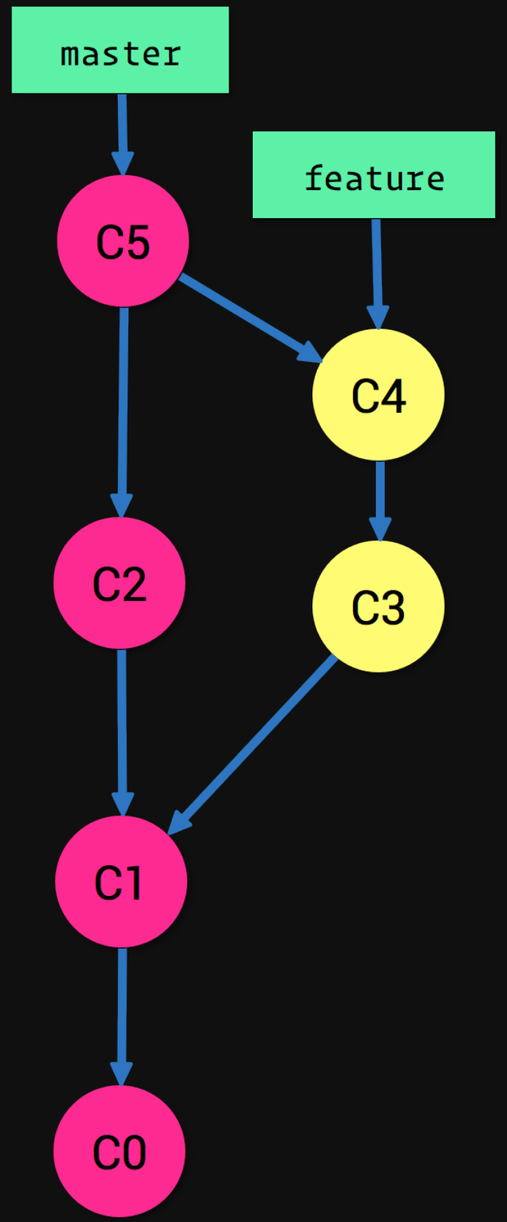
— *Nick Nisi*


Your Git history should tell a chronological story of what happened

Merging and Rebasing







git merge

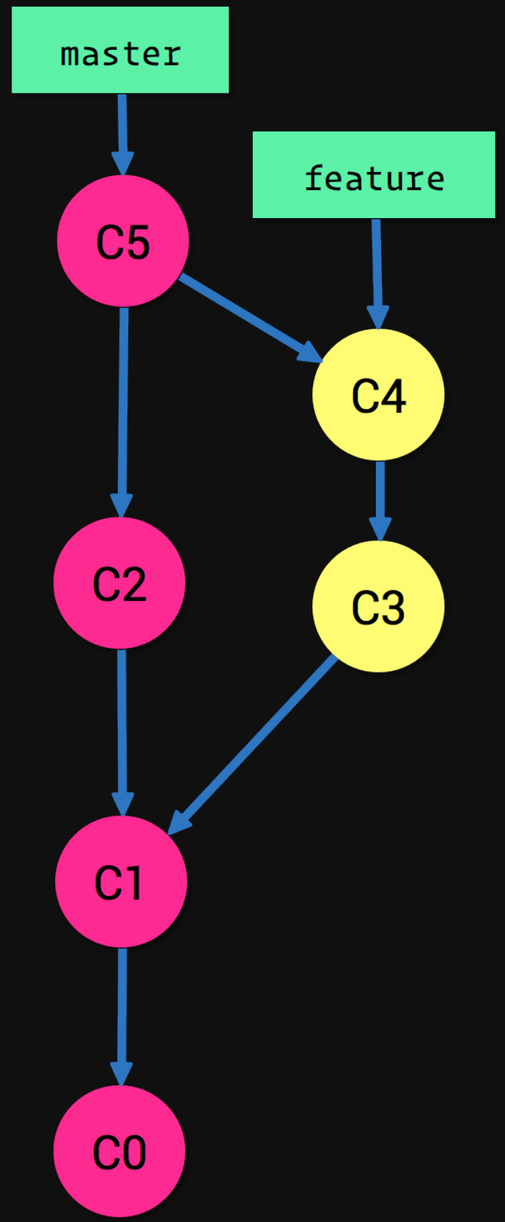
- Incorporate changes from another branch into the current branch
- The contents of the branch are merged into the other branch in a single step
- Can create a merge commit to denote in the history
- All conflicts are taken care of at once

Scenario: Feature branch

- It's typically best practice to do all work in branches
- A feature branch is a branch where a new feature is created
- At some point, the branch is created from a commit off the master branch

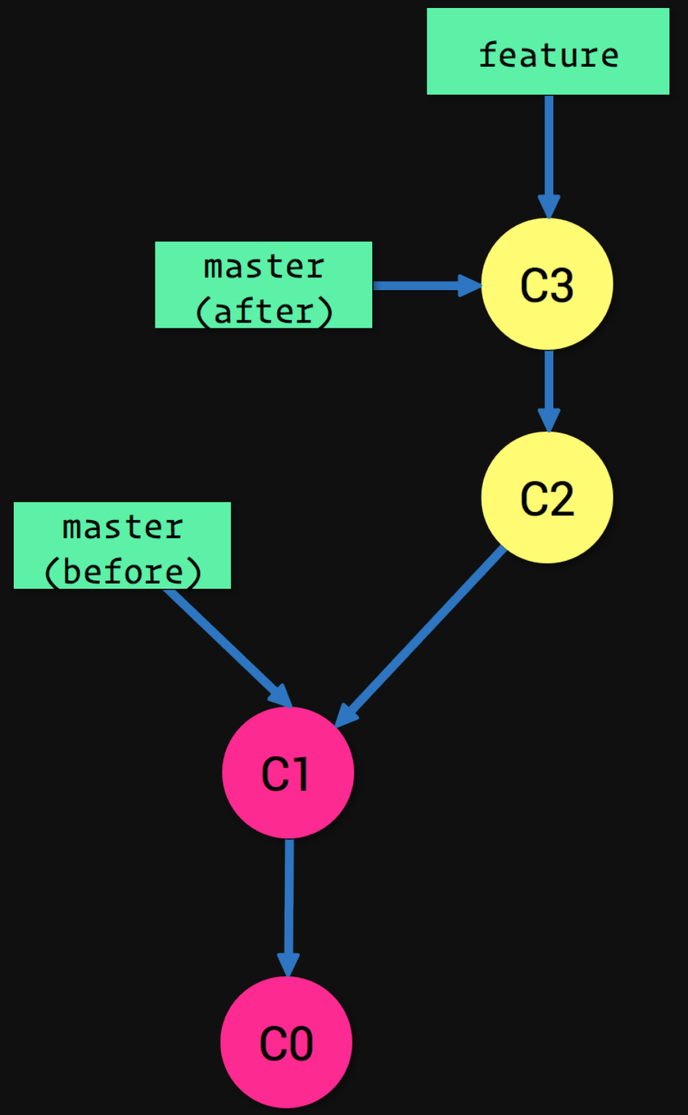
Merging A Feature Branch

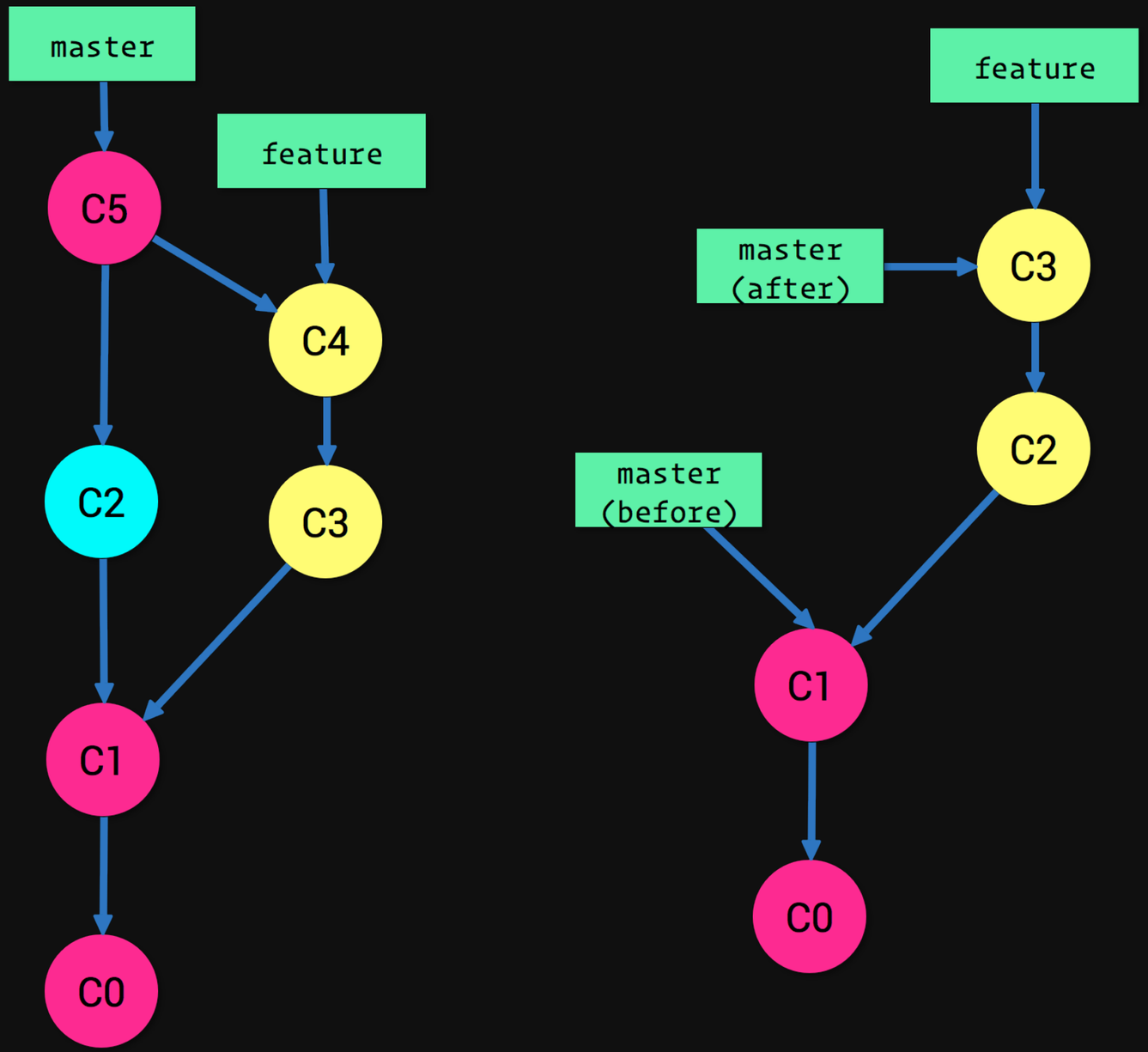
- Once all commits are on the feature branch, switch back to master
- From `master`, run `git merge feature`
- A new commit is created to capture where the branch was merged if the branch has diverged from `master` (C5)



What happens if `master` and `feature-branch` have not diverged?

Fast Forward Merge




master
feature
C5
C4
C2
C3
C1
C0
feature
master
(after)
C3
C2
master
(before)
C1
C0

Controlling ff-merge

- Some git strategies that teams adopt rely on the creation of merge commits
- By default, Git will not create merge commits if the branch can be fast-forwarded
- This can be disabled with the `--no-ff` flag
- Alternatively, merge commits can be disabled with `--ff-only`

```
> git checkout -b feature
Switched to a new branch 'feature'

> git commit --allow-empty -m "feature commit 1"
[feature 1cada0a] feature commit 1

> git commit --allow-empty -m "feature commit 2"
[feature 29f4d7c] feature commit 2
```

```
> git checkout master
Switched to branch 'master'

> git commit --allow-empty -m "master commit 1"
[master 8d9fb65] master commit 1

> git l
* 8d9fb65 - master commit 1
* 6fb33c0 - add spanish hello, goodbye
* 2fcf070 - initial commit
```

```
> git merge feature
Already up to date!
Merge made by the 'recursive' strategy.

> git --no-pager log --oneline --graph
*   1dcdde7 (HEAD -> master) Merge branch 'feature'
|\
| * 29f4d7c (feature) feature commit 2
| * 1cada0a feature commit 1
* | 8d9fb65 master commit 1
|/
* 6fb33c0 add spanish hello, goodbye
* 2fcf070 initial commit
```

Let's reset that

```
> git l
*   1dcdde7 - Merge branch 'feature'
|\
| * 29f4d7c - feature commit 2
| * 1cada0a - feature commit 1
* | 8d9fb65 - master commit 1
|/
* 6fb33c0 - add spanish hello, goodbye
* 2fcf070 - initial commit

> git reset --hard 6fb33c0
HEAD is now at 6fb33c0 add spanish hello, goodbye

> git l
* 6fb33c0 - add spanish hello, goodbye
* 2fcf070 - initial commit
```

```
> git branch -D feature
Deleted branch feature (was 29f4d7c).

> git checkout -b feature2
Switched to a new branch 'feature2'

> git commit --allow-empty -m "feature2 commit 1"
[feature2 09637d8] feature2 commit 1

> git commit --allow-empty -m "feature2 commit 2"
[feature2 476b944] feature2 commit 2
```

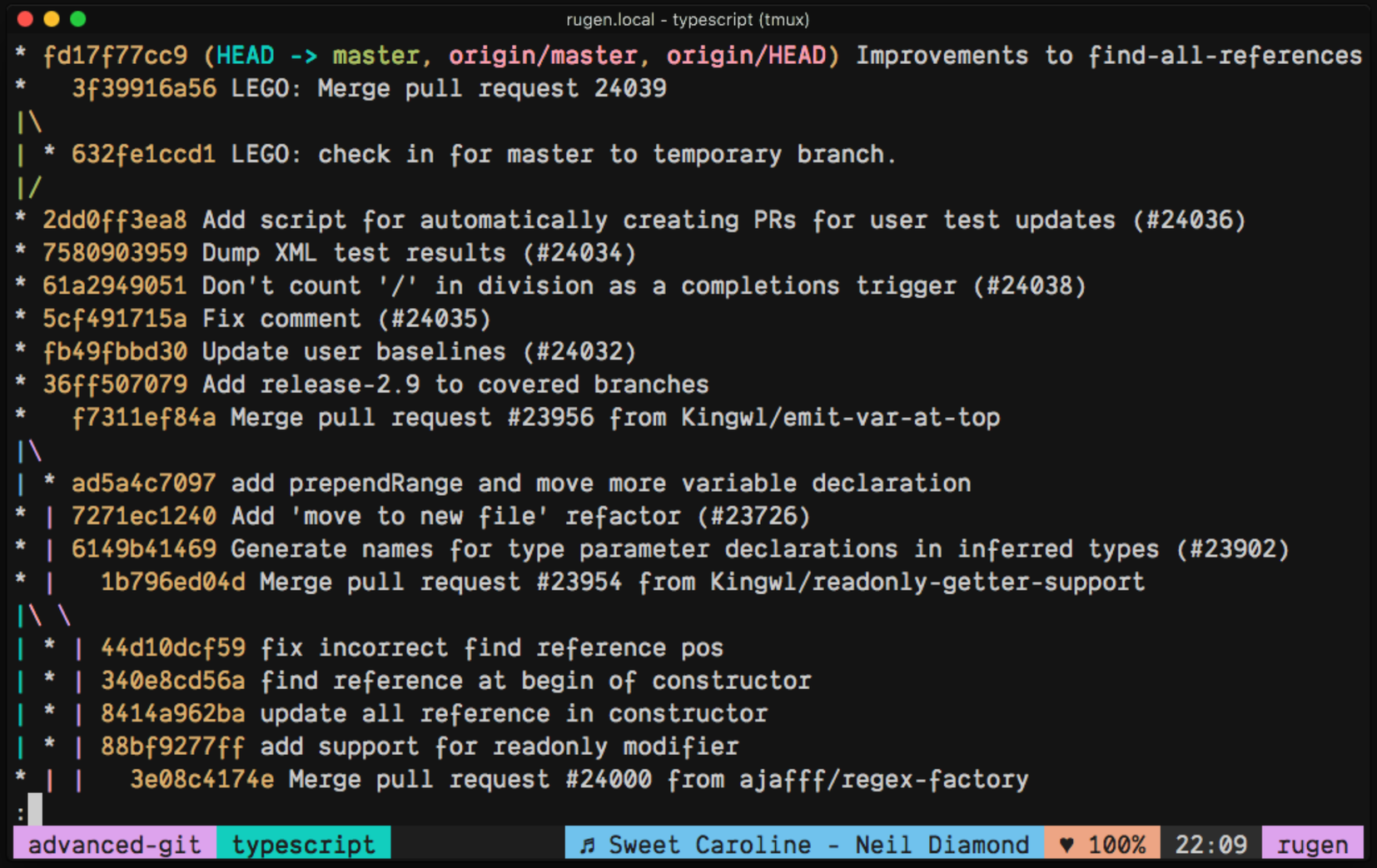
```
> git checkout master
Switched to branch 'master'

> git merge feature2
Updating 6fb33c0..476b944
Fast-forward

> git l
* 476b944 - feature2 commit 2
* 09637d8 - feature2 commit 1
* 6fb33c0 - add spanish hello, goodbye
* 2fcf070 - initial commit
```

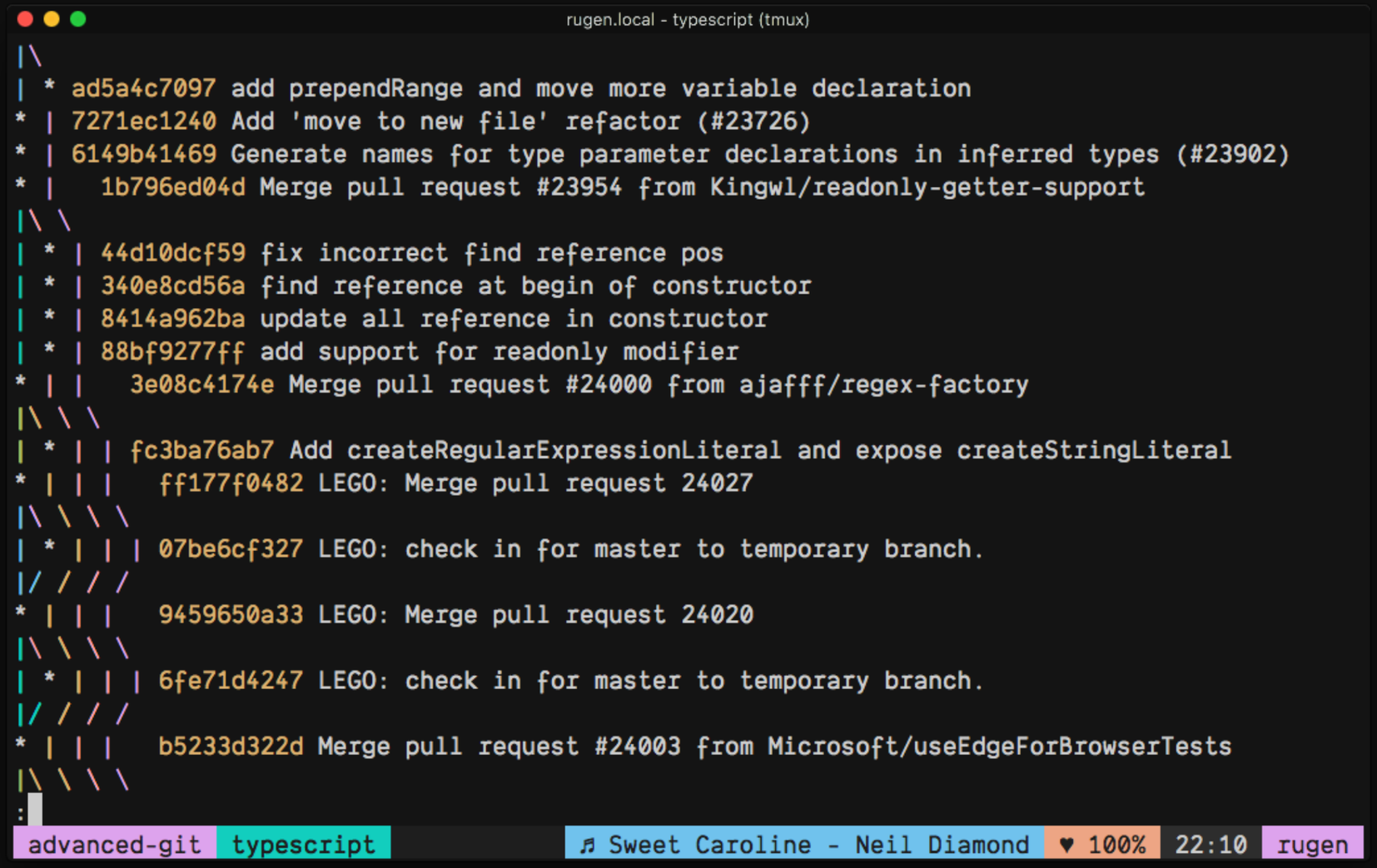
Merge Pros

Traceability - The merge commits will show historical information including full history of the feature branch

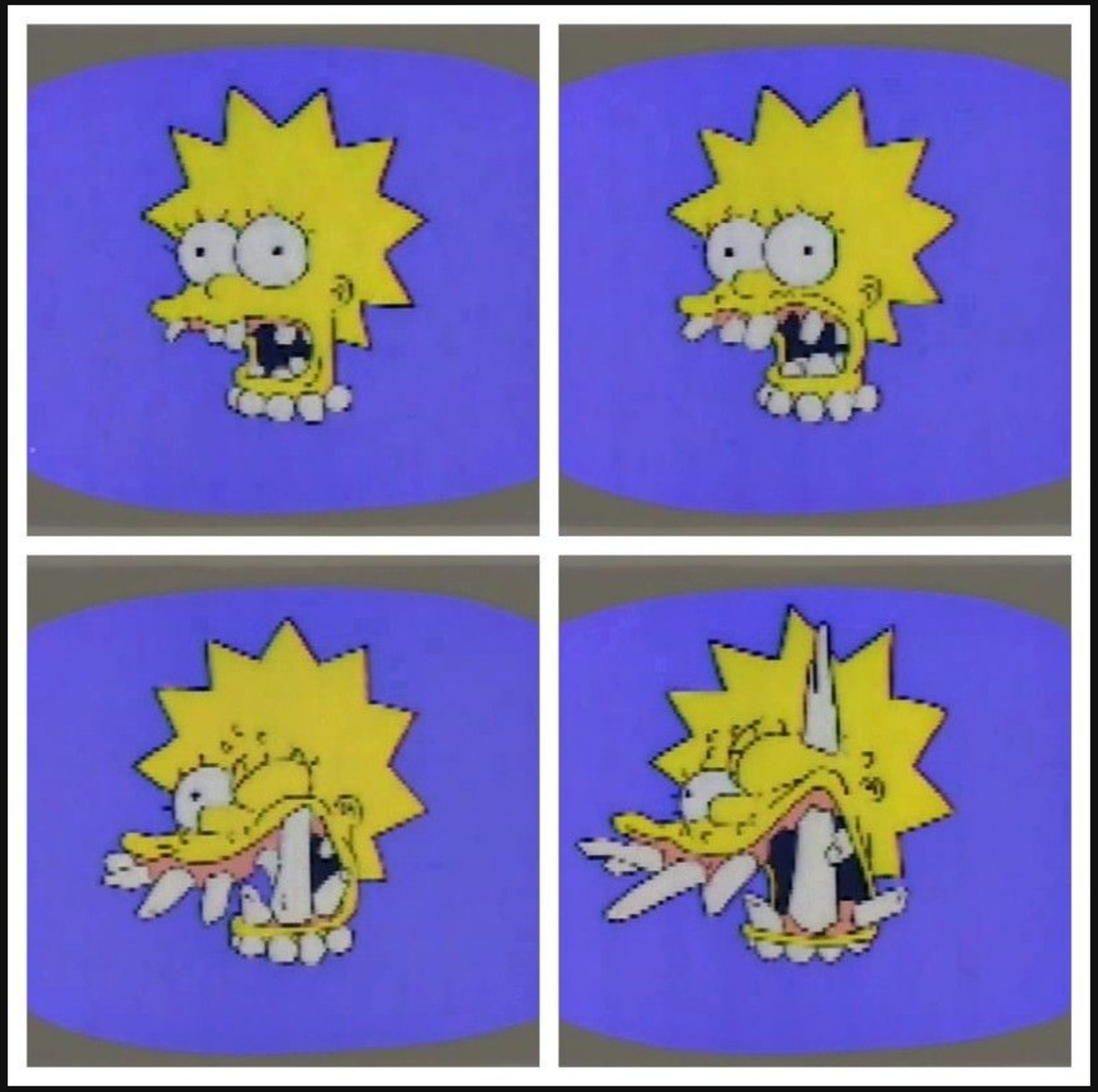
Merge Cons

- The history becomes very polluted. (`--first-parent` flag is helpful here)
- If a feature branch has irrelevant commits, they're now a part of history


rugen.local - typescript (tmux)
* fd17f77cc9 (HEAD -> master, origin/master, origin/HEAD) Improvements to find-all-references
*   3f39916a56 LEGO: Merge pull request 24039
|\
| * 632fe1ccd1 LEGO: check in for master to temporary branch.
|/
* 2dd0ff3ea8 Add script for automatically creating PRs for user test updates (#24036)
* 7580903959 Dump XML test results (#24034)
* 61a2949051 Don't count '/' in division as a completions trigger (#24038)
* 5cf491715a Fix comment (#24035)
* fb49fbbd30 Update user baselines (#24032)
* 36ff507079 Add release-2.9 to covered branches
*   f7311ef84a Merge pull request #23956 from Kingwl/emit-var-at-top
|\
| * ad5a4c7097 add prependRange and move more variable declaration
* | 7271ec1240 Add 'move to new file' refactor (#23726)
* | 6149b41469 Generate names for type parameter declarations in inferred types (#23902)
* | 1b796ed04d Merge pull request #23954 from Kingwl/readonly-getter-support
|\ \
| * | 44d10dcf59 fix incorrect find reference pos
| * | 340e8cd56a find reference at begin of constructor
| * | 8414a962ba update all reference in constructor
| * | 88bf9277ff add support for readonly modifier
* | | 3e08c4174e Merge pull request #24000 from ajafff/regex-factory
:
advanced-git
typescript
♬ Sweet Caroline - Neil Diamond
♥ 100%
22:09
rugen


rugen.local - typescript (tmux)
|\
| * ad5a4c7097 add prependRange and move more variable declaration
* | 7271ec1240 Add 'move to new file' refactor (#23726)
* | 6149b41469 Generate names for type parameter declarations in inferred types (#23902)
* | 1b796ed04d Merge pull request #23954 from Kingwl/readonly-getter-support
|\ \
| * | 44d10dcf59 fix incorrect find reference pos
| * | 340e8cd56a find reference at begin of constructor
| * | 8414a962ba update all reference in constructor
| * | 88bf9277ff add support for readonly modifier
* | | 3e08c4174e Merge pull request #24000 from ajafff/regex-factory
|\ \ \
| * | | fc3ba76ab7 Add createRegularExpressionLiteral and expose createStringLiteral
* | | | ff177f0482 LEGO: Merge pull request 24027
|\ \ \ \
| * | | | 07be6cf327 LEGO: check in for master to temporary branch.
|/ / / /
* | | | 9459650a33 LEGO: Merge pull request 24020
|\ \ \ \
| * | | | 6fe71d4247 LEGO: check in for master to temporary branch.
|/ / / /
* | | | b5233d322d Merge pull request #24003 from Microsoft/useEdgeForBrowserTests
|\ \ \ \
:
advanced-git
typescript
♬ Sweet Caroline - Neil Diamond
♥ 100%
22:10
rugen

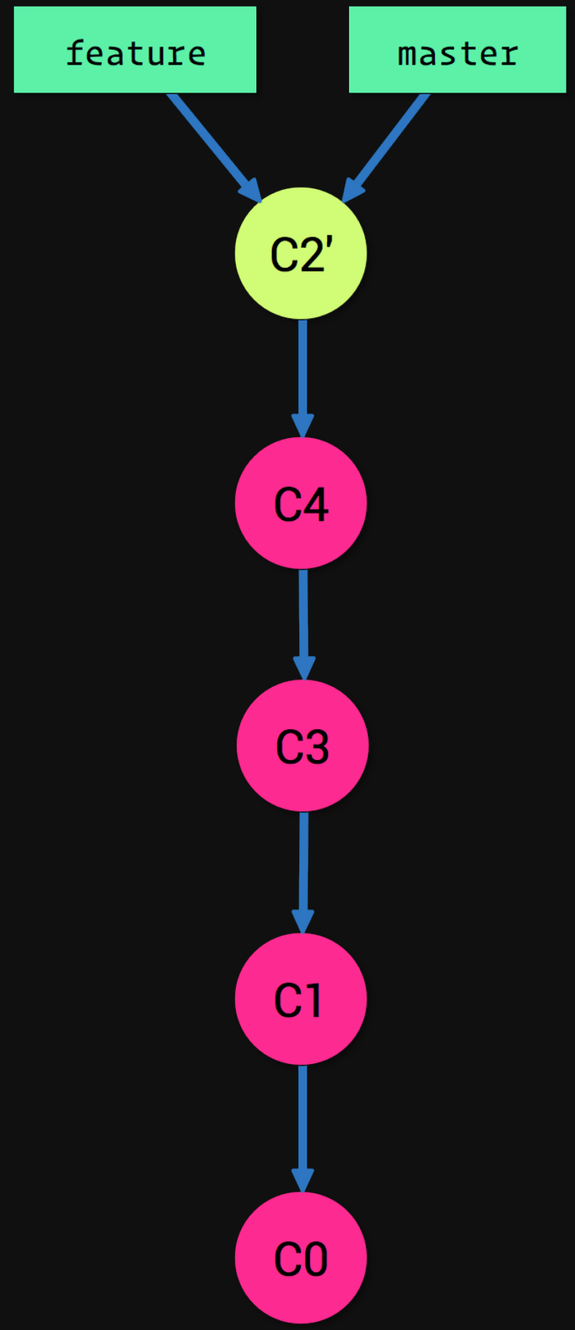

rugen.local - typescript (tmux)
:
advanced-git
typescript
♬ Sweet Caroline - Neil Diamond
♥ 100%
21:49
rugen

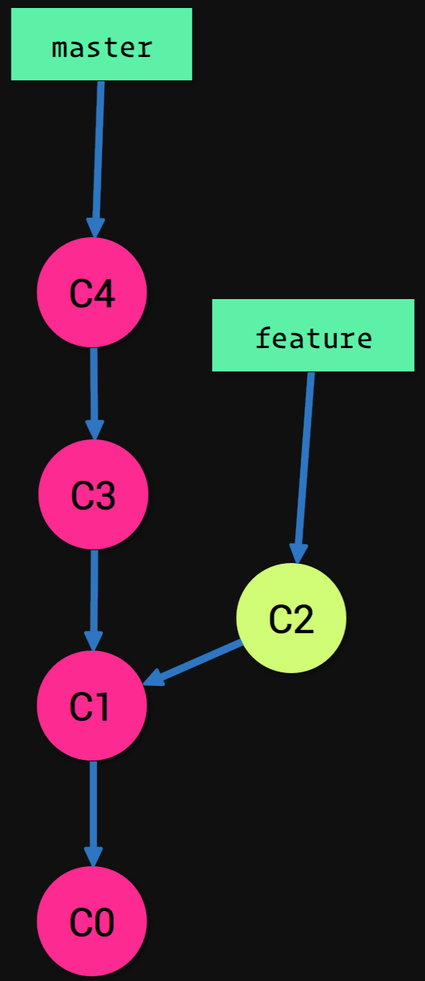


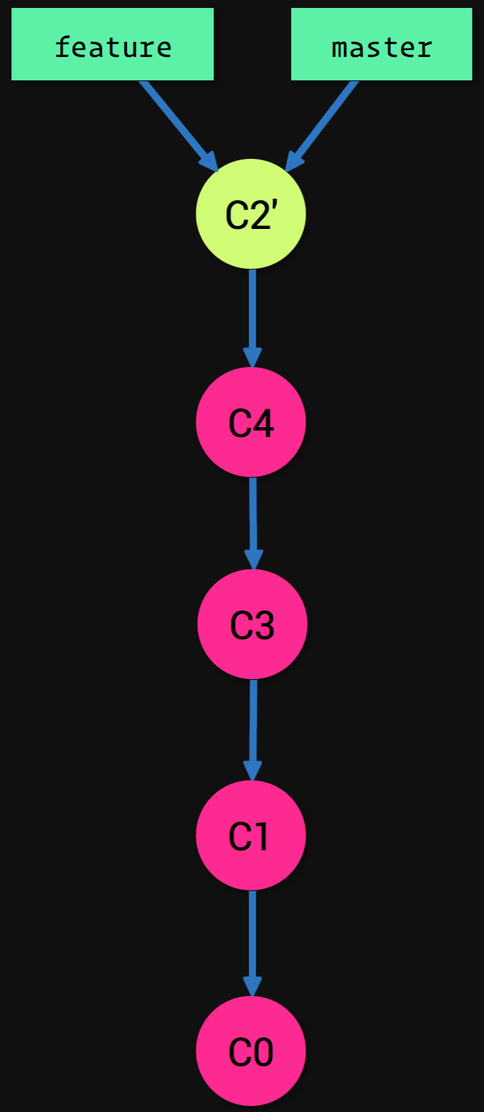

Rebase

`git rebase`

- Find the original diverge point and copy all subsequent commits
- Replay those commits on top of a new diverge point
- Keeps history cleaner
- 😱 **Does modify history**
- Creates new SHAs for the commits




master
C4
C3
C1
C0
feature
C2


feature
master
C2'
C4
C3
C1
C0

```
> git checkout -b feature
Switched to a new branch 'feature'

> git commit --allow-empty -m "feature commit 1"
[feature d98a752] feature commit 1

> git commit --allow-empty -m "feature commit 2"
[feature af66d1b] feature commit 2
```

```
> git checkout master
Switched to branch 'master'

> git commit --allow-empty -m "master commit 1"
[master 8686679] master commit 1

> git commit --allow-empty -m "master commit 2"
[master 3c6af08] master commit 2

> git log --oneline --graph
* 3c6af08 (HEAD -> master) master commit 2
* 8686679 master commit 1
* 6fb33c0 add spanish hello, goodbye
* 2fcf070 initial commit
```

```
> git checkout feature
Switched to branch 'feature'

> git log --oneline --graph
* af66d1b (HEAD -> feature) feature commit 2
* d98a752 feature commit 1
* 6fb33c0 add spanish hello, goodbye
* 2fcf070 initial commit
```

```
> git rebase --keep-empty master
Successfully rebased and updated refs/heads/feature.

> git log --oneline --graph
* 9ce0d3b (HEAD -> feature) feature commit 2
* 31f61ae feature commit 1
* 3c6af08 (master) master commit 2
* 8686679 master commit 1
* 6fb33c0 add spanish hello, goodbye
* 2fcf070 initial commit
```

Rebase

The commits in a rebase are effectively replayed on top of a new diverge point

Rebase Pros

- History remains flat and readable

Rebase Cons

- Can be dangerous. Rebase changes commit SHAs, effectively making new commits and diverging a branch from itself
- If a branch has already been synced with a remote, then it needs to be pushed with `--force`

Both strategies have pros and cons

Both also have full religions dedicated to their one true way online

One nifty tool can make both better

Interactive Rebase

Warning

This next section is dedicated to rewriting history

Good books aren't written - they're rewritten

— Michael Crichton

Interactive Rebase

```
git rebase -i 9ce0d3b
```

```
> git log --oneline --graph
* 6fb33c0 (HEAD -> feature, master) add spanish hello, goodbye
* 2fcf070 initial commit

> git commit --allow-empty -m "add clone method"
[feature 8e5ea57] add clone method

> git commit --allow-empty -m "fix bug in clone method"
[feature ba85899] fix bug in clone method

> git commit --allow-empty -m "add map mehtodd"
[feature 227bd10] add map mehtodd
```

```
> git commit --allow-empty -m "add reduce method"
[feature c19cabf] add reduce method

> git commit --allow-empty -m "get CI to work"
[feature e9e0a58] get CI to work

> git commit --allow-empty -m "more CI changes"
[feature edda363] more CI changes

> git commit --allow-empty -m "failing unit tests fixed"
[feature 868a671] failing unit tests fixed

> git commit --allow-empty -m "add deepCopy method"
[feature e75062c] add deepCopy method
```

```
> git log --oneline --graph
* e75062c (HEAD -> feature) add deepCopy method
* 868a671 failing unit tests fixed
* edda363 more CI changes
* e9e0a58 get CI to work
* c19cabf add reduce method
* 227bd10 add map mehtodd
* ba85899 fix bug in clone method
* 8e5ea57 add clone method
* 6fb33c0 (master) add spanish hello, goodbye
* 2fcf070 initial commit

> git rebase -i --keep-empty master
```

```
pick 8e5ea57 [Nick Nisi - 5 minutes ago] add clone method
pick ba85899 [Nick Nisi - 5 minutes ago] fix bug in clone method
pick 227bd10 [Nick Nisi - 4 minutes ago] add map mehtodd
pick c19cabf [Nick Nisi - 4 minutes ago] add reduce method
pick e9e0a58 [Nick Nisi - 4 minutes ago] get CI to work
pick edda363 [Nick Nisi - 4 minutes ago] more CI changes
pick 868a671 [Nick Nisi - 4 minutes ago] failing unit tests fixed
pick e75062c [Nick Nisi - 3 minutes ago] add deepCopy method
```

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
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
```

```
pick 8e5ea57 [Nick Nisi - 5 minutes ago] add clone method
fixup ba85899 [Nick Nisi - 5 minutes ago] fix bug in clone method
reword 227bd10 [Nick Nisi - 4 minutes ago] add map mehtodd
pick c19cabf [Nick Nisi - 4 minutes ago] add reduce method
fixup e9e0a58 [Nick Nisi - 4 minutes ago] get CI to work
fixup edda363 [Nick Nisi - 4 minutes ago] more CI changes
squash 868a671 [Nick Nisi - 4 minutes ago] failing unit tests fixed
pick e75062c [Nick Nisi - 3 minutes ago] add deepCopy method
```

```
> git log --oneline --graph
* 7ec4c60 (HEAD -> feature) add deepCopy method
* eed0261 add reduce method
* c68da08 add map method
* bc92a03 add clone method
* 6fb33c0 (master) add spanish hello, goodbye
* 2fcf070 initial commit
```

Keep only the commits that matter

Works for both rebase and merge strategies

Clean History Matters

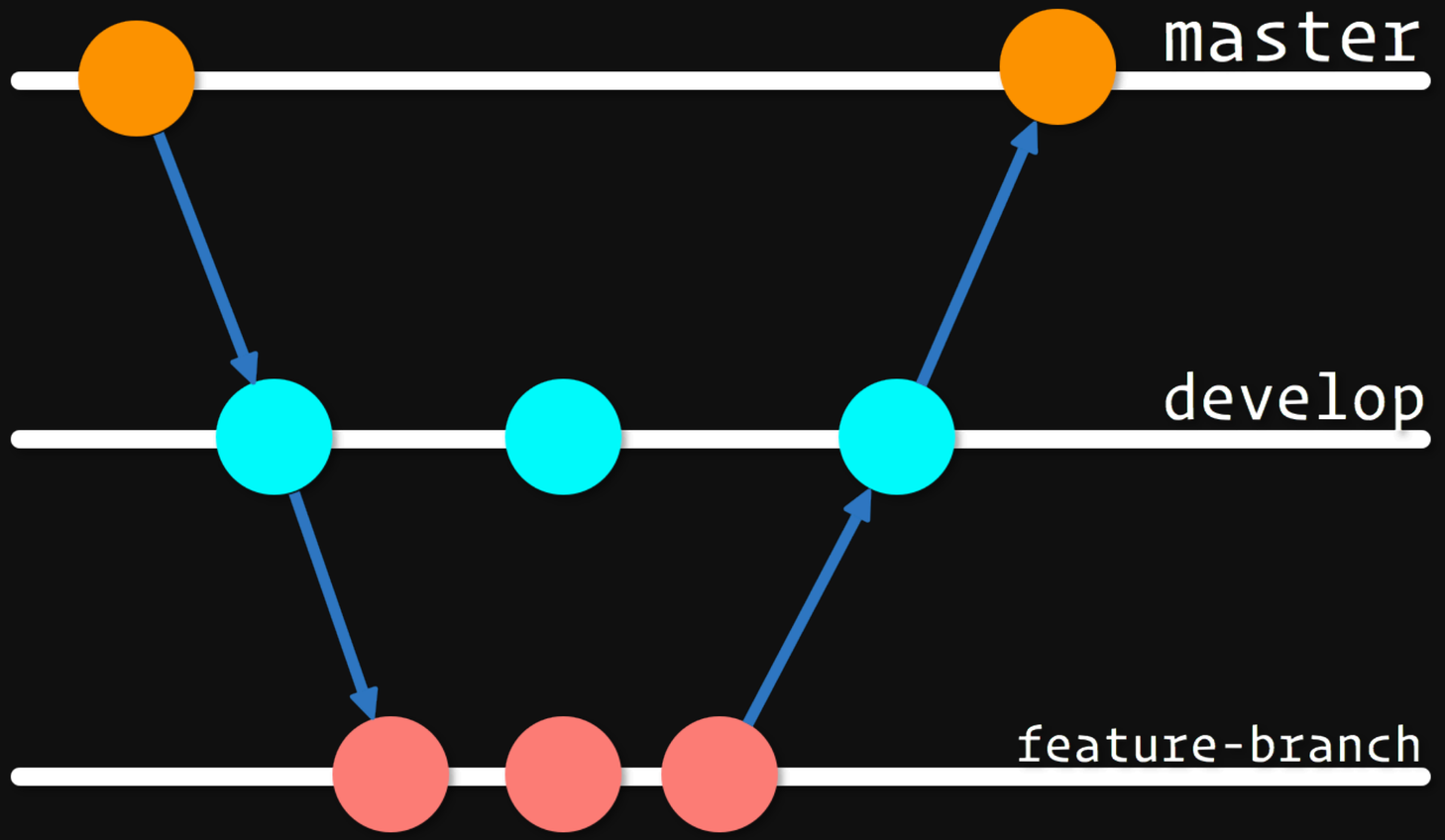
Clean History

- Tells the story of the project
- Approachable to new developers on the project
- Helps you utilize more git tools, easily
- Keeps you sane

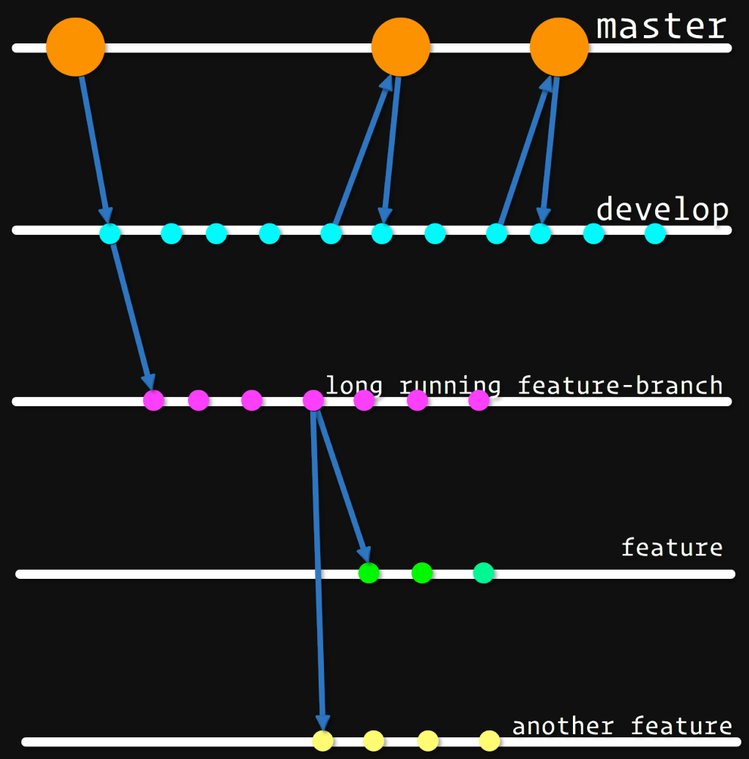
Your Git history should tell a chronological story of what ~~happened~~ *SHOULD HAVE HAPPENED*

Clean Branching Strategies

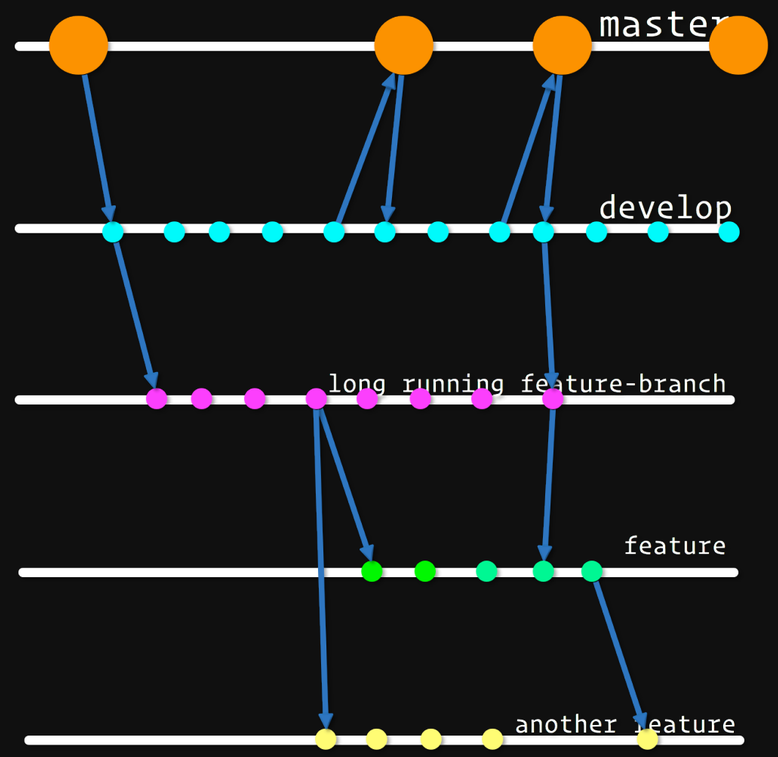
Typical Git Flow



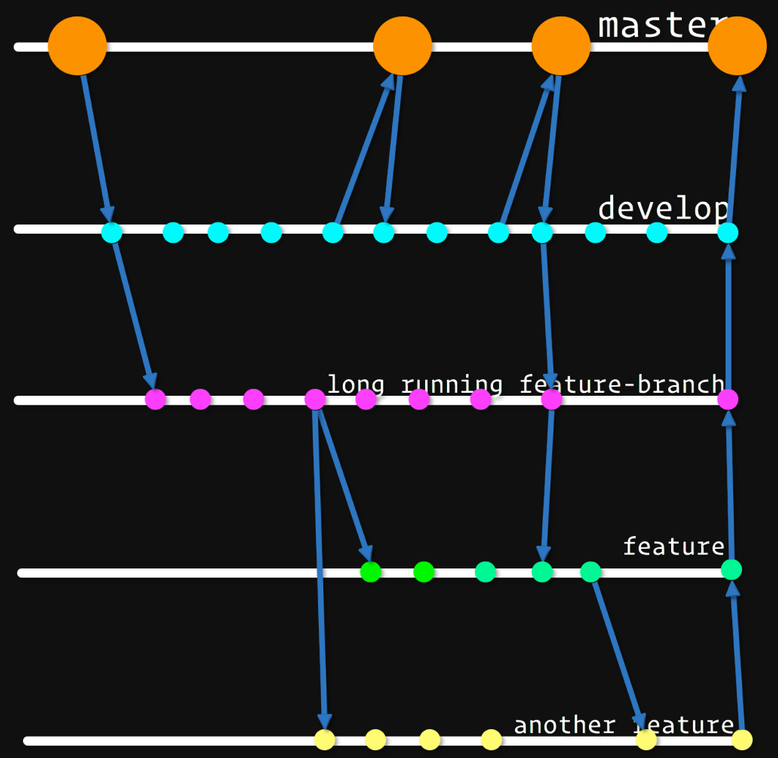
More Complex Flow

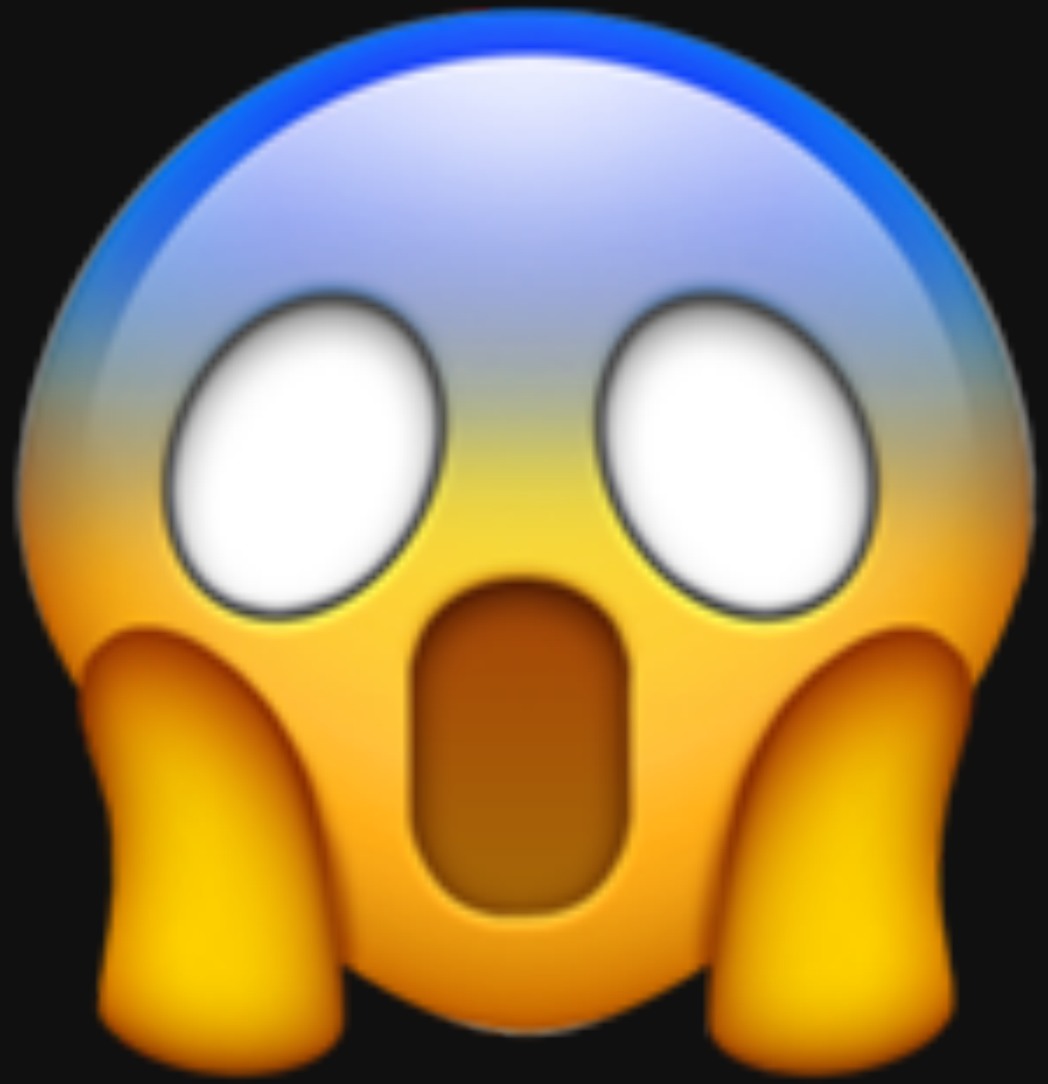


Complex Merging Down

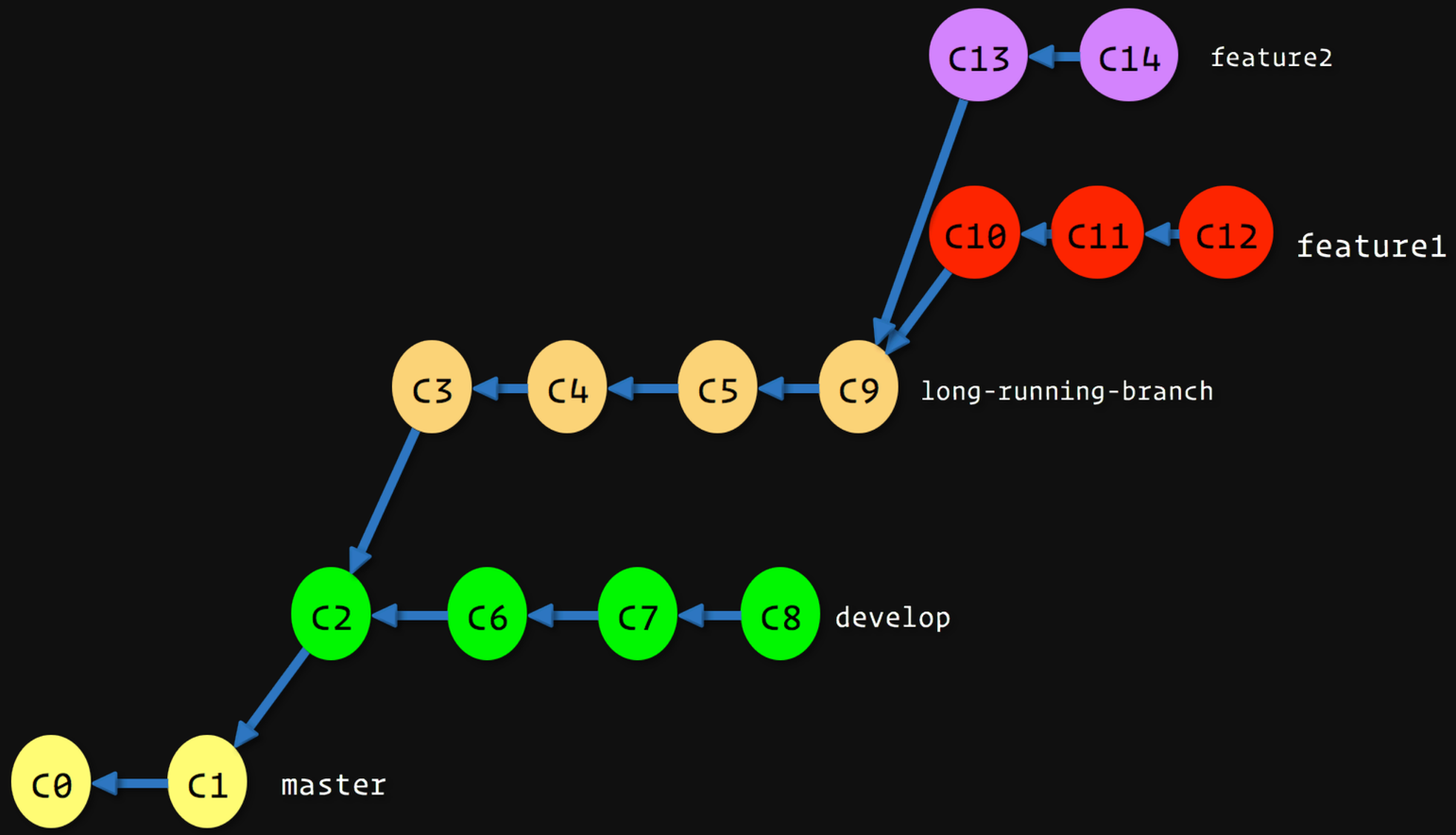


Merging Back Up





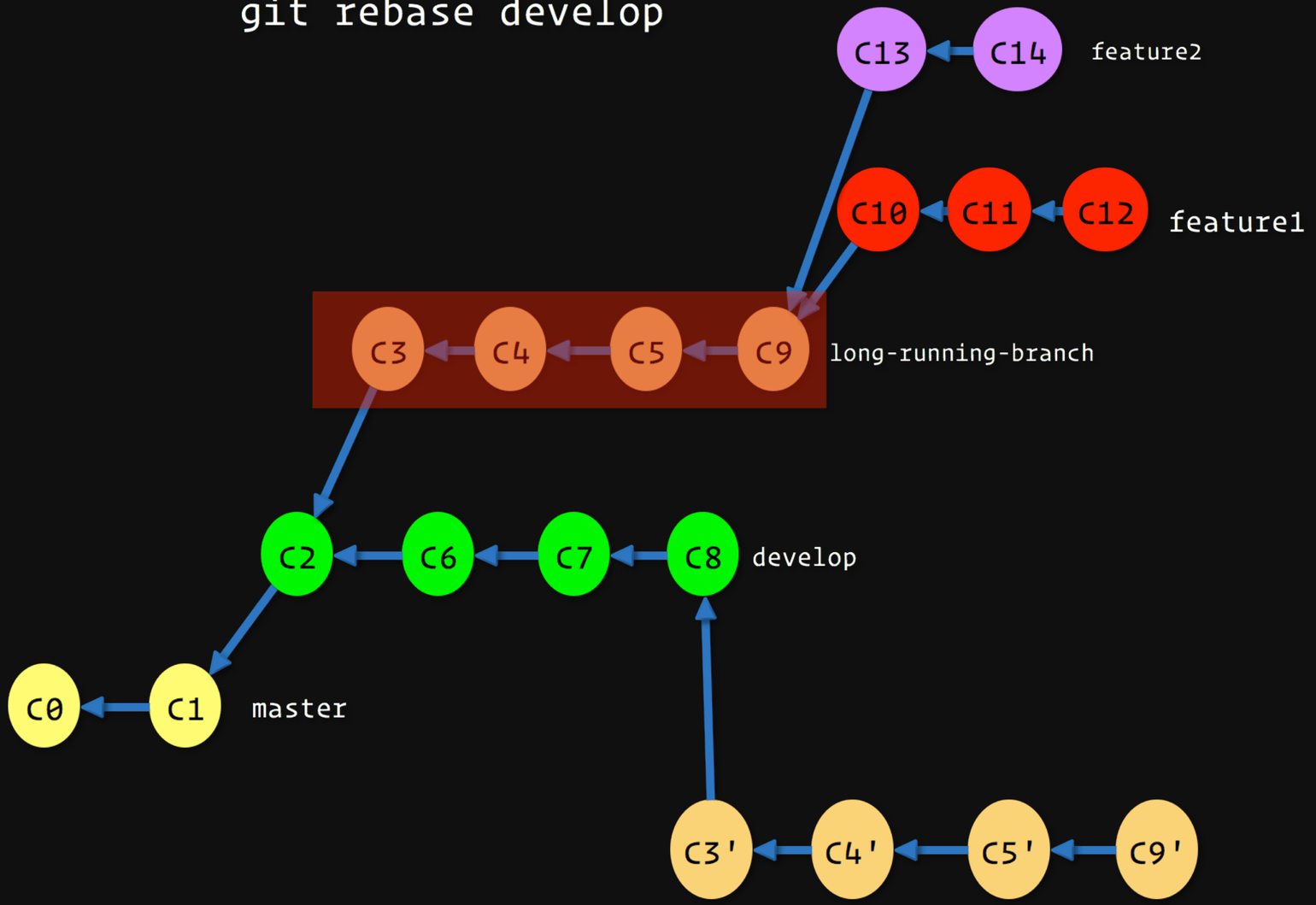
A Better Way With Rebase


C13
C14
feature2
C10
C11
C12
feature1
C3
C4
C5
C9
long-running-branch
C2
C6
C7
C8
develop
C0
C1
master

Let's rebase `long-running-branch` off the latest `develop`

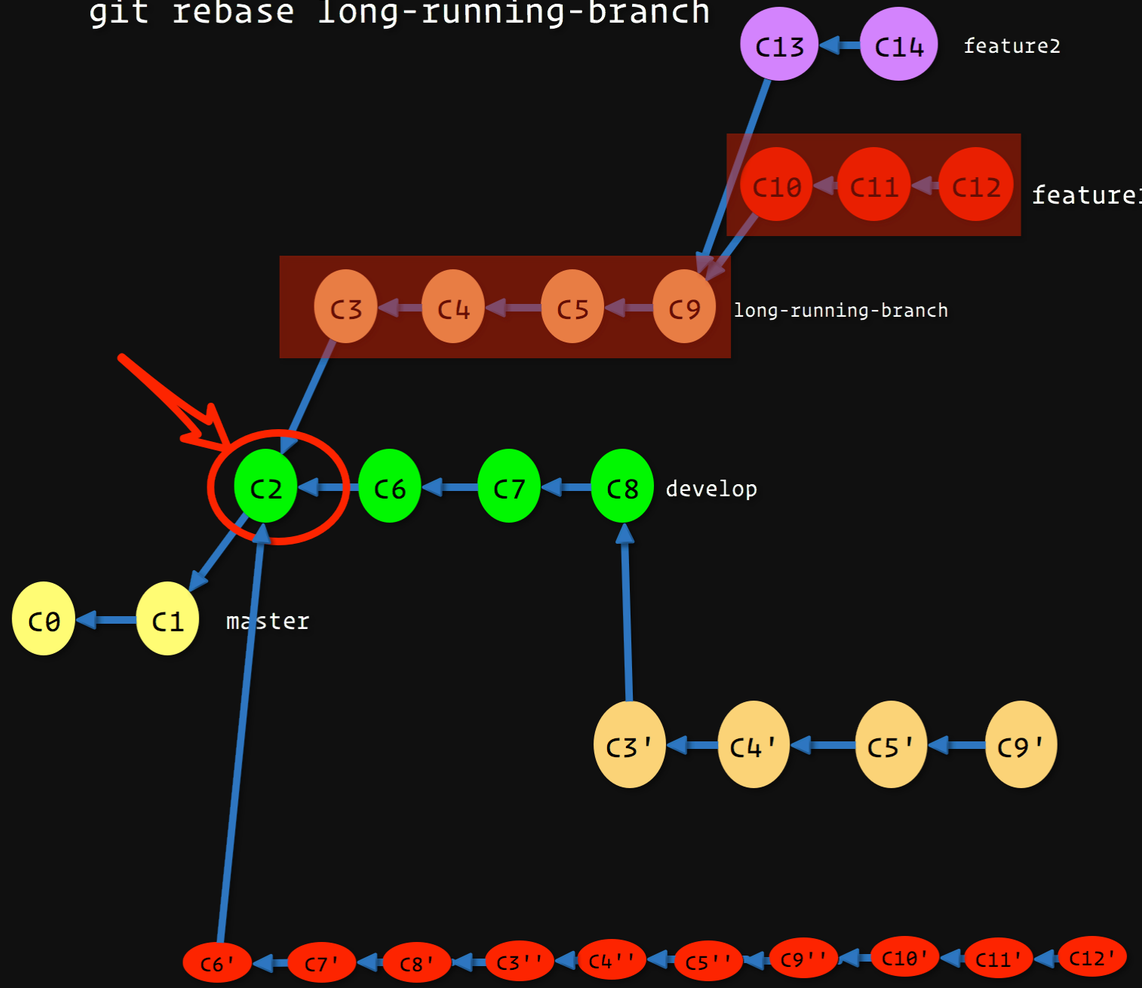
git checkout long-running branch

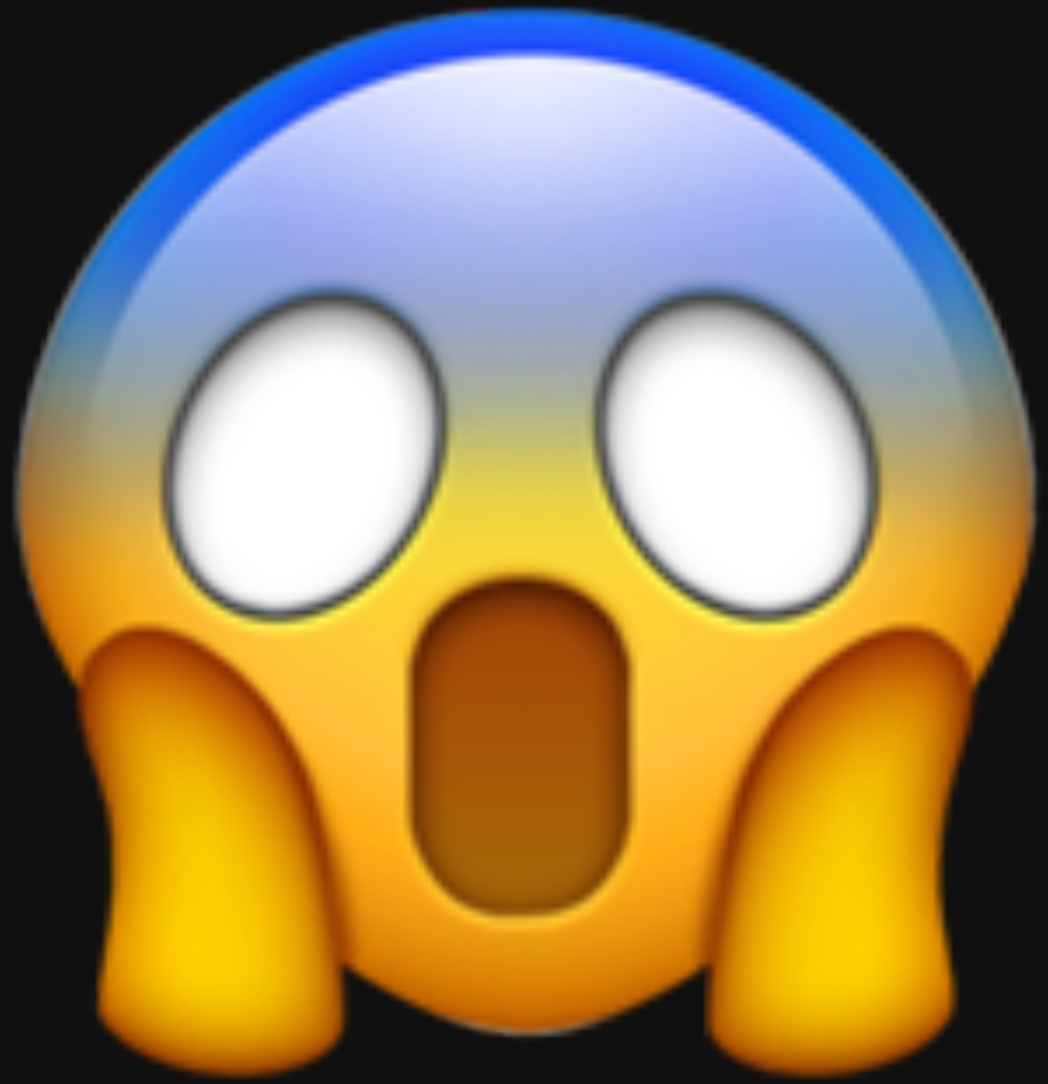
git rebase develop



Now let's rebase `feature1` off the latest `long-running-branch`

```
git checkout feature1

git rebase long-running-branch
```





Issues

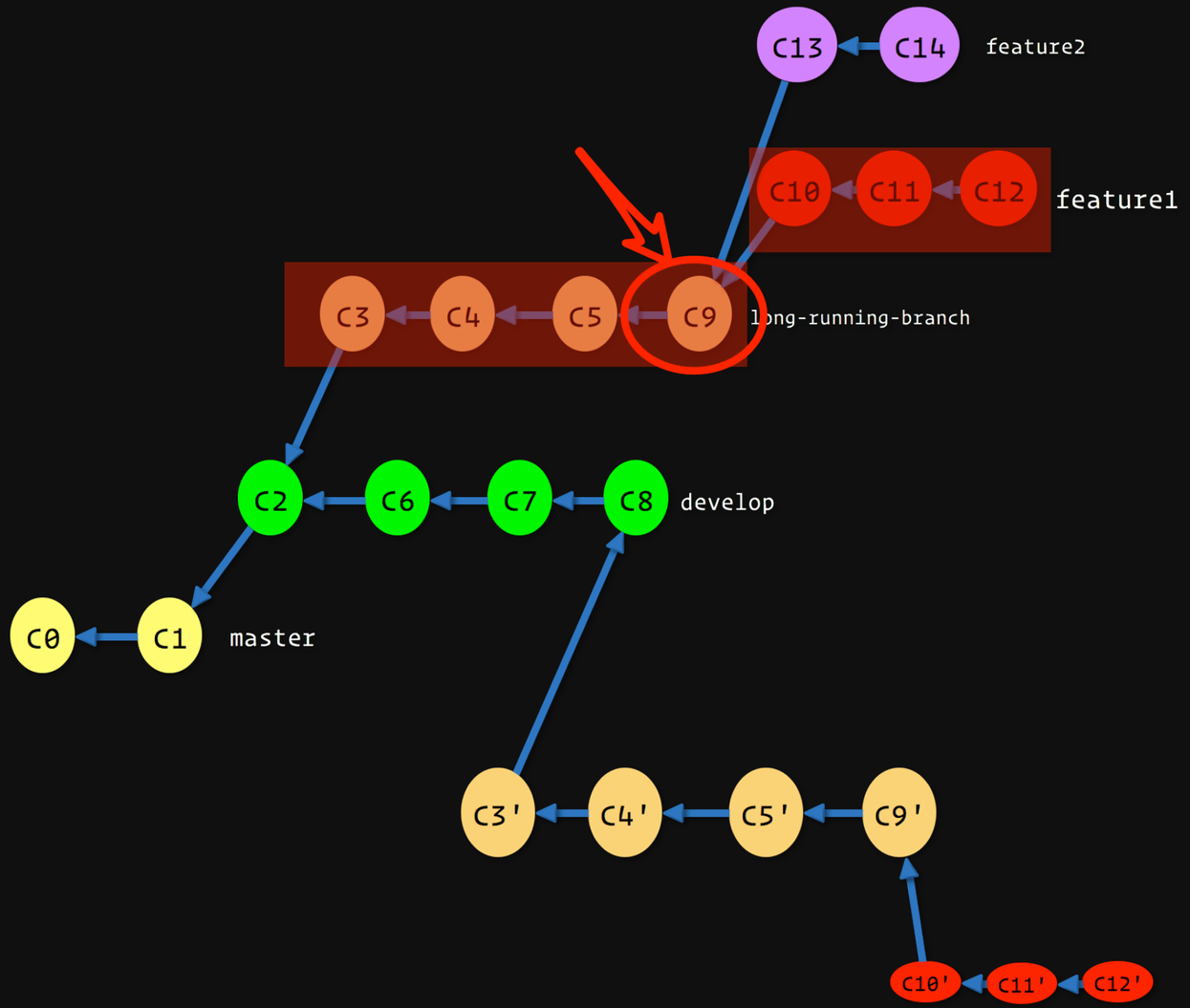
Git did not notice that `C9` and `C9'` are essentially the same commit. Instead, it saw the common commit between `feature1` and `long-running-branch` was `C2`.

To complete this rebase, Git will reapply all commits from `C2` through `C12`.

`git rebase --onto`

- Gives more control about what is being rebased and where
- `git rebase --onto <branch> <starting commit> <ending commit>`

git rebase --onto long-running-branch c9 feature1

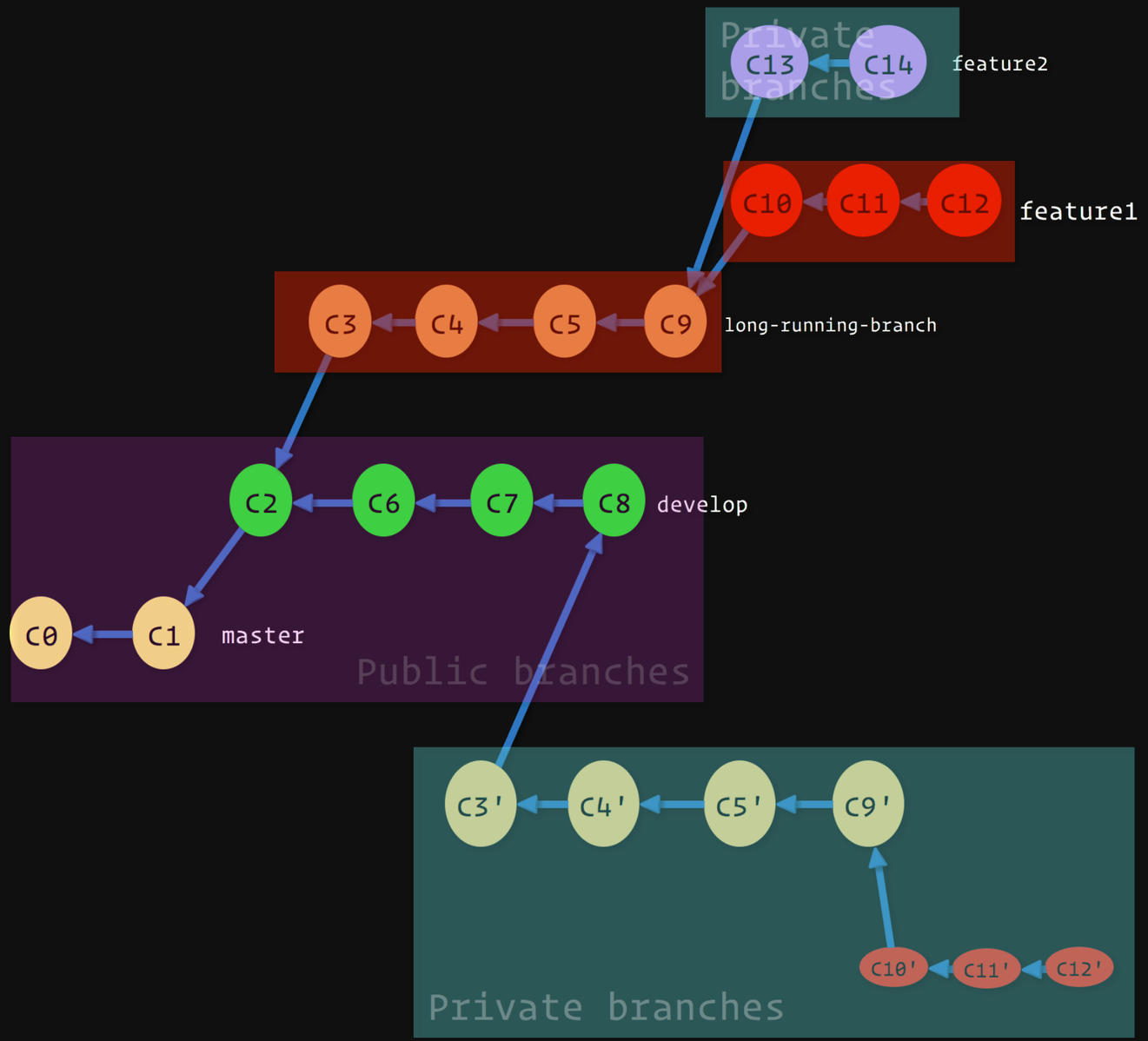


So Much Cleaner

We're only changing history on **private** branches

Never do this on a public branch

Always **communicate** with teammates


Private branches
C13
C14
feature2
C10
C11
C12
feature1
C3
C4
C5
C9
long-running-branch
C2
C6
C7
C8
develop
C0
C1
master
Public branches
C3'
C4'
C5'
C9'
C10'
C11'
C12'
Private branches

```
git push --force-with-lease
```

Section: `git bisect`

`git bisect`

- A helpful tool to find where something is different in the repository
  - Bugs
  - Bug fixes
- Drops the user into a chosen commit to perform a test (or it can be automated)
- User says whether the commit passes or fails with `good` / `bad`
- Binary search quickly lands the user on the commit that caused/fixed the bug

A Random Egg

```
#!/usr/bin/env bash

RAND=$(( ( RANDOM % 10 ) + 1 ))
FOUND=0
for i in {1..20}; do
    if [ "$i" -eq "$RAND" ]; then
        echo "HERE'S THE EGG" >| egg.txt
        FOUND=1
    elif [ "$FOUND" -eq "0" ]; then
        echo "NO EGG" >> egg.txt
    fi
    git commit --allow-empty -am "random commit $i"
done
```

```
> ./randomegg.sh

> git log --oneline --graph
* 6962087 (HEAD -> master) random commit 20
* 0158f2b random commit 19
* e399e03 random commit 18
* b2562f9 random commit 17
* 68eaf42 random commit 16
* 0cf152b random commit 15
* c6dbce2 random commit 14
* d4f978c random commit 13
* 9bc3b89 random commit 12
* b23efa3 random commit 11
* 4eab98d random commit 10
* e885a26 random commit 9
* 35331a2 random commit 8
* 7008e63 random commit 7
* 35a0b25 random commit 6
* 99722dc random commit 5
* c173293 random commit 4
* cad3986 random commit 3
* 43dd705 random commit 2
* bb5ca8f random commit 1
* 8cc5d6a initial commit
```

```
> git bisect start --term-old noegg --term-new egg

> git bisect noegg 8cc5d6a

> git bisect egg HEAD
Bisecting: 9 revisions left to test after this (roughly 3 steps)
[4eab98db42a0a4875689b2d390bc5314cfe8ac5c] random commit 10
```

```
> cat egg.txt
HERE'S THE EGG

> git bisect egg
Bisecting: 4 revisions left to test after this (roughly 2 steps)
[99722dc0c9b6c0ea865c3fd3b06829a3d9547db0] random commit 5

> cat egg.txt
there is no egg.

> git bisect noegg
Bisecting: 2 revisions left to test after this (roughly 1 step)
[7008e633682151a47d0c949682f9051b26e30a1b] random commit 7

> cat egg.txt
there is no egg.

> git bisect noegg
Bisecting: 0 revisions left to test after this (roughly 1 step)
[e885a26b178471750c286b5fb02fe5b511e68afd] random commit 9

> cat egg.txt
HERE'S THE EGG

> git bisect egg
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[35331a25aa3cd2f90e40fc88f0e297e668d933a9] random commit 8

> cat egg.txt
there is no egg.
```

```
> git bisect noegg
e885a26b178471750c286b5fb02fe5b511e68afd is the first egg commit
commit e885a26b178471750c286b5fb02fe5b511e68afd
Author: Nick Nisi <nick@nisi.org>
Date:   Wed Jul 11 03:25:23 2018 -0500

    random commit 9

:100644 100644 8f0174ba5b657d0fa26a5a3f67b153a885b0eaa5 84337fe8021dede703bdf196d1960815edd0f377 M       egg.txt
```

Exercise 03: Git Bisect

Find the egg.

If there's time

- filter-branch walkthrough
- reflog walkthrough

Wrapping up

- Git is a very powerful tool
- Don't be afraid to rewrite history
- History is actually pretty hard to lose

Thanks!