

Git Rebase Work Flow

How to minimize time wasted when committing code



Typical Git Commit Flow

Write some code

Self explanatory: new feature is defined and coding begins

Commit to branch

Code is updated in the Git repo as a series of commits

Merge branch to master

Feature is completed, code is finalized, and it is brought back into master

Git usage based on number of developers

Solo Developer

- Commit directly to master, maybe a branch every now and then
- Almost no conflicts
- Does not require co-ordination between teammates, because you have none
- No wait time between merges

2-4 Developers

- Start to use branching strategy
- Start to see conflicts
- Some wait time between branch merges
- Git history begins to get complex but still manageable

5+ Developers

- Lots of branches clutter git history
- Conflicts begin to rise quickly, and each teammate spends a lot of time resolving or maybe 1 developer dedicated to merging (usually senior)
- If one dev dedicated to merge, person becomes bottleneck and this process won't scale

Solo Developer Git History -- `general_spider` Repository

```

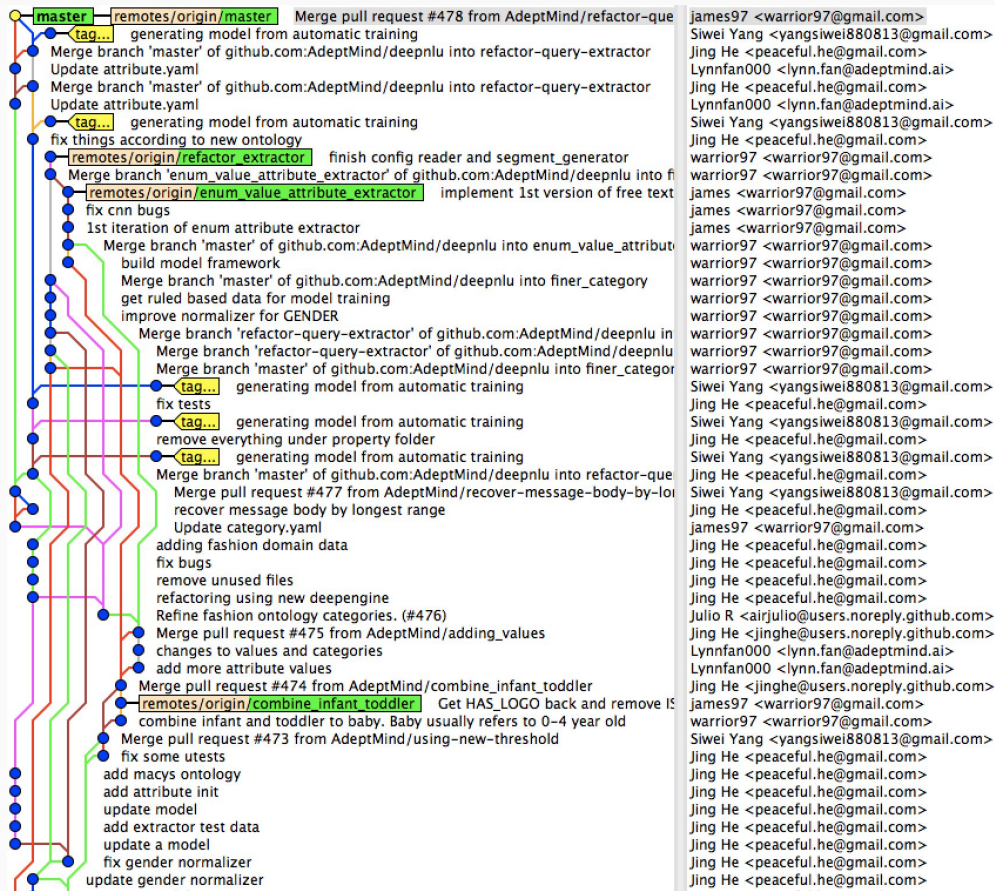
graph LR
    master[master] --- remotes/remotes/origin/master
    remotes/remotes/origin/master --- add[add canadagoose]

```

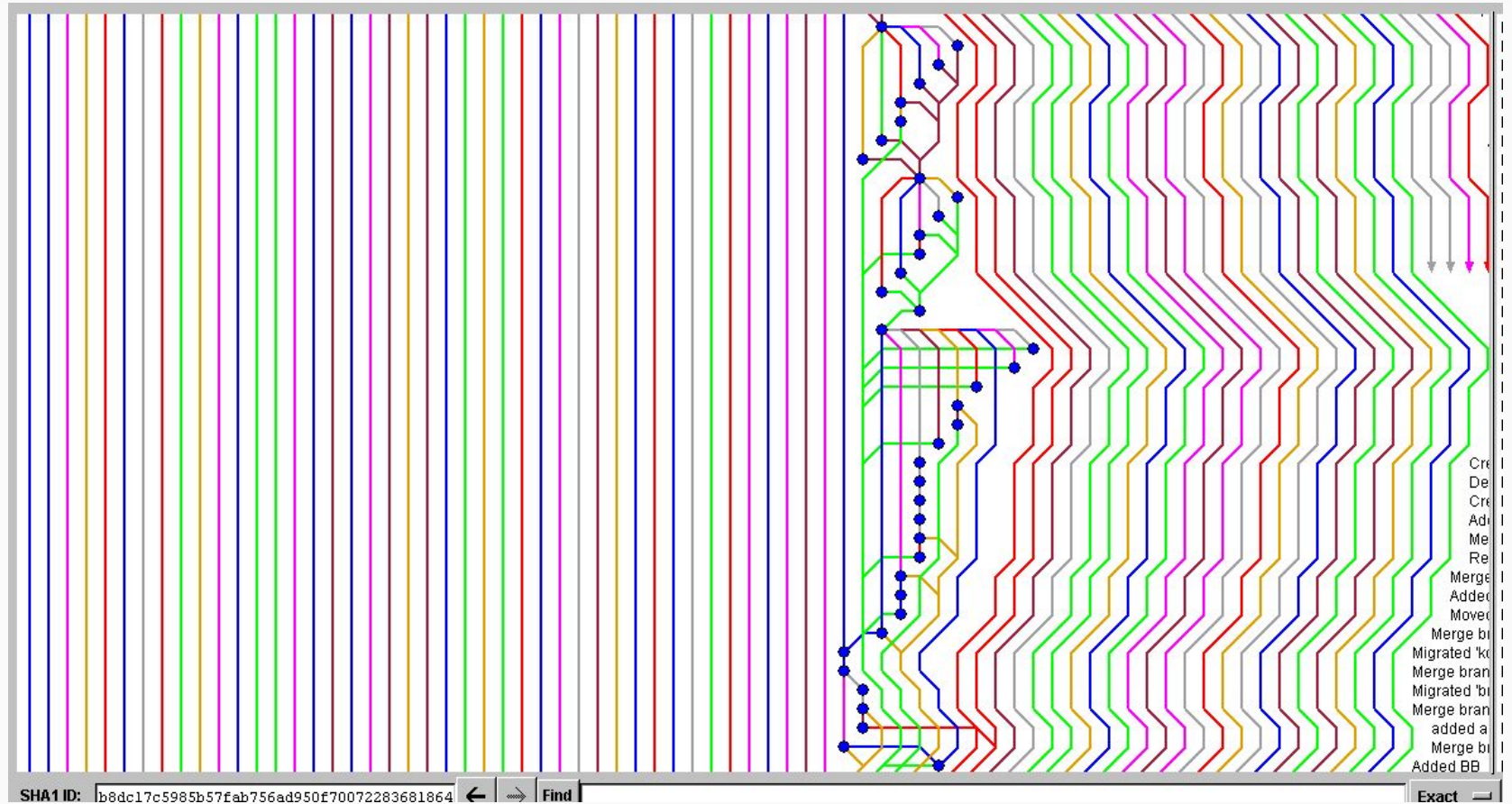
- fix config spider
- add new spiders
- fix bug
- avoid invalid keys
- merging
 - fix
 - extracting configurations
 - prepare for another round of extraction
 - do local color normalization
 - update persistence
 - update color extractor
 - working on color
 - add facts resolver
 - fix color extractor
 - command-line fix
 - got macys colors
 - fix filter url
 - fix routes management; apply url filter universally
 - labeler fix
 - primitive labeler
 - remotes/origin/WIP wip
 - move to ubuntu
 - adding random review text to test set
 - adding search expansion
 - remove redundant url management
 - updated response checking
 - adopt the document model
 - add primary url index; move broken page out of tests
 - split amazon spider; add extraction tests
 - extraction optimization
 - further optimization
 - performance improvements
 - no url expansion for now
 - upgrading amazon search crawler with separate storage and search group management
 - price extractor update
 - update extraction storage
 - fix extraction
 - update
 - mre restrictive link chasing
 - continues on to property extraction
 - decoding search results
 - add noahsnaturalfoods; cosmetic fixes
 - macys api decoded; speed fix
 - deal with page inconsistency
 - fix
 - db as dependency injection
 - much more sites; generalized evaluator
 - first commit with tests

[illegible]

Git Merge History for 2-4 Developers -- `deepnlu` repository



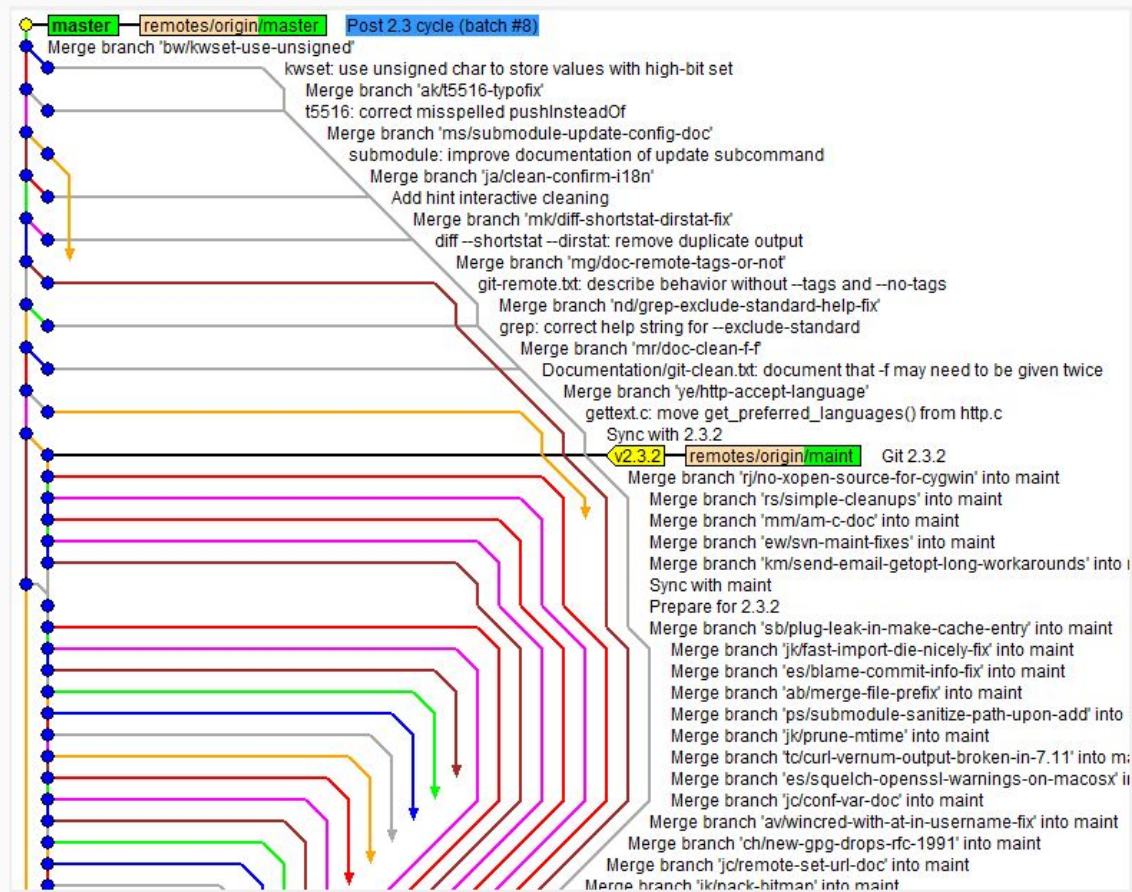
Potential Chaos Scenario With 5+ Developers



nononononononononononon

Wat.

Git Merge -- Time consumption approaches infinite



Solution

Use

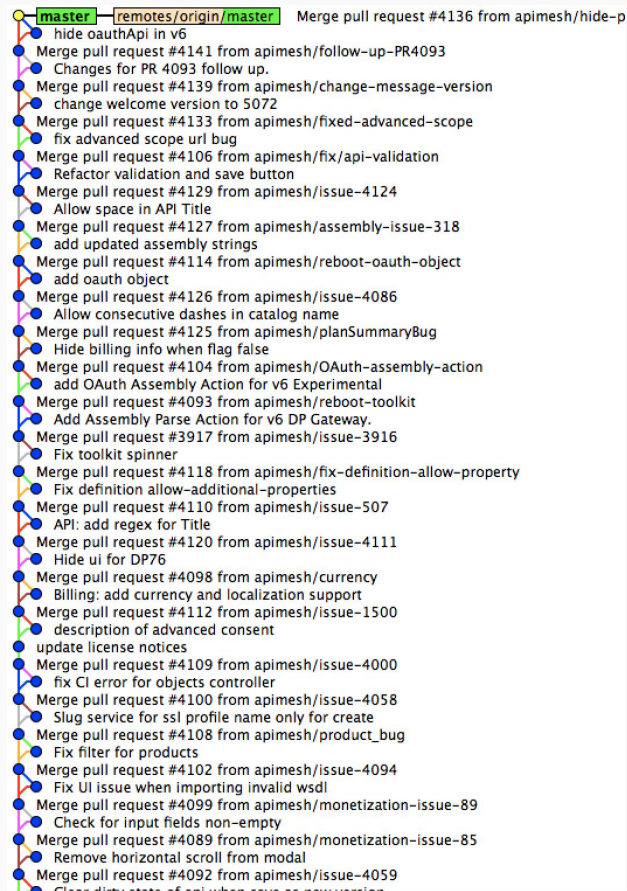
```
`git pull --rebase origin master`
```

or

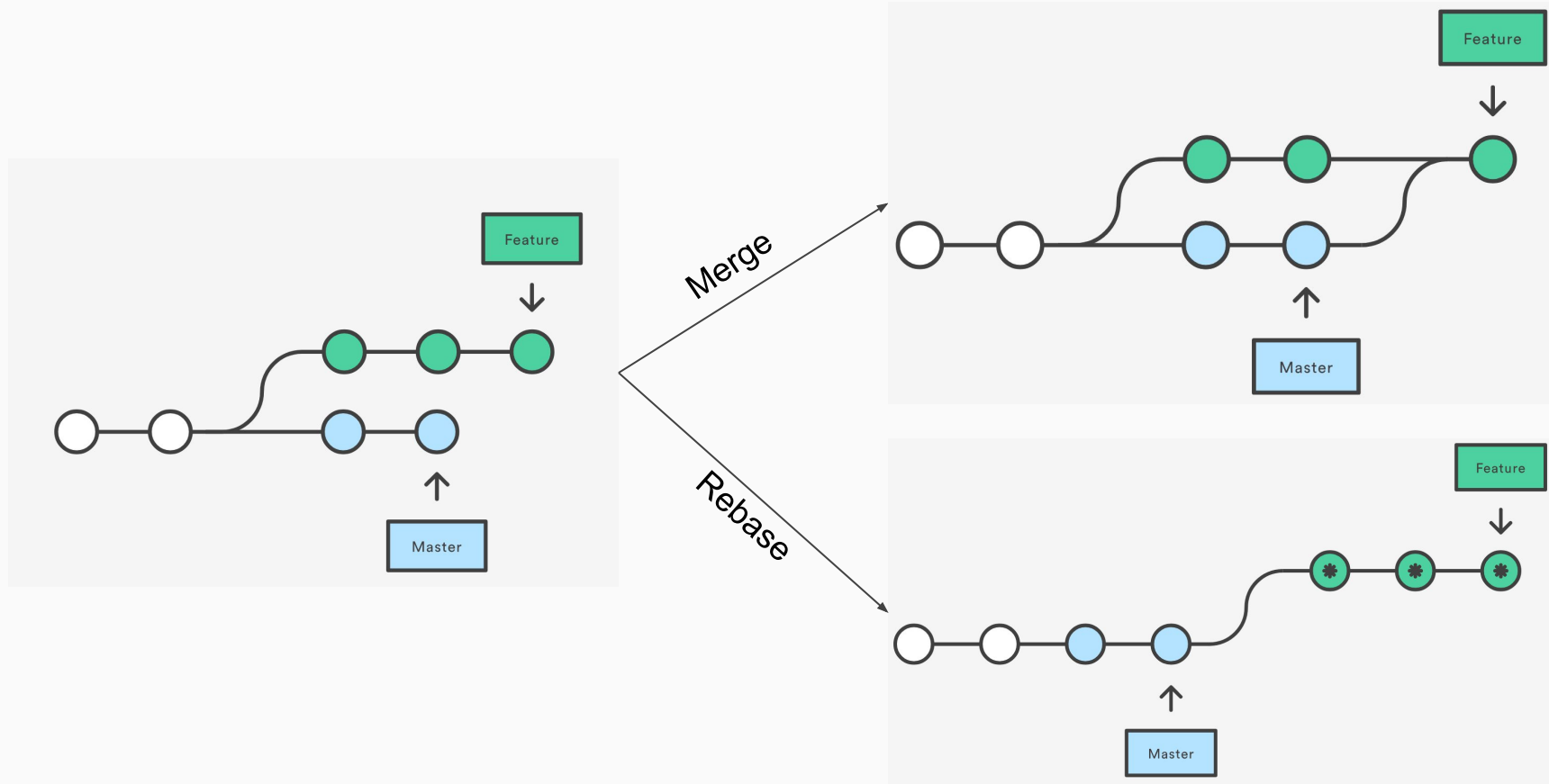
```
`git rebase -i origin/master`
```

- Linear history based on merge
- Easy to handle product release, backporting, and versioning
- Conflicts resolved by single developer, everyone else gets for free (most of the time)
- Scales to infinite developers, proven by Open-Source community

Git Rebase Work-Flow: A Team Effort

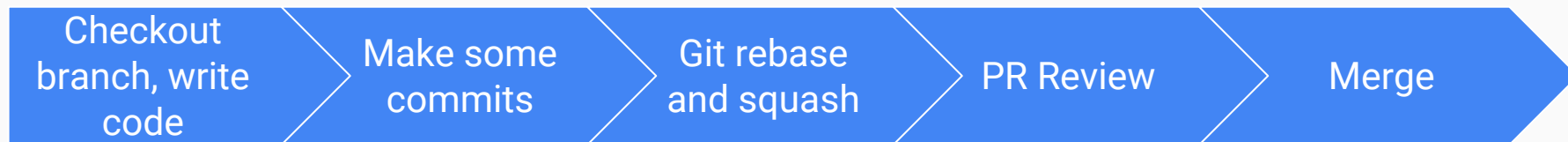


What is Git Rebase?



Git Rebase Flow

Changes history, requires
a forced push



Swappable
(if you know what you're doing)

TEAM SPIRIT

(Do some demos?)