Git Real

@dylanninin

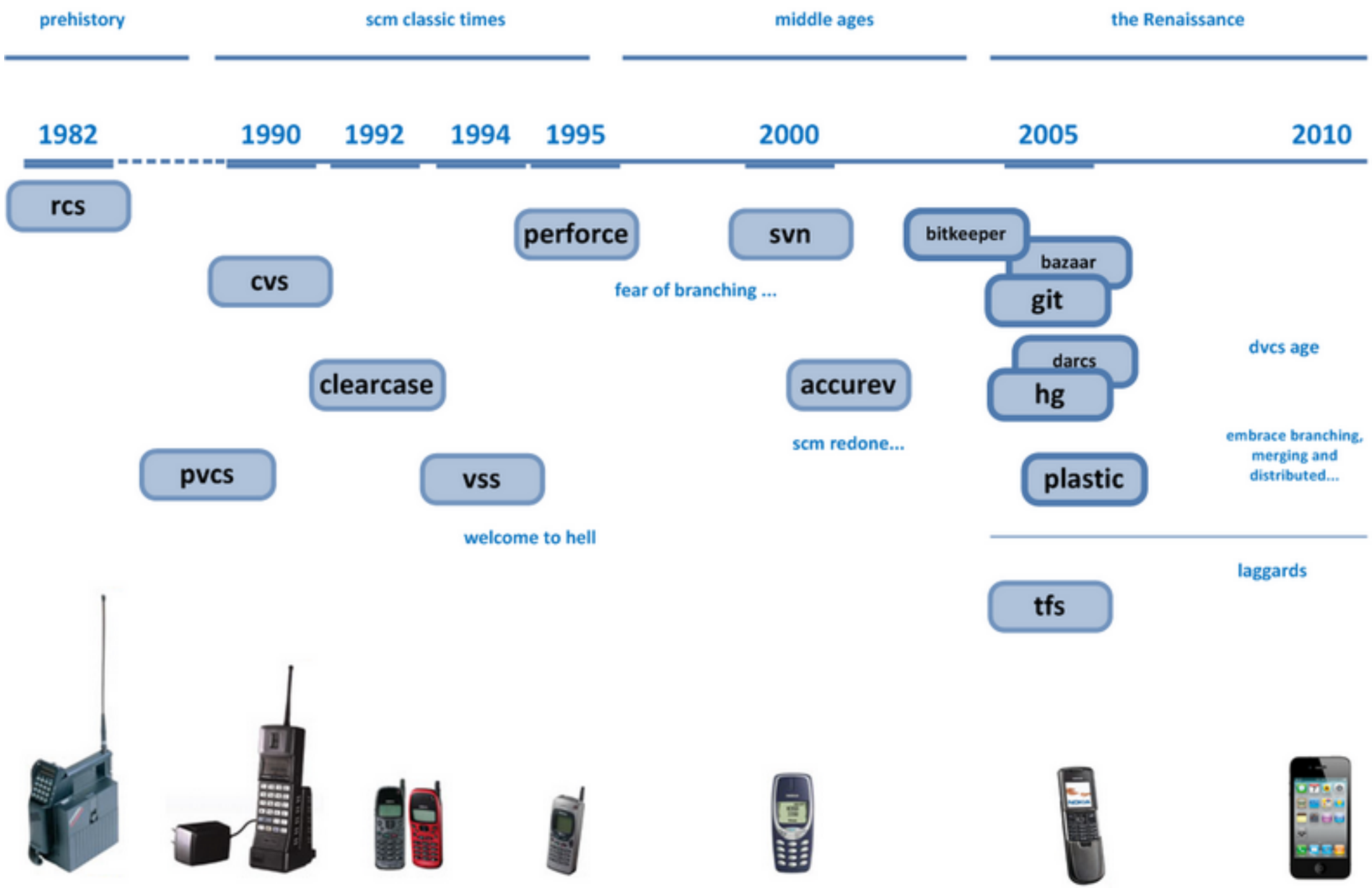
@2016-02-14

- **What is Git**
- **Internals**
- **Play with Git**
- **Workflow**
- **Q&A**

What is Git

A long time ago

SCM Timeline



Git != Subversion + Magic!

Git != Subversion + Magic!

- Git is a stupid content tracker.
- It is simply used as an SCM, not really designed as one.
- In many ways you can just see git as a file system

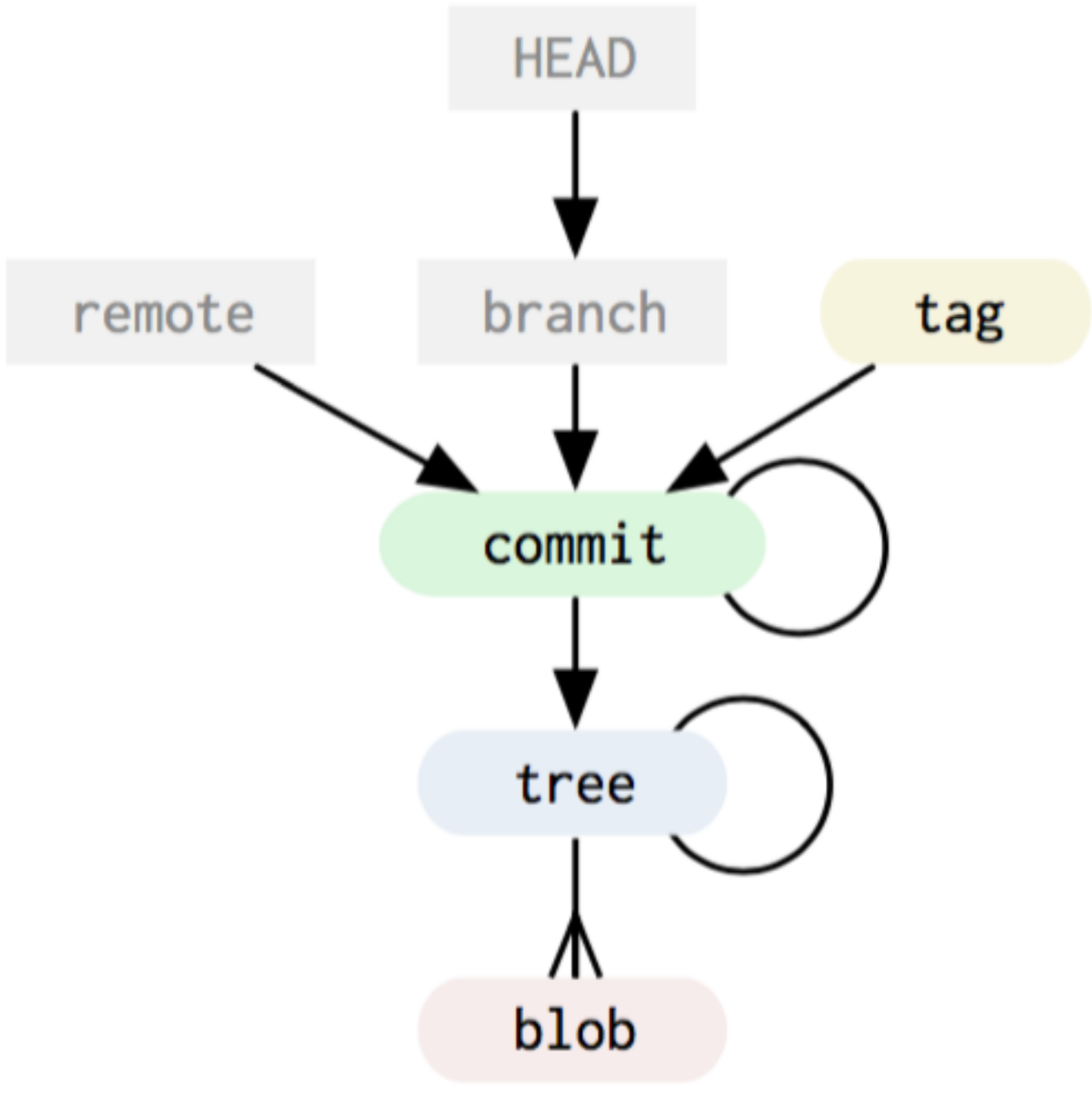
> “In many ways you can just see git as a file system — it’s content- addressable, and it has a notion of versioning, but I really really designed it coming at the problem from the viewpoint of a file system person (hey, kernels is what I do), and I actually have absolutely zero interest in creating a traditional SCM system.”
>
> – Linus (http://marc.info/?l=linux-kernel&m=111314792424707)

Git Design

- Non-Linear Development
- Distributed Development
- Efficiency

Git Internals

Internals - The Model



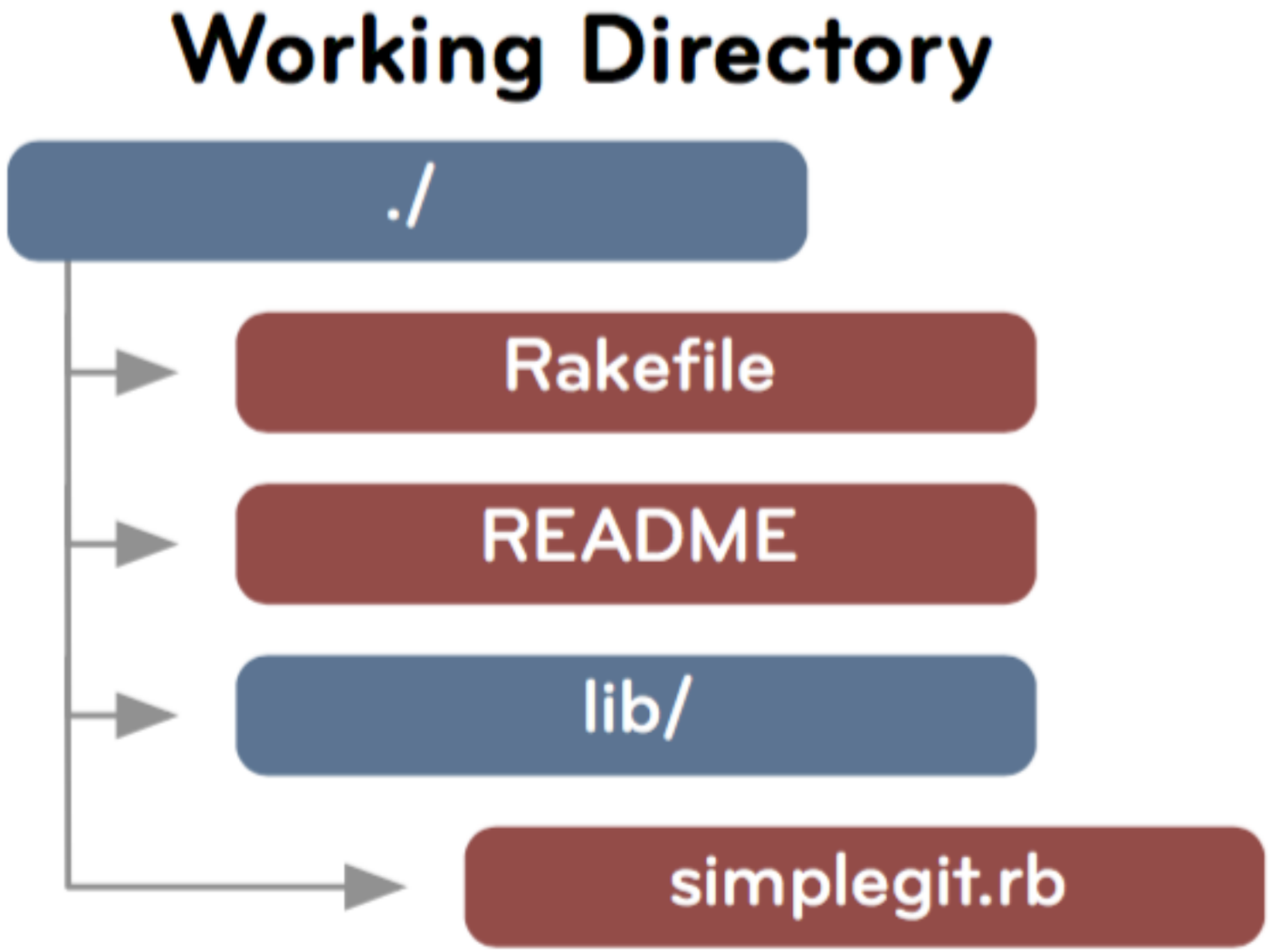
Internals - Git Objects

- Git objects are the actual data of Git, the main thing that the repository is made up of

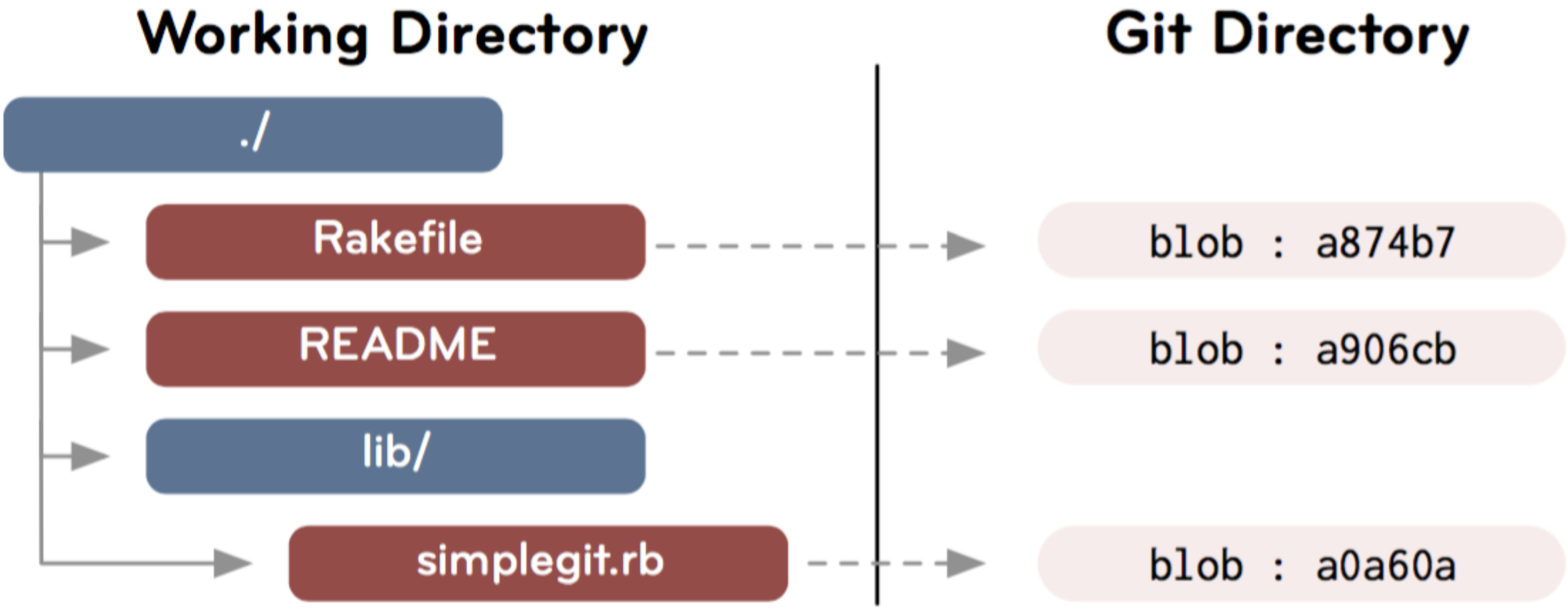
- Each object is compressed (with Zlib) and referenced by the SHA-1 value of its contents plus a small header:
  - full SHA-1: dae86e1950b1277e545cee180551750029cfe735
  - partial SHA-1: dae86e

- There are four main object types in Git:
  - Blob
  - Tree
  - Commit
  - Tag

Internals - Working Directory



The Blob


Working Directory
Git Directory
./
Rakefile
README
lib/
simplegit.rb
blob : a874b7
blob : a906cb
blob : a0a60a

The Blob

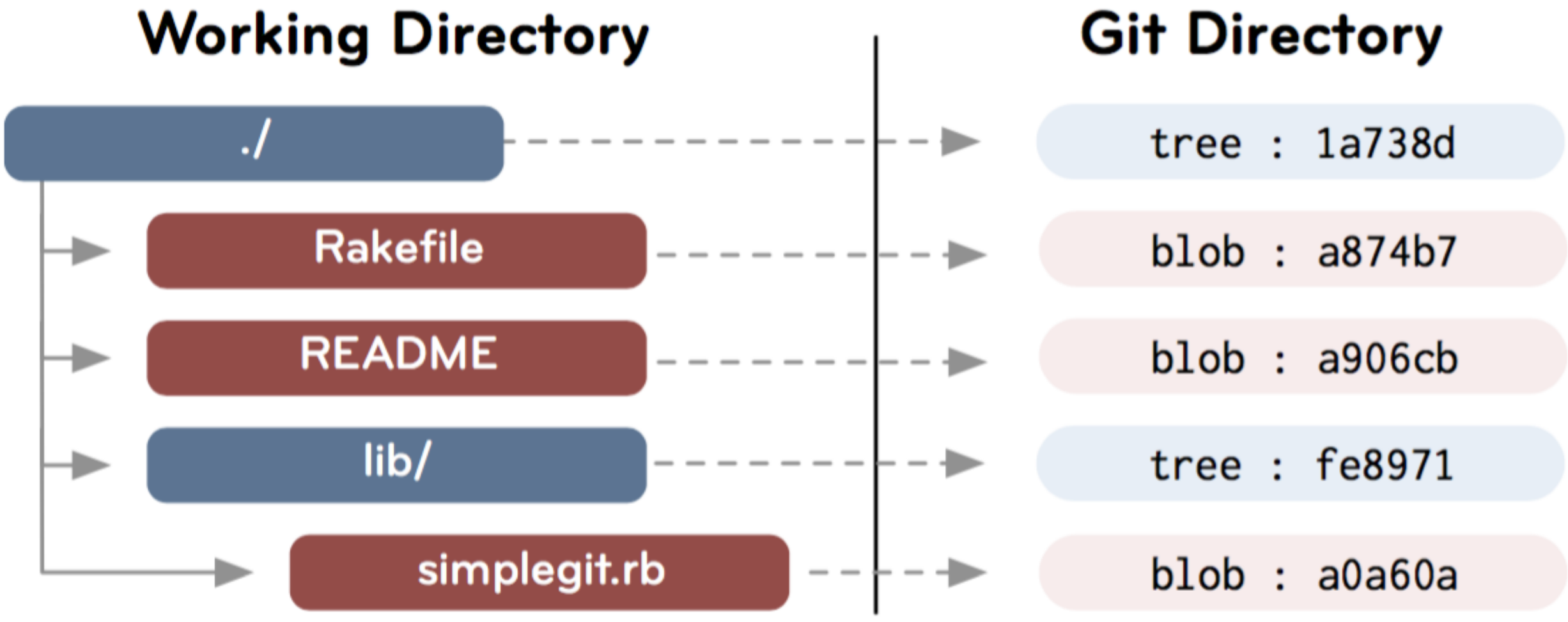
```
blob [content size]\0
```

```
SimpleGit Ruby Library
======================

This library calls git commands and
returns the output.

Author : Scott Chacon
```


Zlib::Deflate

```
blob : a906cb
```

The Tree


Working Directory
Git Directory
./
Rakefile
README
lib/
simplegit.rb
tree : 1a738d
blob : a874b7
blob : a906cb
tree : fe8971
blob : a0a60a

The Tree

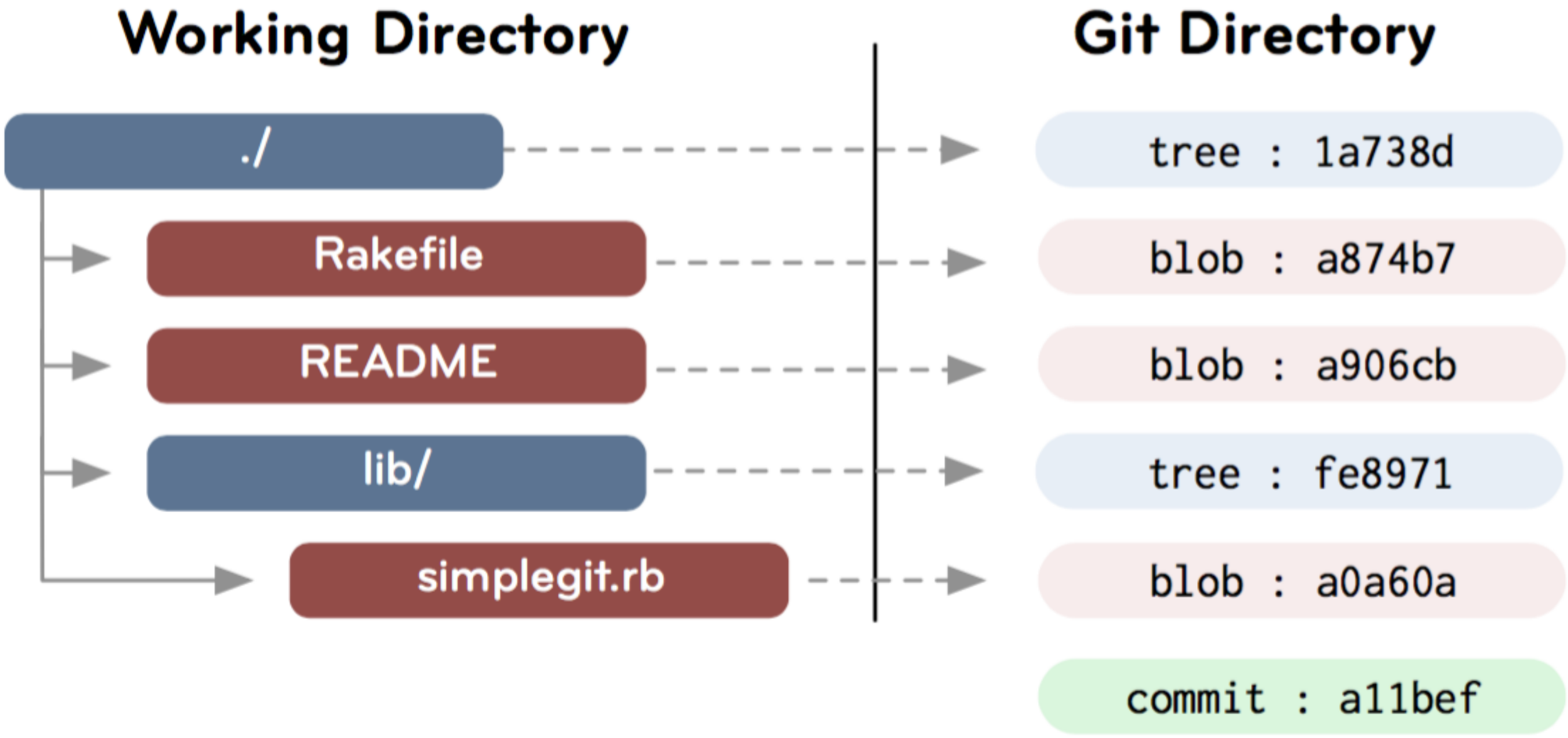
```
tree [content size]\0
```

```
100644 blob a906cb    README
100644 blob a874b7    Rakefile
040000 tree fe8971    lib
```

Zlib::Deflate

```
tree : 1a738d
```

The Commit


Working Directory
Git Directory
./
Rakefile
README
lib/
simplegit.rb
tree : 1a738d
blob : a874b7
blob : a906cb
tree : fe8971
blob : a0a60a
commit : a11bef

The Commit

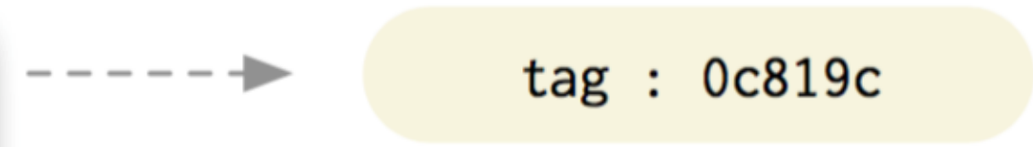
```
commit [content size]\0
tree 1a738d
author Scott Chacon
        <schacon@gmail.com> 1205602288
committer Scott Chacon
        <schacon@gmail.com> 1205602288

first commit
```

```
commit : a11bef
```

The Tag

```
tag [content size]\0
```

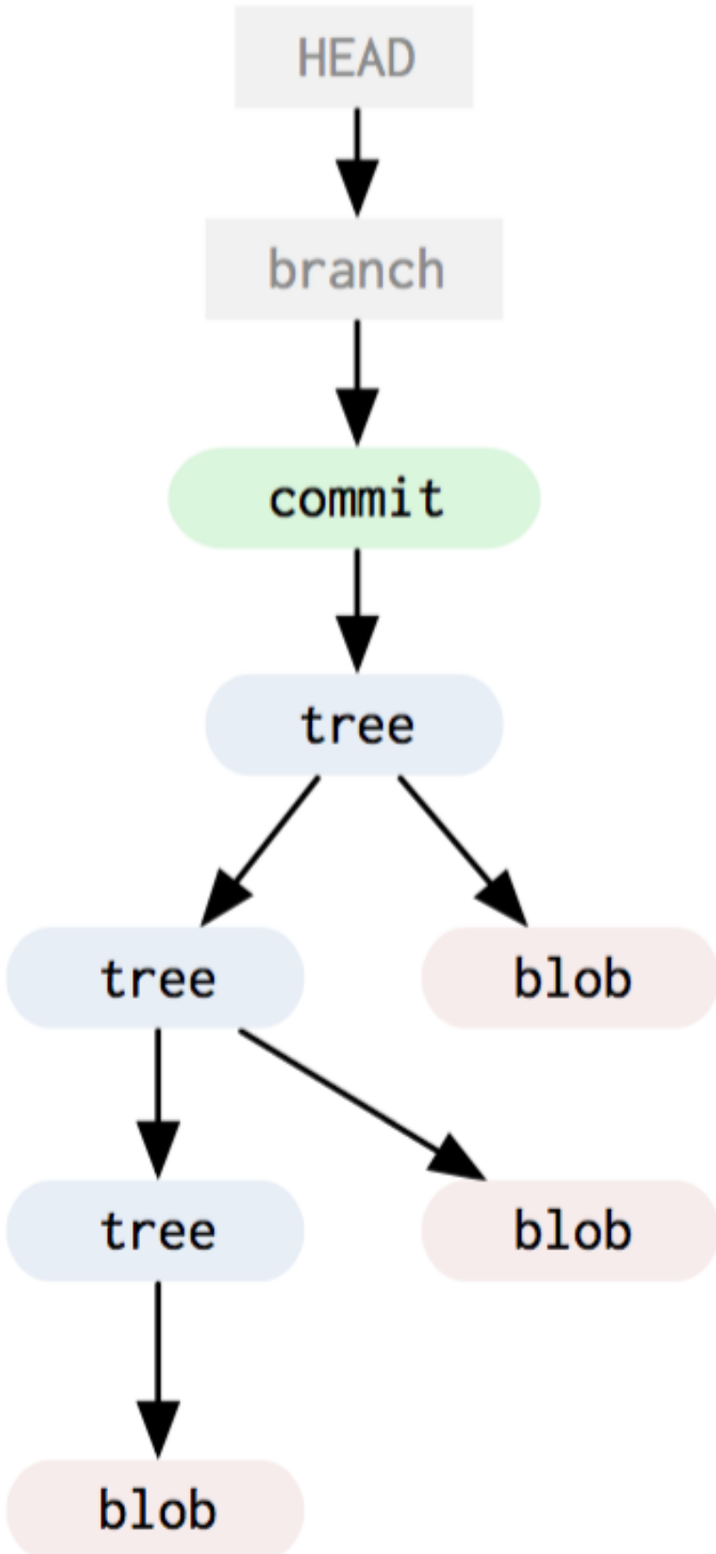
```
object 0576fa
type commit
tag v0.1
tagger Scott Chacon
        <schacon@gmail.com> 1205624655

this is my v0.1 tag
```



Internals - Example

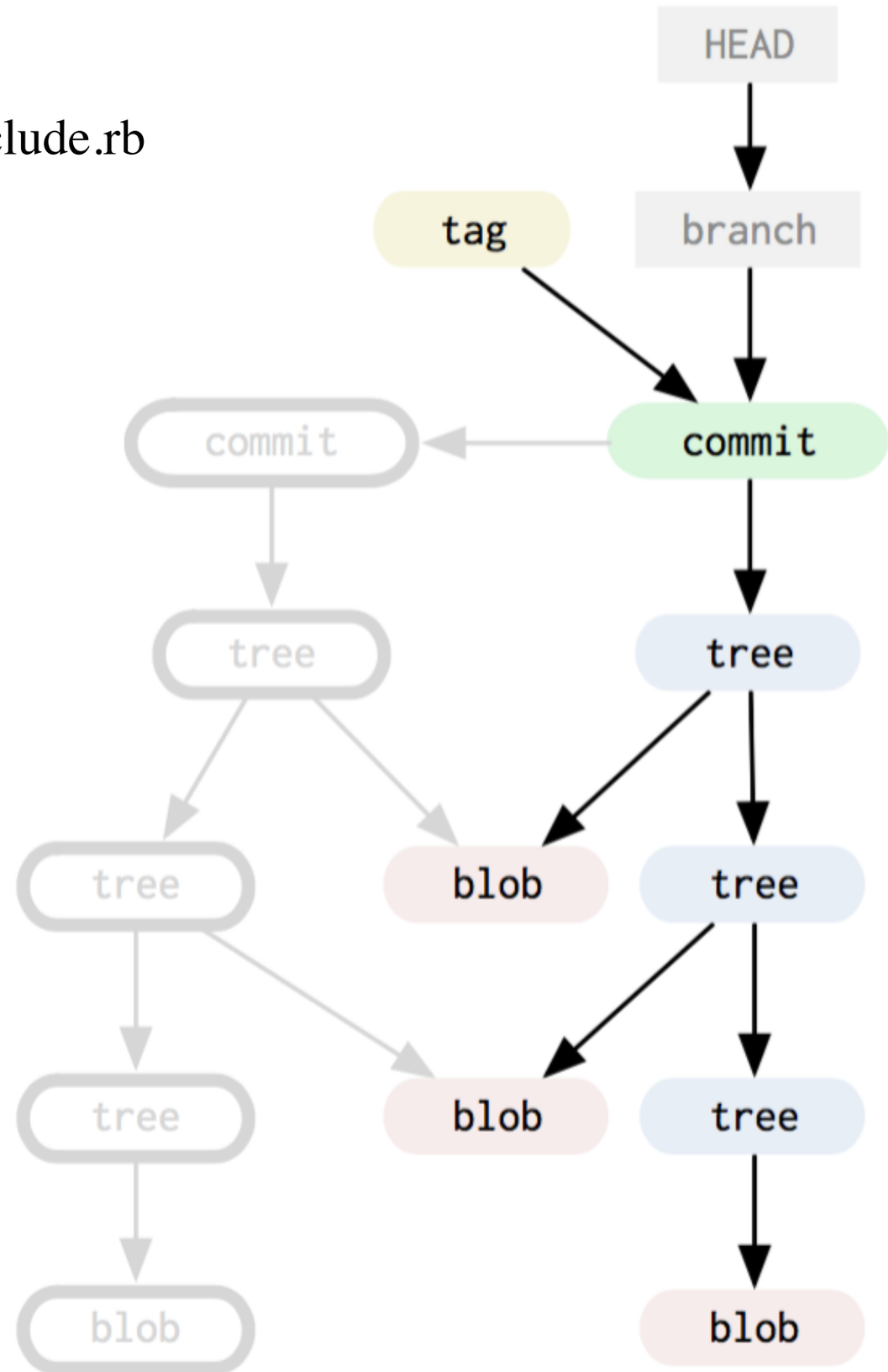
- current repository with one commit
- and its' logical structure

```
.
|-- init.rb
`-- lib
    |-- base
    |   `-- base_include.rb
    `-- my_plugin.rb
```



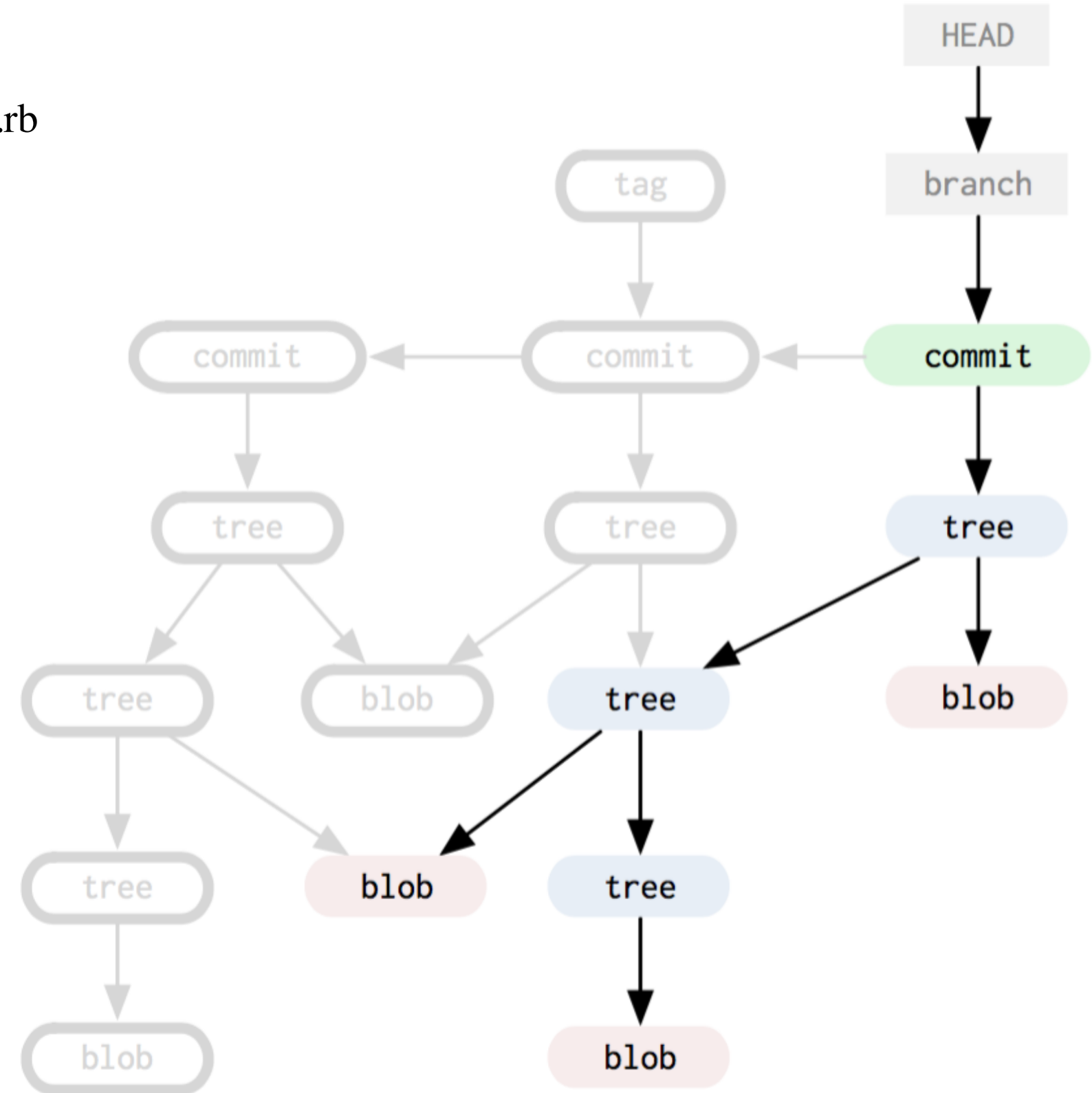
Internals - Example

- modify lib/base/base_include.rb
- add a release tag v0.1



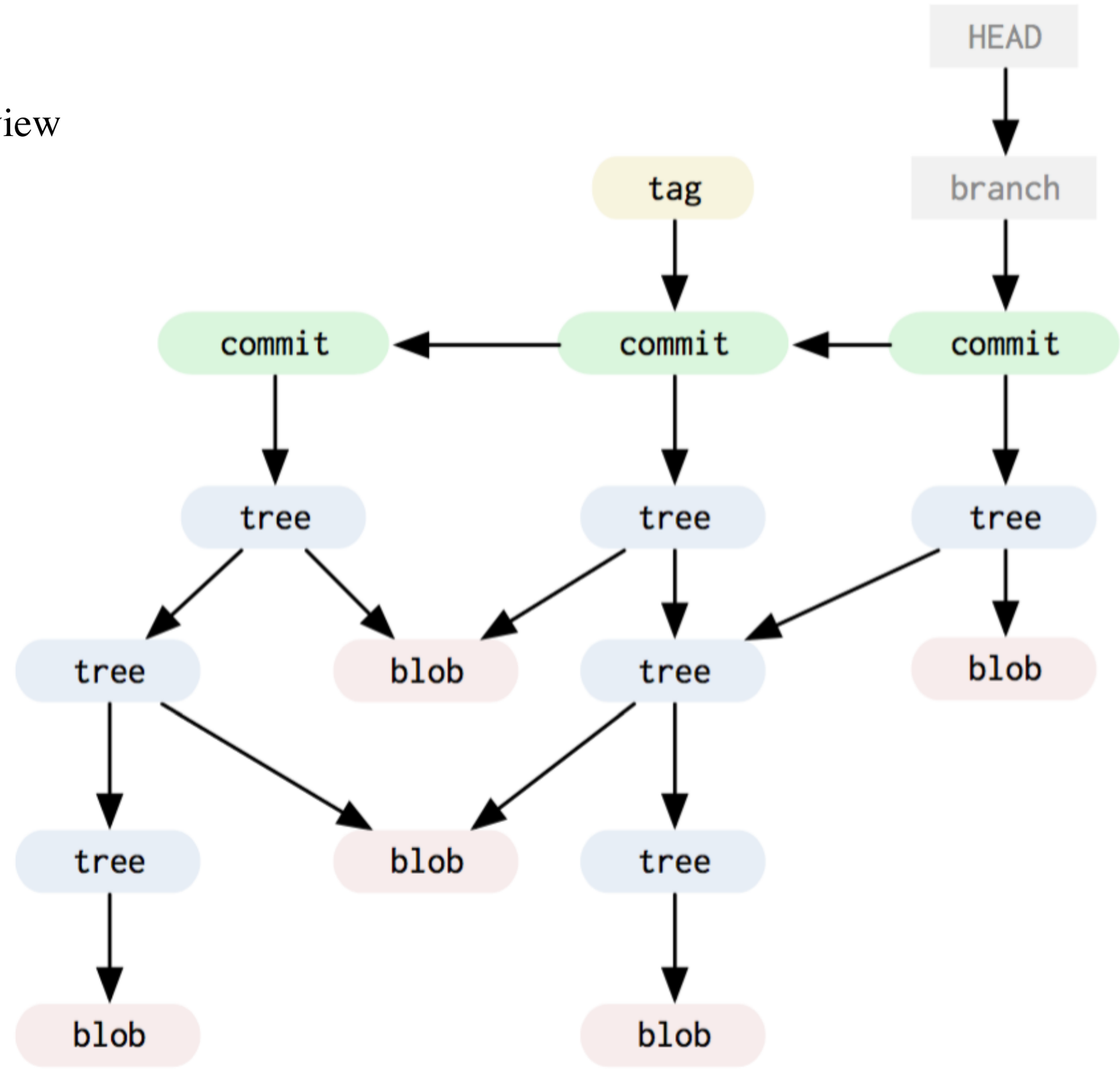
Internals - Example

- modify init.rb



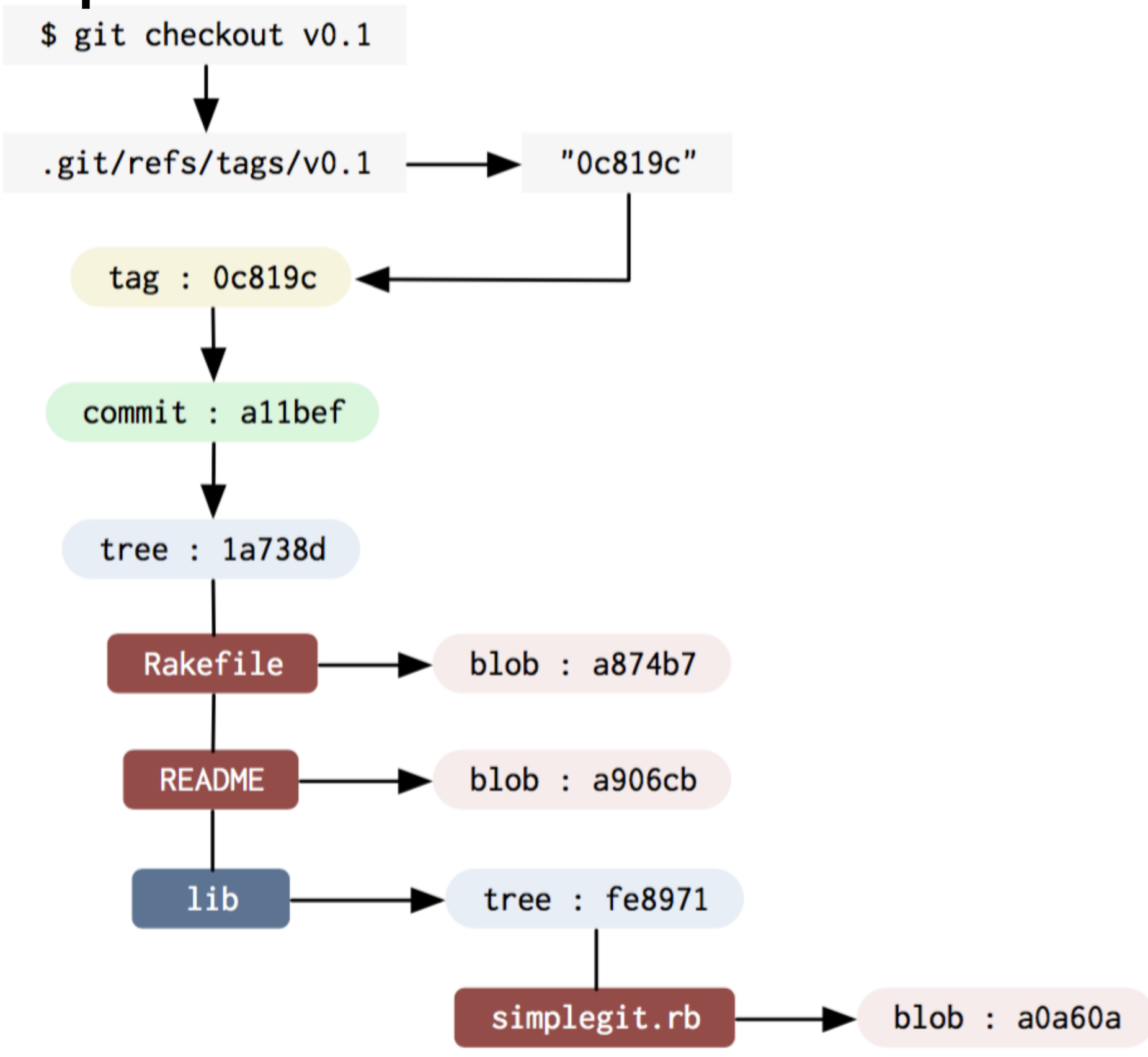
Internals - Example

- history view



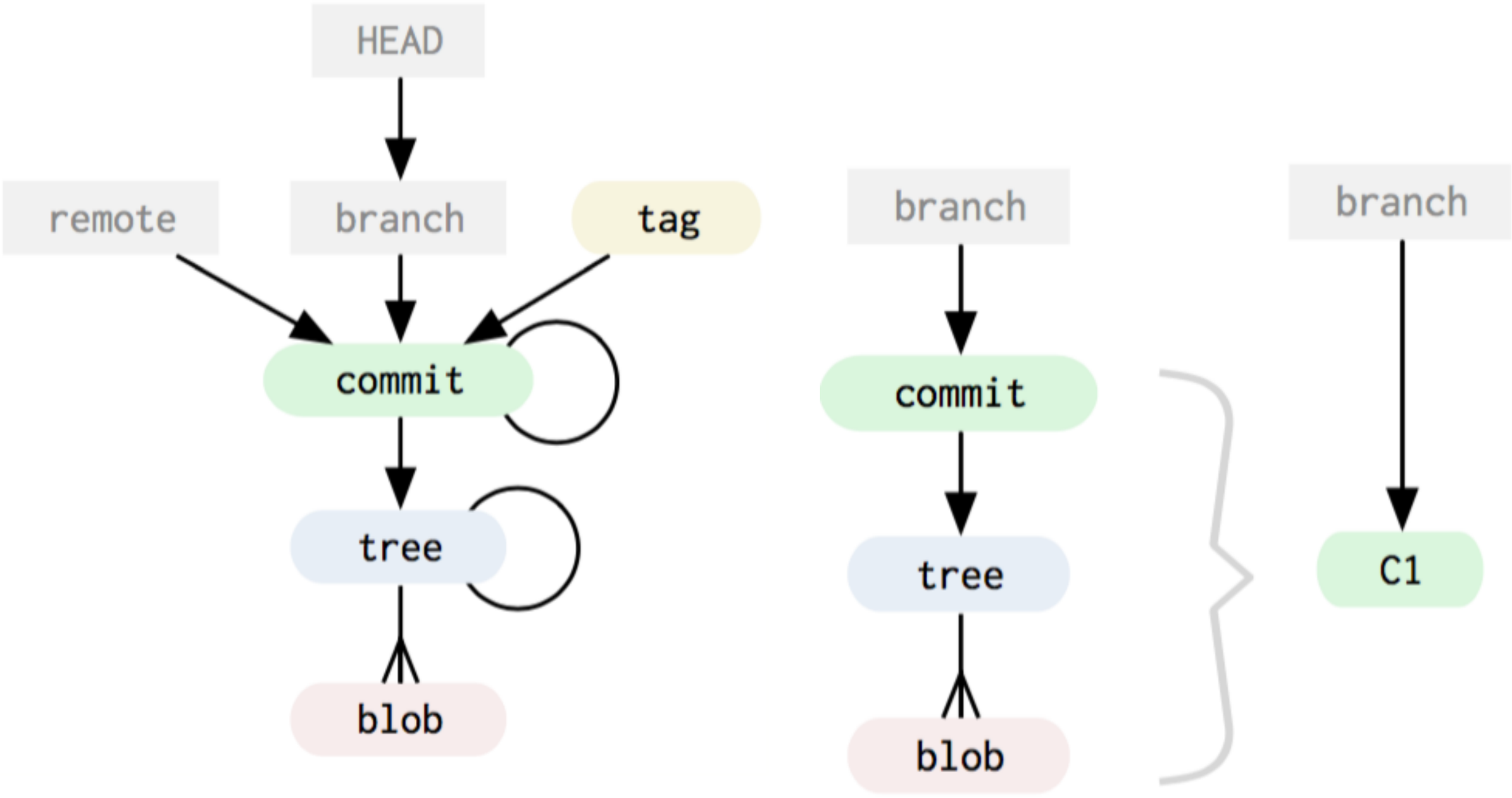
Internals - Example

- git checkout v0.1



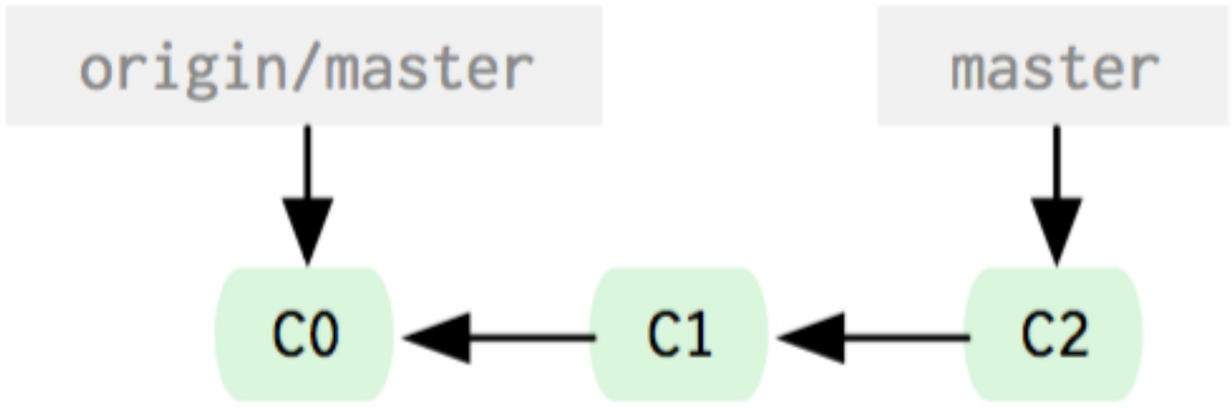
Internals - Branching and Merging

- Branches resident in .git/ refs/heads
- Creating a branch is nothing more than just writing 40 characters to a file



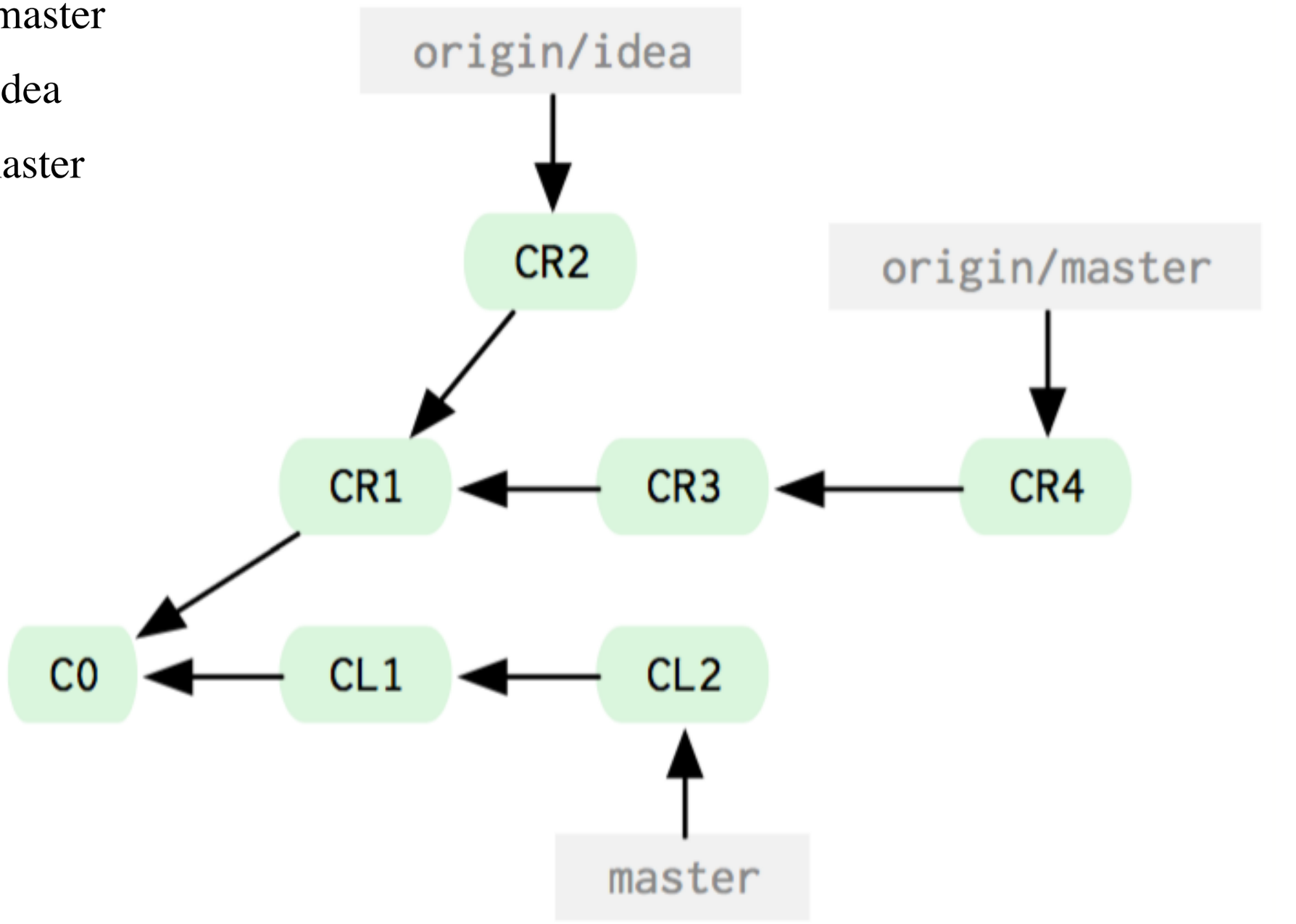
Internals - Remotes

- origin/master
- local master



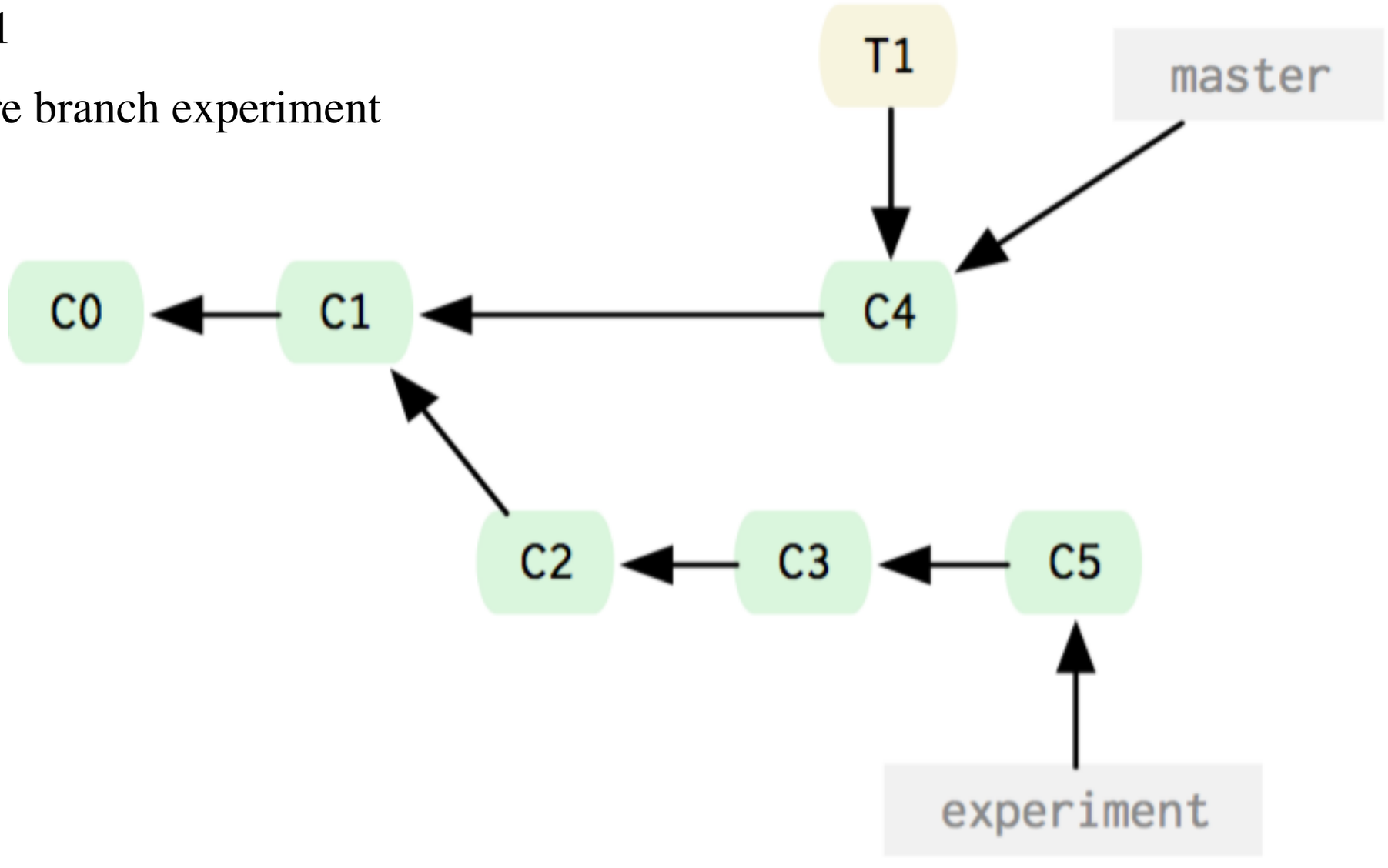
Internals - Remotes

- origin/master
- origin/idea
- local master



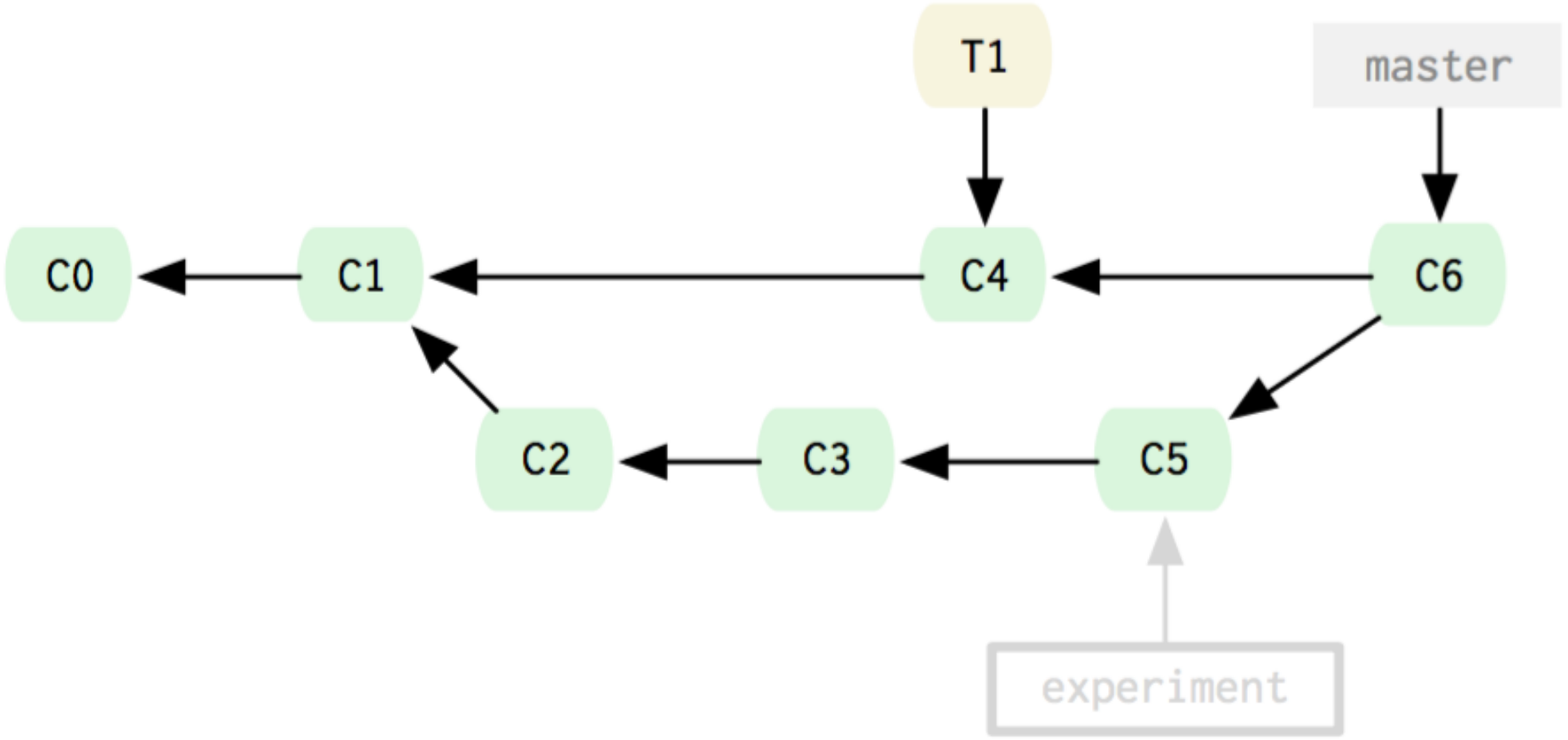
Internals - Merge

- local master
- tag T1
- feature branch experiment



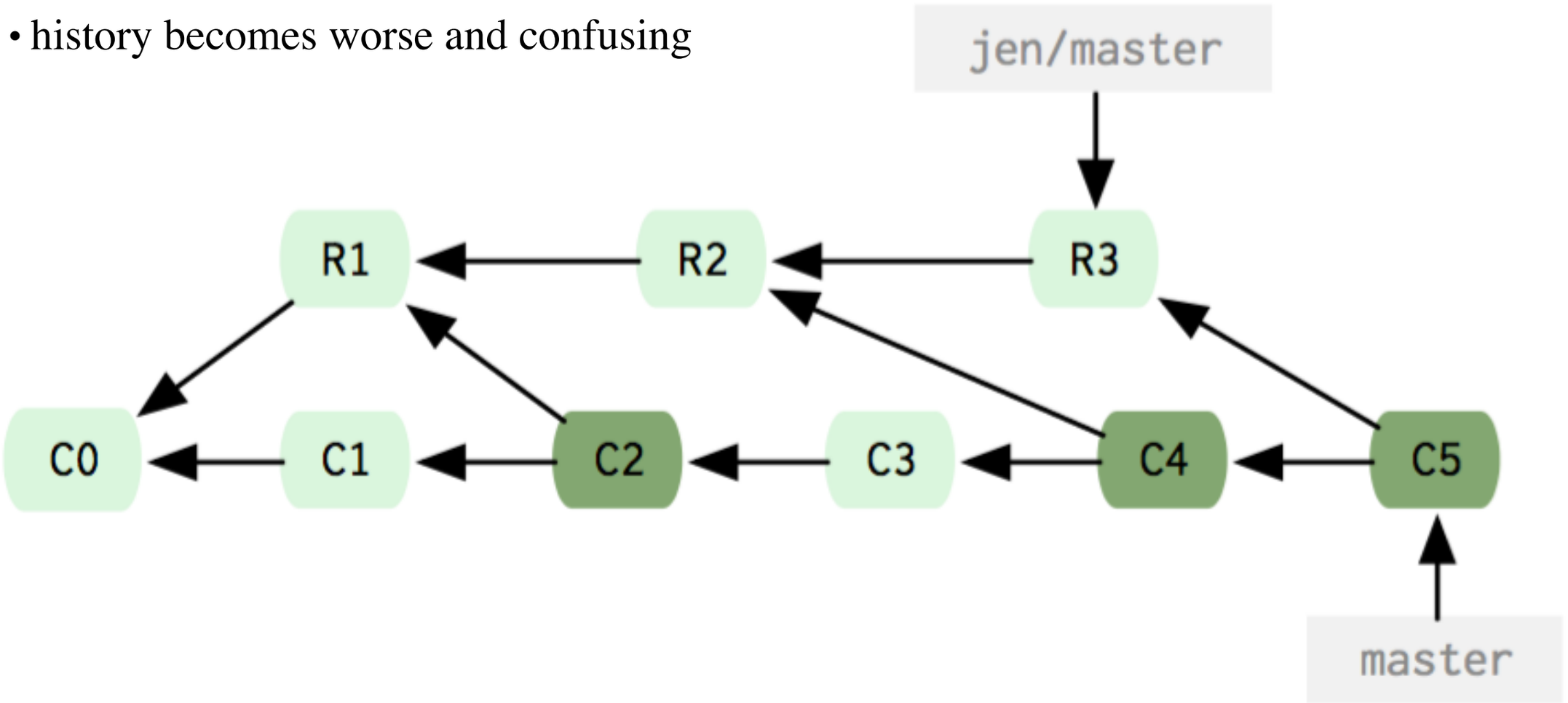
Internals - Merge

- merge experiment into master
- delete experiment branch



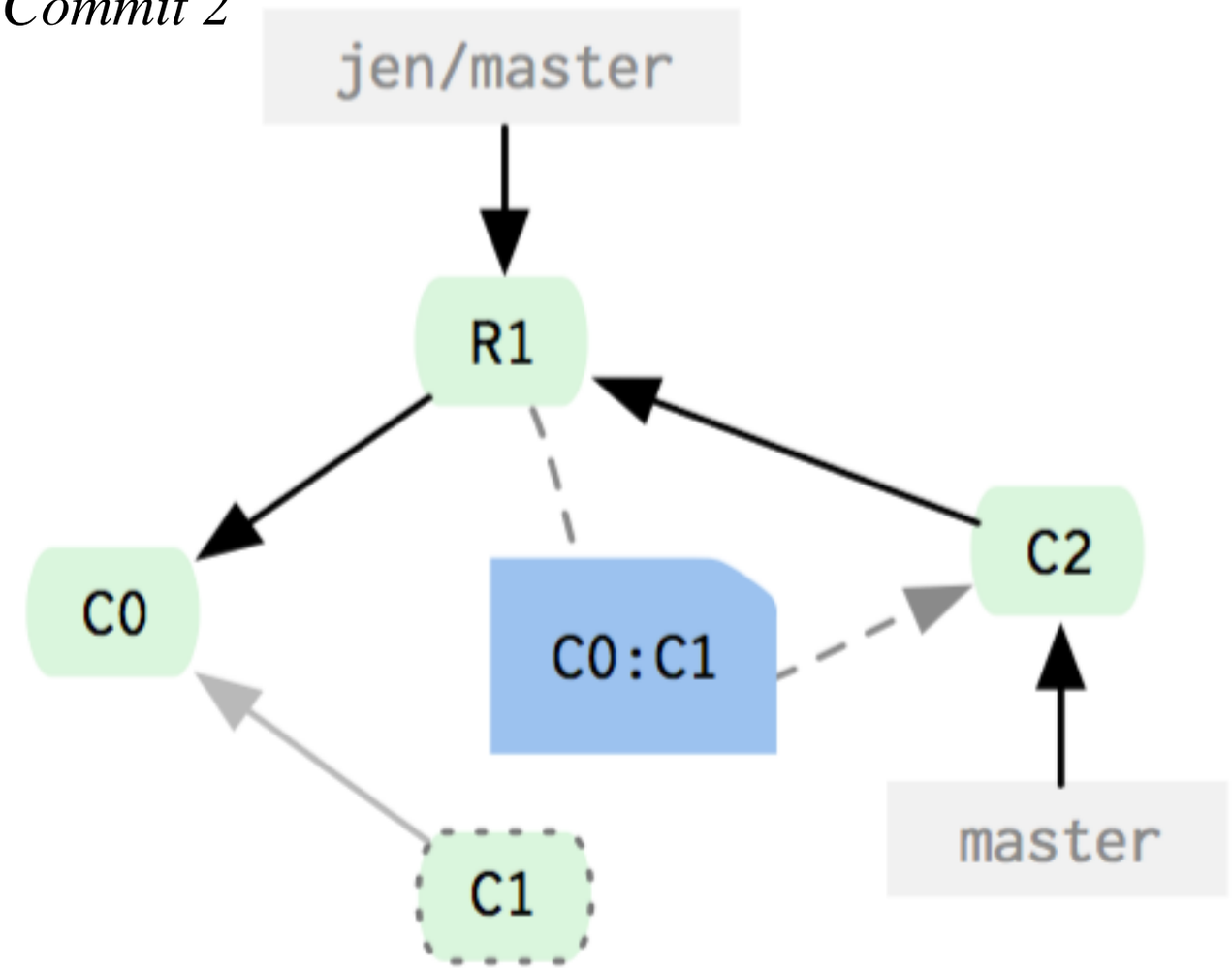
Internals - Rebase

- commits, pull and merge
- history becomes worse and confusing



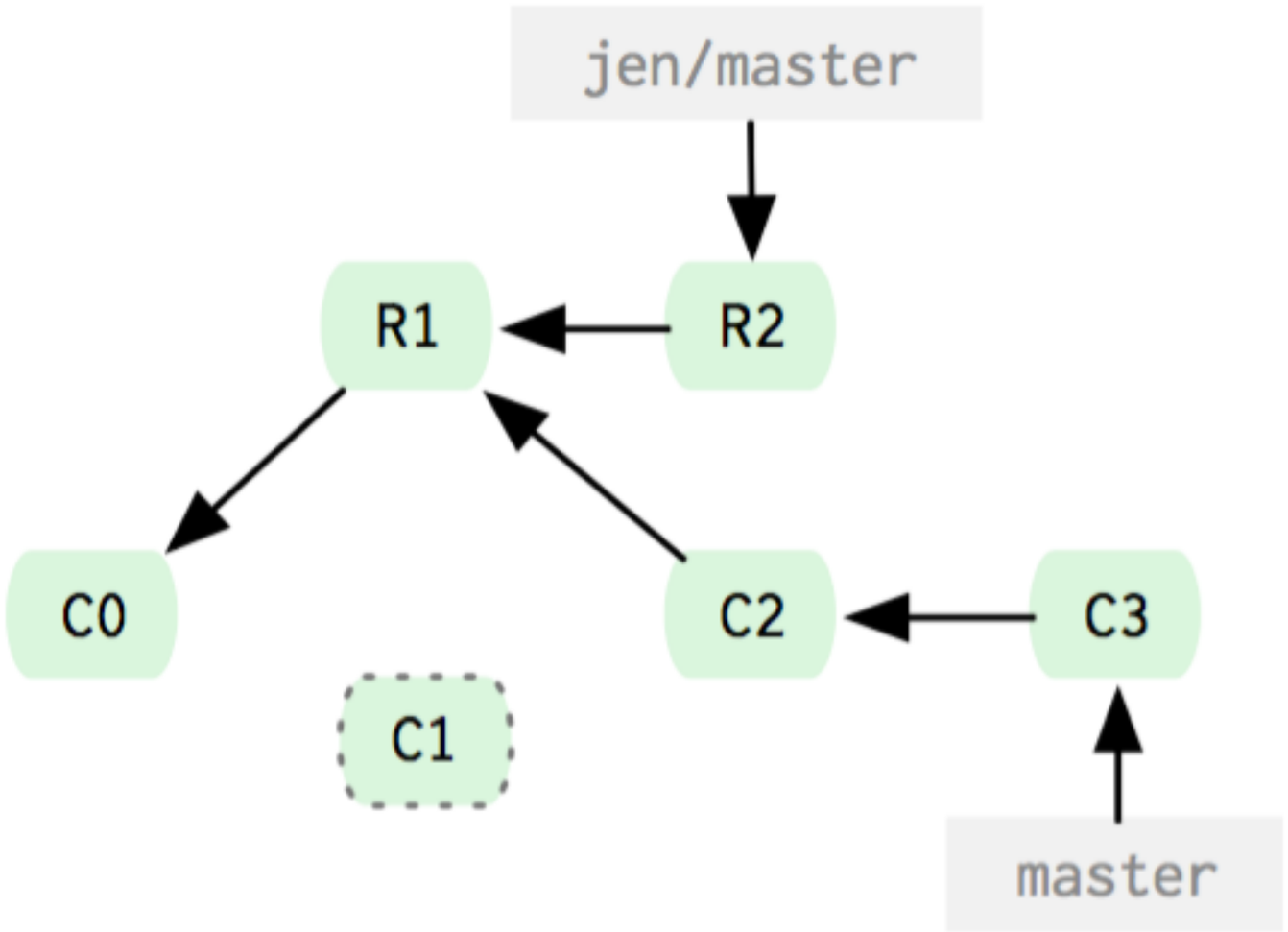
Internals - Rebase

- orphans *Commit 1*
- applies the changes between *Commit 0* and *Commit 1* to the files in *Remote Commit 1*
- creating a new *Commit 2*



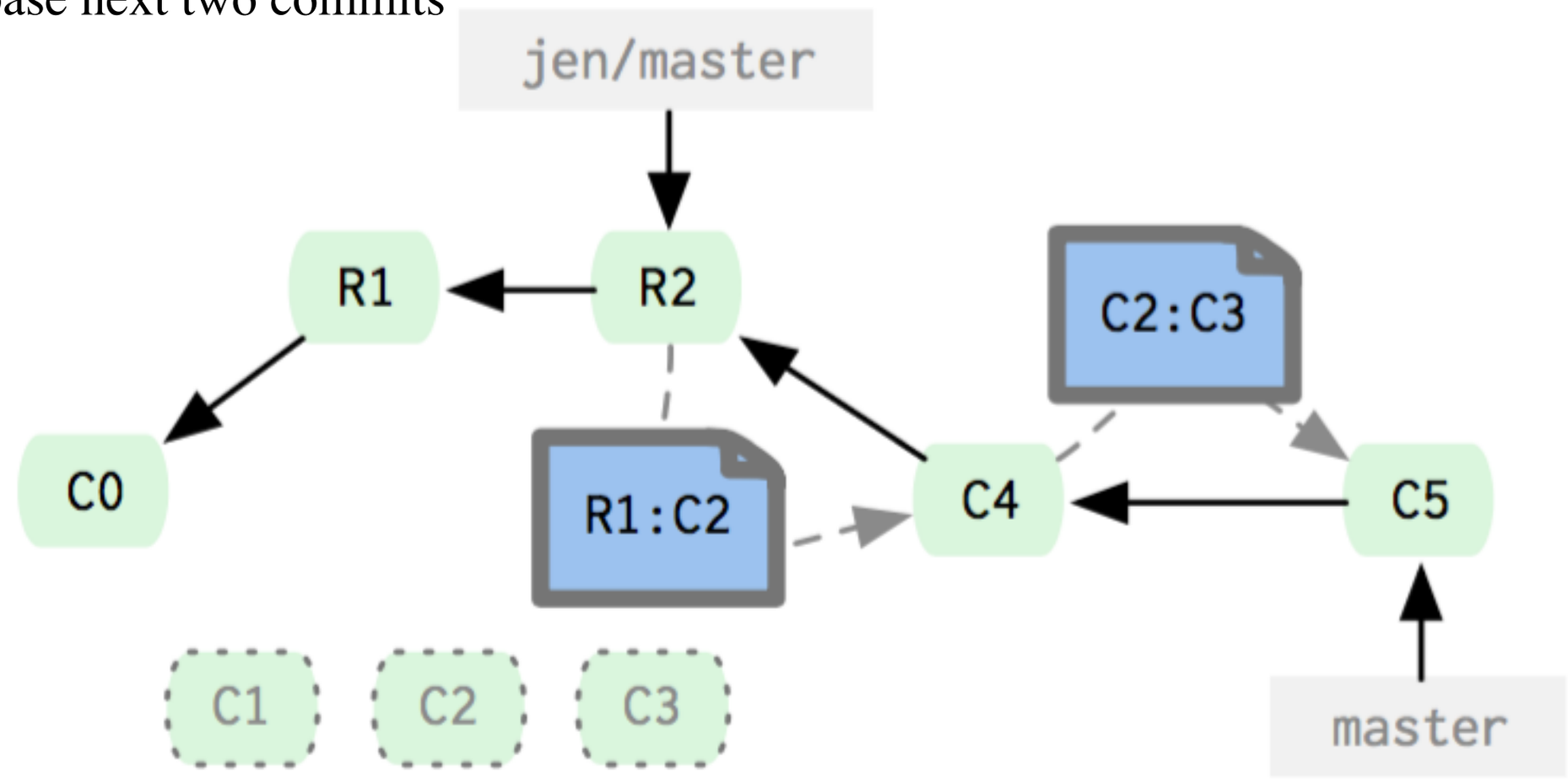
Internals - Rebase

- as you'll remember, you and Jen both commit again



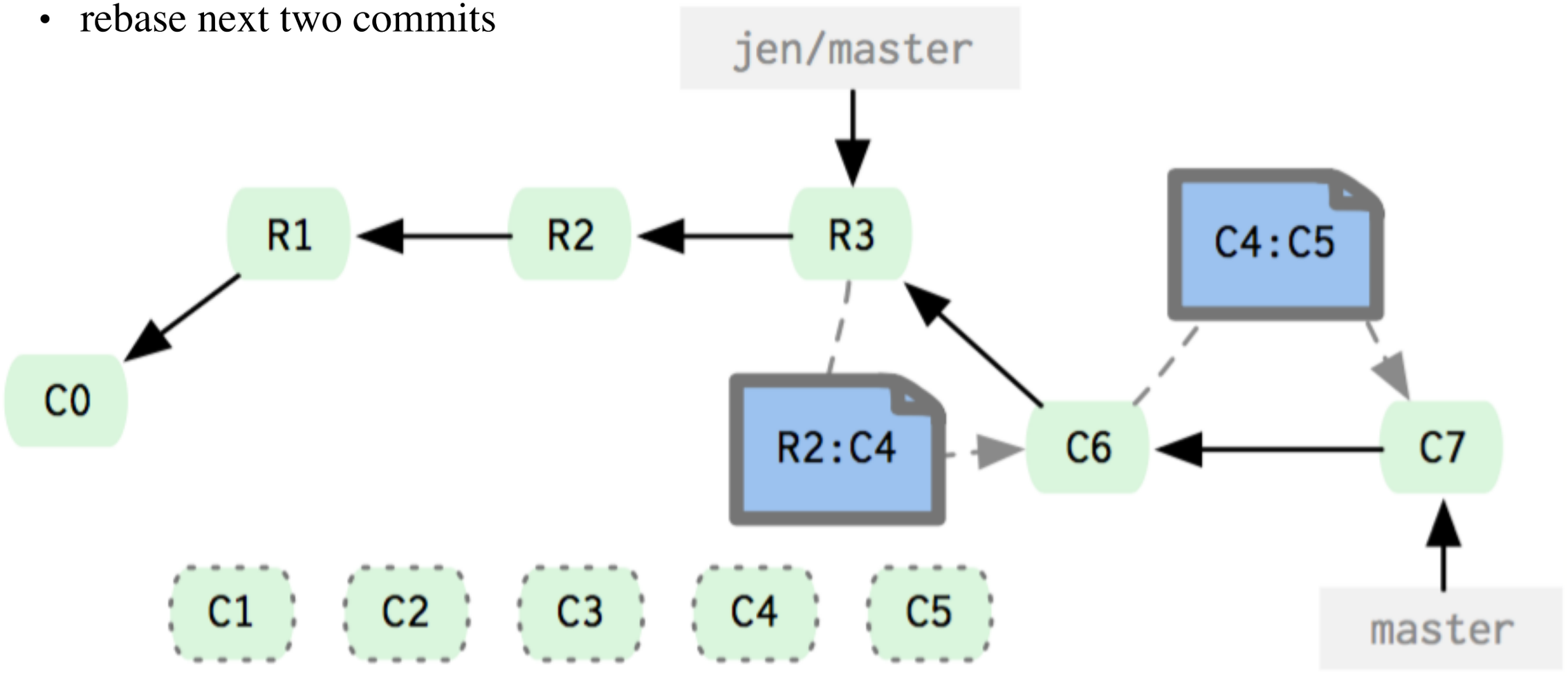
Internals - Rebase

- rebase next two commits



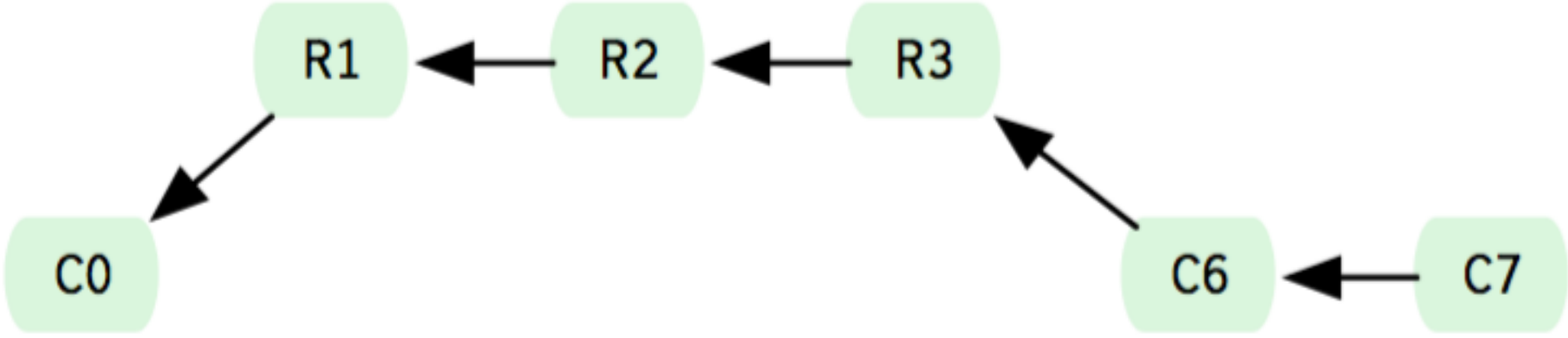
Internals - Rebase

- rebase next two commits

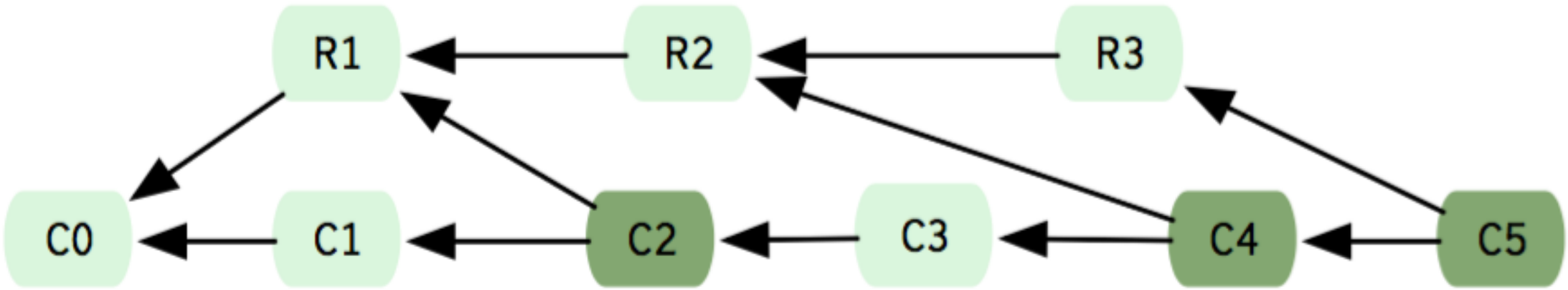


Internals - Rebase

- rebase with linear history



- merge sucks a lot



Play with Git

Play with Git - config

- git config —global user.name “Dylan”
- git config —global user.email “dylanninin@gmail.com

Or

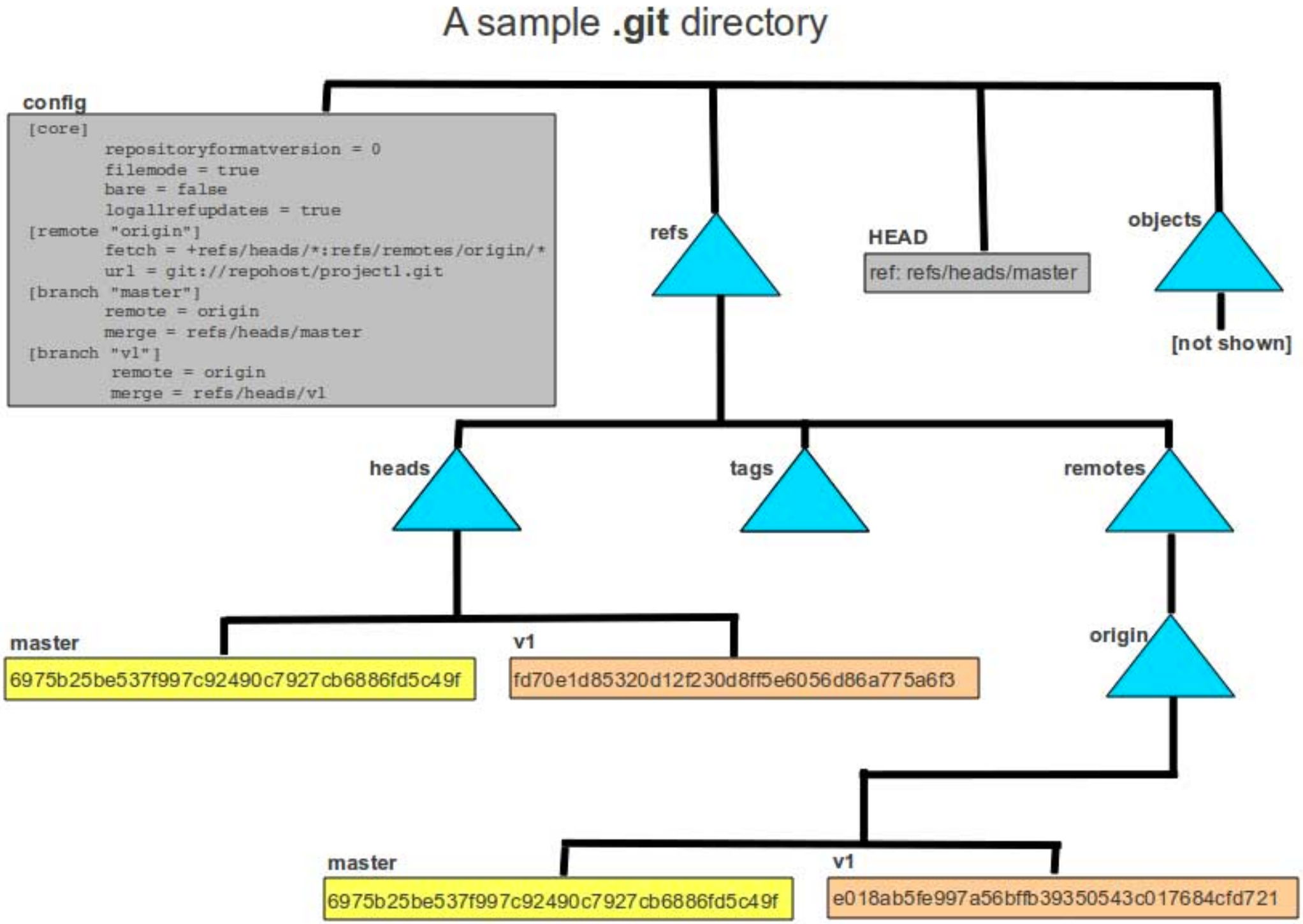
- ~/.gitconfig

```
[user]
  name = Dylan
  email = dylanninin@gmail.com
[core]
  excludesfile = ~/.gitignore
  editor = vim
[alias]
  co = checkout
[log]
  date = relative
```

More such example

- https://gist.github.com/pksunkara/988716

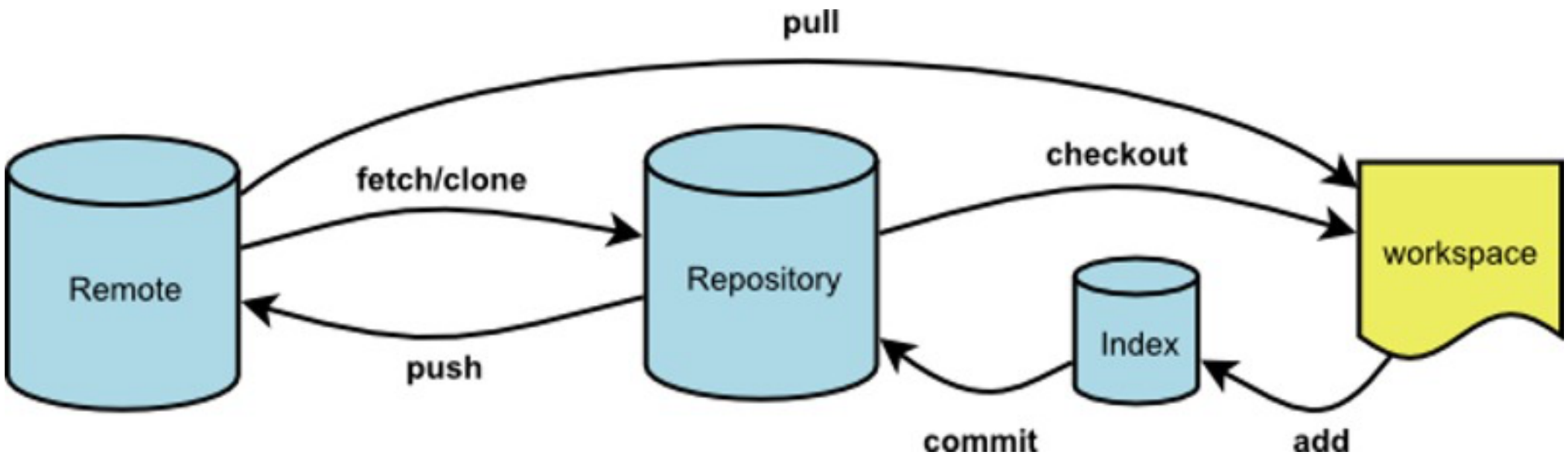
Play with Git - .git directory



http://www.gitguys.com/topics/the-git-directory/

Play with Git - model

- repository/remote/workspace
- index area



Play with Git - work hard

http://www.gitguys.com/topics/the-git-directory/

Play with Git - help

- git help <command>|<concept>
- git help -a
- git help -g

The common Git guides are:

attributes   Defining attributes per path
everyday   Everyday Git With 20 Commands Or So
glossary   A Git glossary
ignore   Specifies intentionally untracked files to ignore
modules   Defining submodule properties
revisions   Specifying revisions and ranges for Git
tutorial   A tutorial introduction to Git (for version 1.5.1 or newer)
workflows   An overview of recommended workflows with Git

'git help -a' and 'git help -g' list available subcommands and some concept guides. See 'git help <command>' or 'git help <concept>' to read about a specific subcommand or concept.

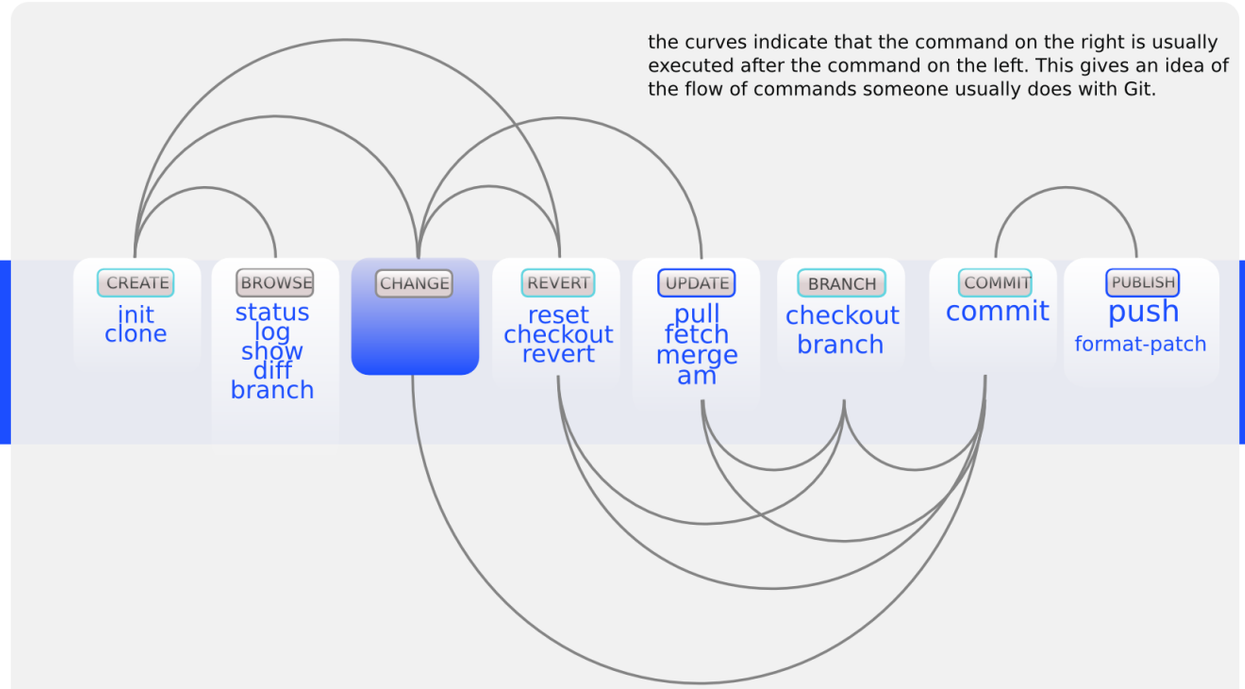
Git Cheat Sheet

http://git.or.cz/

Remember: git command --help

Global Git configuration is stored in $HOME/.gitconfig (git config --help)

Commands Sequence



Create

From existing data

```
cd ~/projects/myproject
git init
git add .
```

From existing repo

```
git clone ~/existing/repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://you@host.org/proj.git
```

Show

Files changed in working directory

```
git status
```

Changes to tracked files

```
git diff
```

What changed between $ID1 and $ID2

```
git diff $id1  $id2
```

History of changes

```
git log
```

History of changes for file with diffs

```
git log -p $file $dir/ec/tory/
```

Who changed what and when in a file

```
git blame $file
```

A commit identified by $ID

```
git show $id
```

A specific file from a specific $ID

```
git show $id:$file
```

All local branches

```
git branch
```

(star '*' marks the current branch)

Cheat Sheet Notation

$id : notation used in this sheet to represent either a commit id, branch or a tag name
$file : arbitrary file name
$branch : arbitrary branch name

Concepts

Git Basics

```
master    : default development branch
origin    : default upstream repository
HEAD      : current branch
HEAD^     : parent of HEAD
HEAD~4    : the great-great grandparent of HEAD
```

Revert

Return to the last committed state

```
git reset --hard
```

⚠ you cannot undo a hard reset

Revert the last commit

```
git revert HEAD
```

Creates a new commit

Revert specific commit

```
git revert $id
```

Creates a new commit

Fix the last commit

```
git commit -a --amend
```

(after editing the broken files)

Checkout the $id version of a file

```
git checkout $id $file
```

Branch

Switch to the $id branch

```
git checkout $id
```

Merge branch1 into branch2

```
git checkout $branch2
git merge branch1
```

Create branch named $branch based on the HEAD

```
git branch $branch
```

Create branch $new_branch based on branch $other and switch to it

```
git checkout -b $new_branch $other
```

Delete branch $branch

```
git branch -d $branch
```

Update

Fetch latest changes from origin

```
git fetch
```

(but this does not merge them).

Pull latest changes from origin

```
git pull
```

(does a fetch followed by a merge)

Apply a patch that some sent you

```
git am -3 patch.mbox
```

(in case of a conflict, resolve and use `git am --resolved`)

Publish

Commit all your local changes

```
git commit -a
```

Prepare a patch for other developers

```
git format-patch origin
```

Push changes to origin

```
git push
```

Mark a version / milestone

```
git tag v1.0
```

Useful Commands

Finding regressions

```
git bisect start      (to start)
git bisect good $id   ($id is the last working version)
git bisect bad $id    ($id is a broken version)

git bisect bad/good   (to mark it as bad or good)
git bisect visualize  (to launch gitk and mark it)
git bisect reset      (once you're done)
```

Check for errors and cleanup repository

```
git fsck
git gc --prune
```

Search working directory for foo()

```
git grep "foo()"
```

Resolve Merge Conflicts

To view the merge conclicts

```
git diff      (complete conflict diff)
git diff --base  $file  (against base file)
git diff --ours  $file  (against your changes)
git diff --theirs $file (against other changes)
```

To discard conflicting patch

```
git reset --hard
git rebase --skip
```

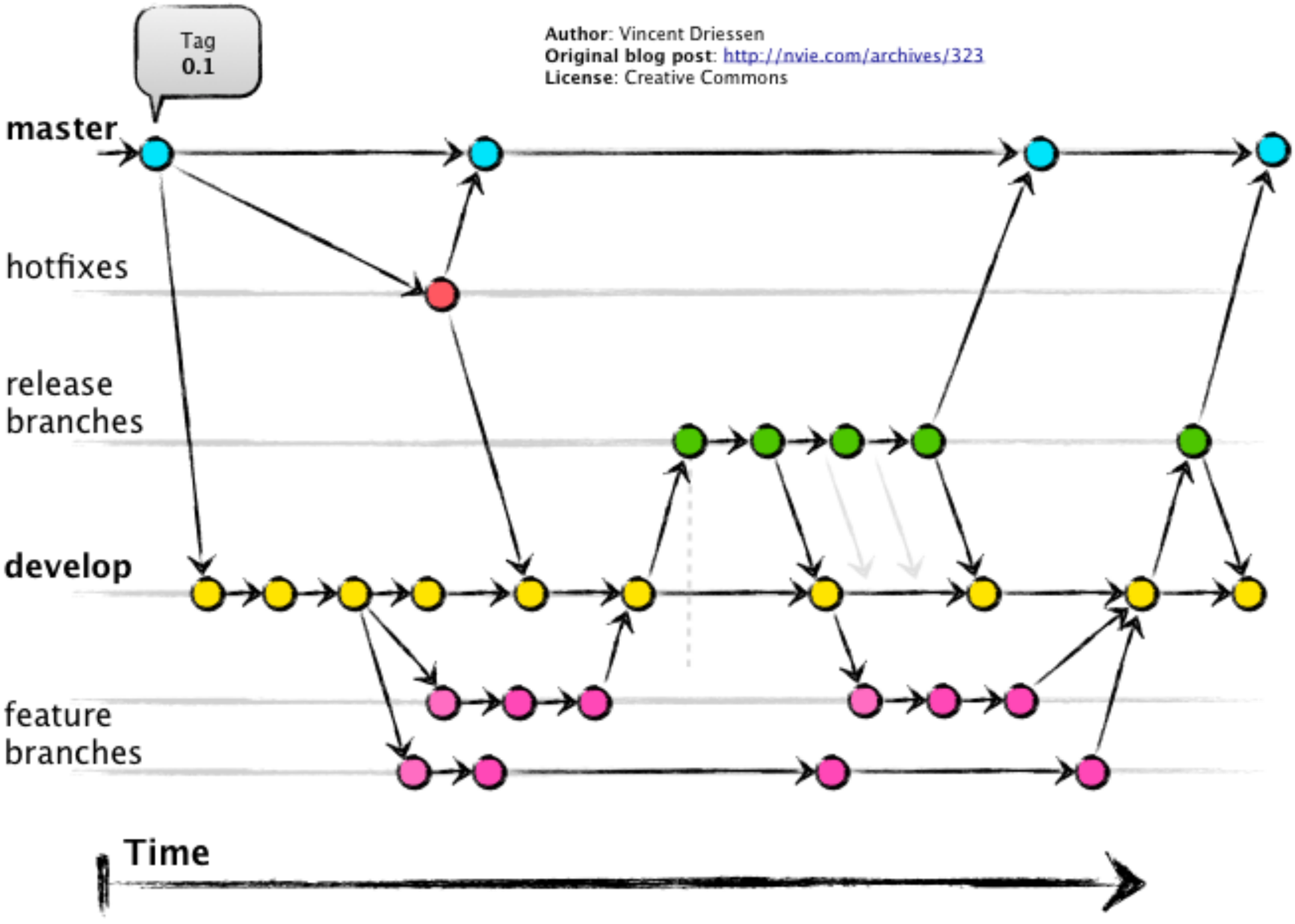
After resolving conflicts, merge with

```
git add $conflicting_file  (do for all resolved files)
git rebase --continue
```

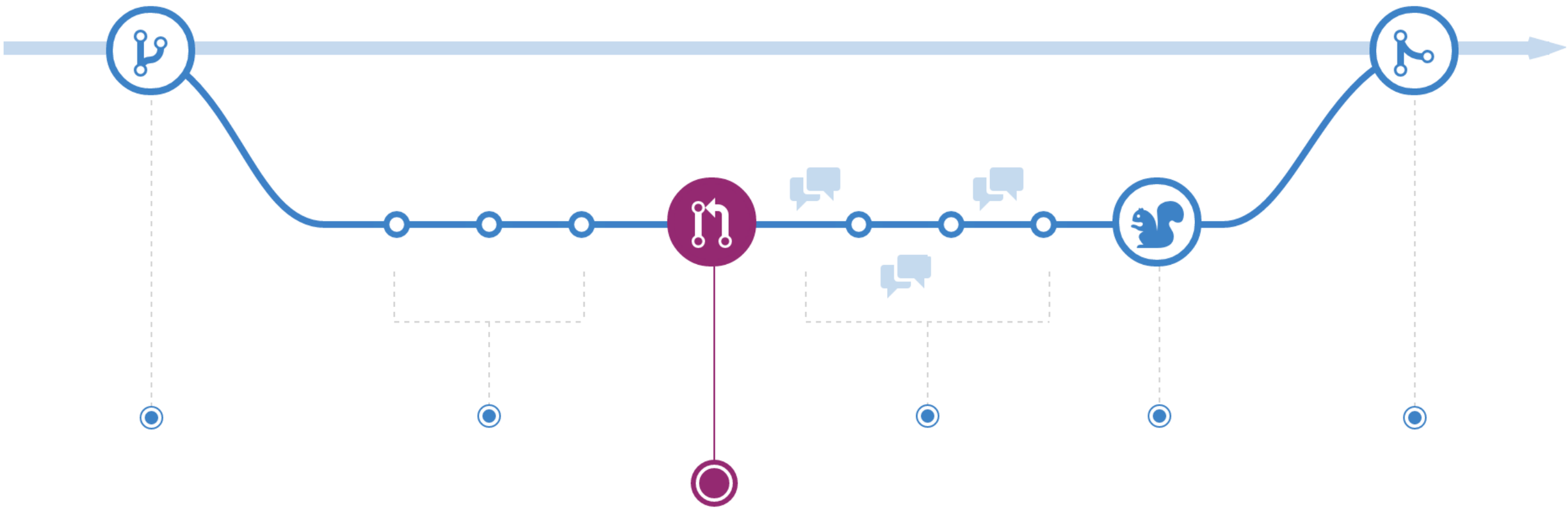
Zack Rusin
Based on the work of:
Sébastien Pierre
Xprima Corp.

Workflow

Git-flow


Tag
0.1
Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323
License: Creative Commons
master
hotfixes
release
branches
develop
feature
branches
Time

Github Flow



create a branch

add commits

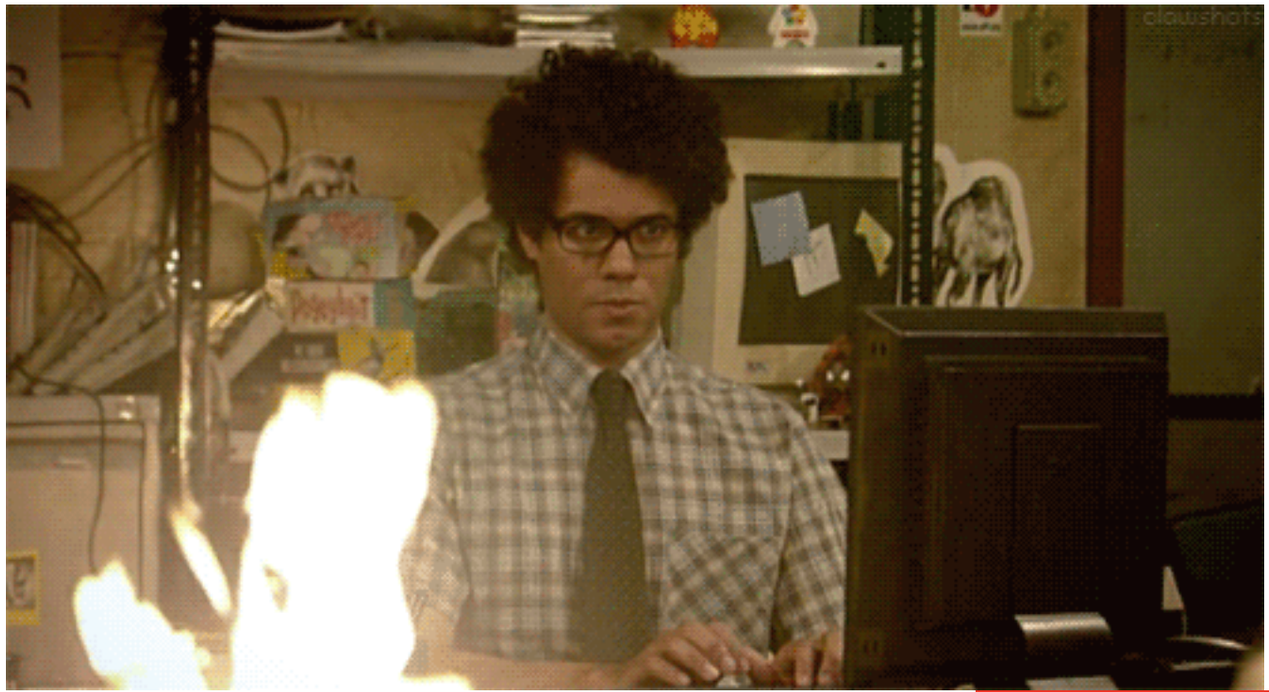
open a pull request

discuss & review

deploy

merge into master

In case of fire



1. git commit
2. git push
3. leave building

Q & A

Reference

- **Git**: https://en.wikipedia.org/wiki/Git_(software)
- **Linus on Git**: https://www.youtube.com/watch?v=4XpnKHJAok8
- **Pro Git**: http://git-scm.com/book
- **Git Internals**: http://git-scm.com/book/en/v1/Git-Internals
- **Peepcode Git Internals**: https://github.com/pluralsight/git-internals-pdf
- **SHA-1**: https://en.wikipedia.org/wiki/SHA-1
- **Facebook's git repo is 54GB**: https://news.ycombinator.com/item?id=7648237
- **Git Branching model**: http://nvie.com/posts/a-successful-git-branching-model/
- **Github Flow**:
  - http://scottchacon.com/2011/08/31/github-flow.html
  - https://guides.github.com/introduction/flow/