

git practical cases

Tomasz Gebarowski

@tgebarowski

github.com/tgebarowski/

Clean Git History

Regression tracking

Several working trees

Branch management
pro tips

Clean Git History

Regression tracking

Several working trees

Branch management
pro tips

Typical work flow

1

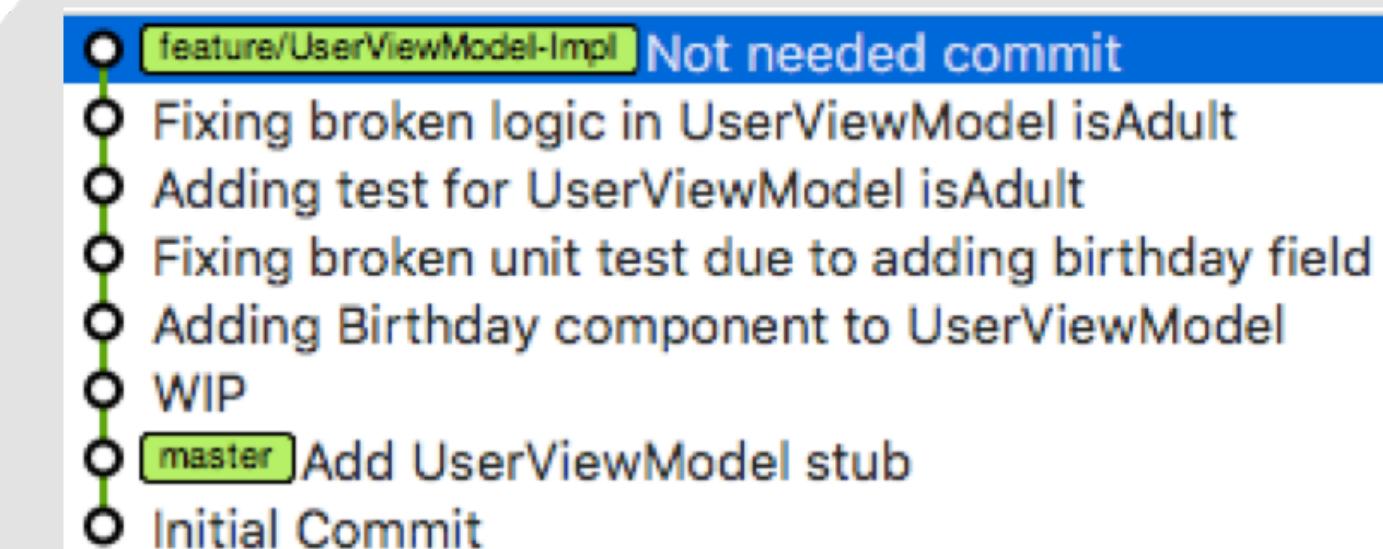
Create a branch
from develop

2

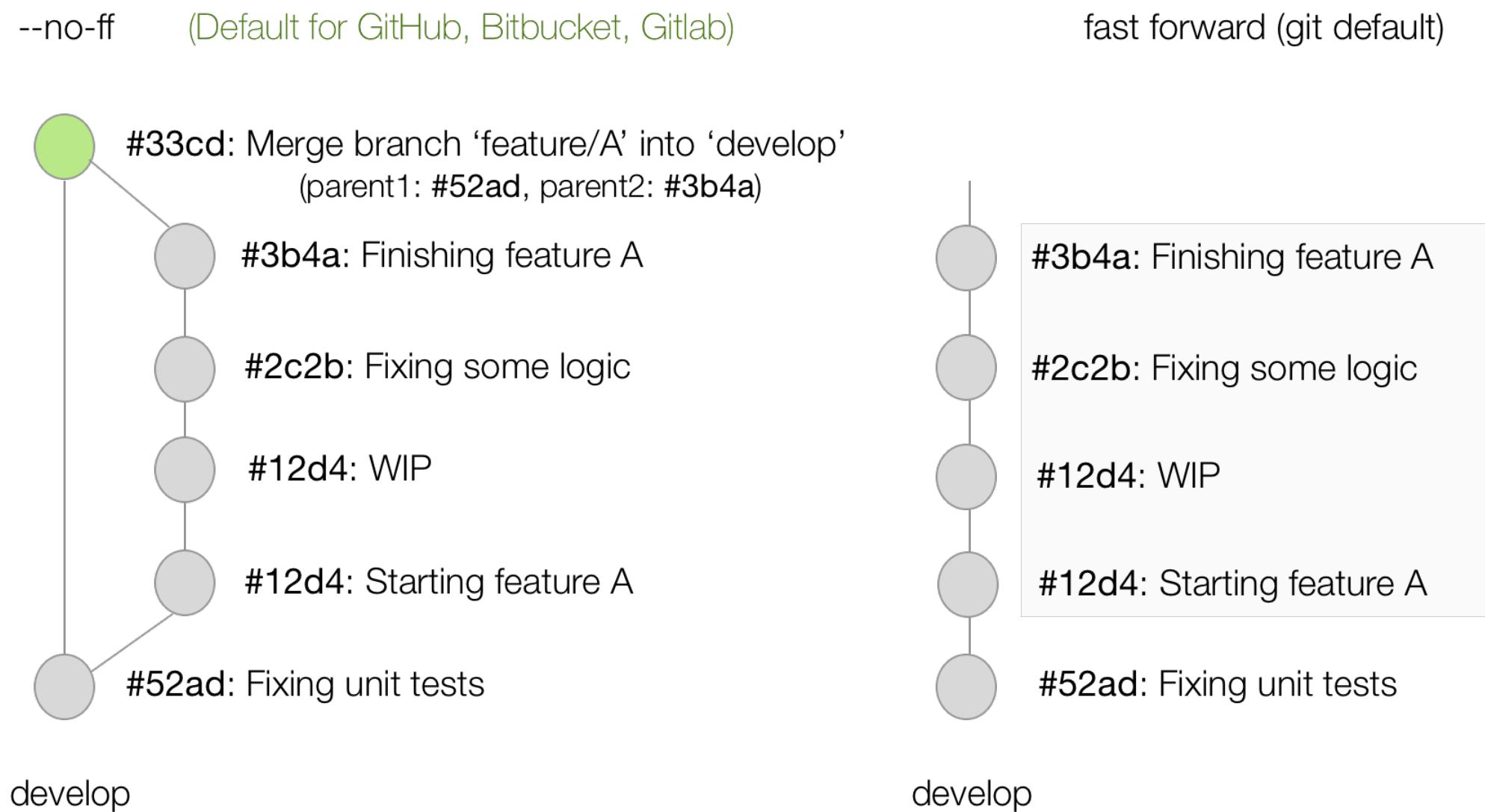
Work

3

Merge back
to develop

- 
- feature/UserViewModel-Impl Not needed commit
 - Fixing broken logic in UserViewModel isAdult
 - Adding test for UserViewModel isAdult
 - Fixing broken unit test due to adding birthday field
 - Adding Birthday component to UserViewModel
 - WIP
 - master Add UserViewModel stub
 - Initial Commit

git merge



¿ Fast forward or not ?

- Merge with fast forward is hard to revert
 - No single commit to revert
 - History is linear

but wait...

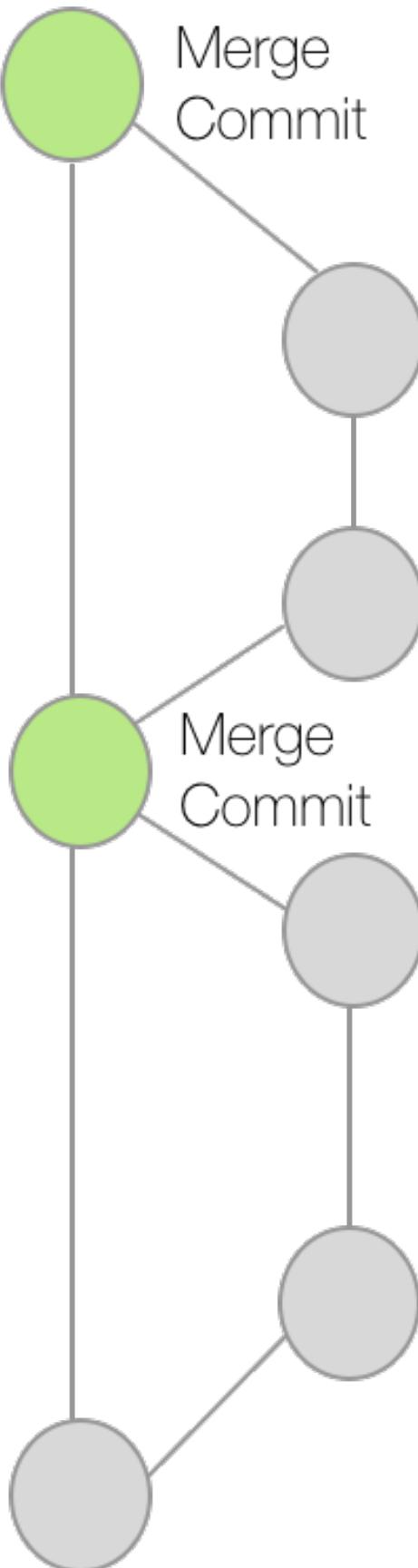


linear history is easier to search
(bisect, blame, log)

Option

3

1. rebase
2. create PR after rebase
3. merge with --no-ff



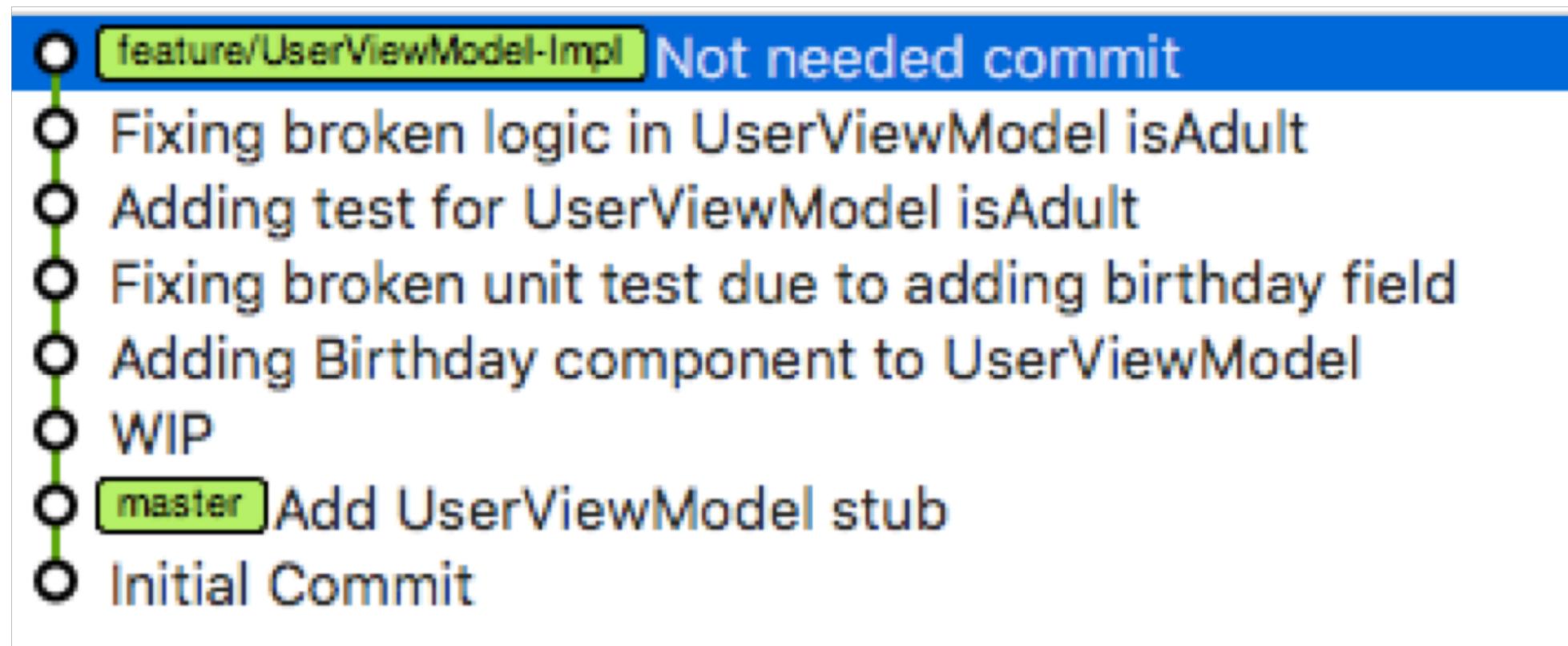
Option

3



- Conflicts are harder to resolve when rebasing
- Commit hash is re-generated
- Extra step

Let's go back to our example

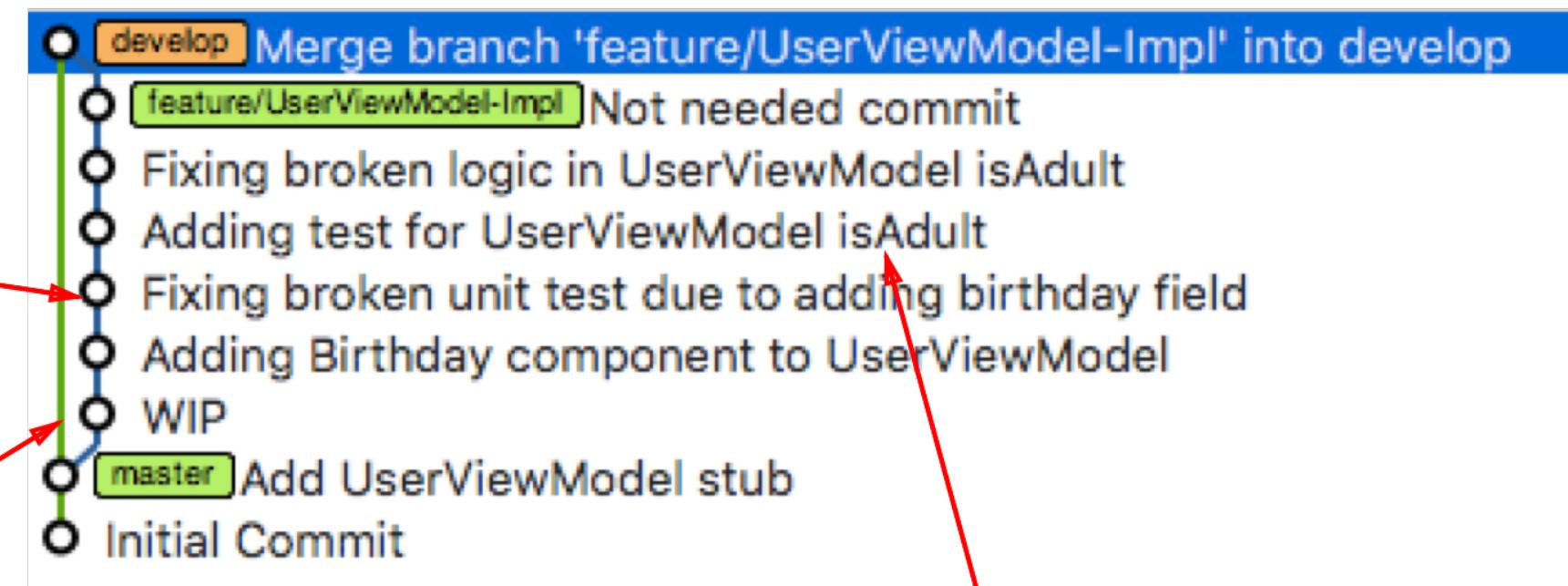


What are the pitfalls ?

- Feature branches often have snapshot (WIP) commits

fixes something broken
in previous commit

some work in progress commit



is related to change
in one of previous commits

Why snapshot commits are bad?

- Usually break the build
- Frequently do not pass unit tests
- Create line noise: One commit fixes another in the same feature branch

Why do we make them?

- Adhoc commits are faster and easier to make
- We need checkpoints in our code
- Before intermediate code reviews

They get merged into develop



Breaking git bisect, leaving develop in messy
state

What are the options

Regular History

VS

Clean History

Let's be clean...

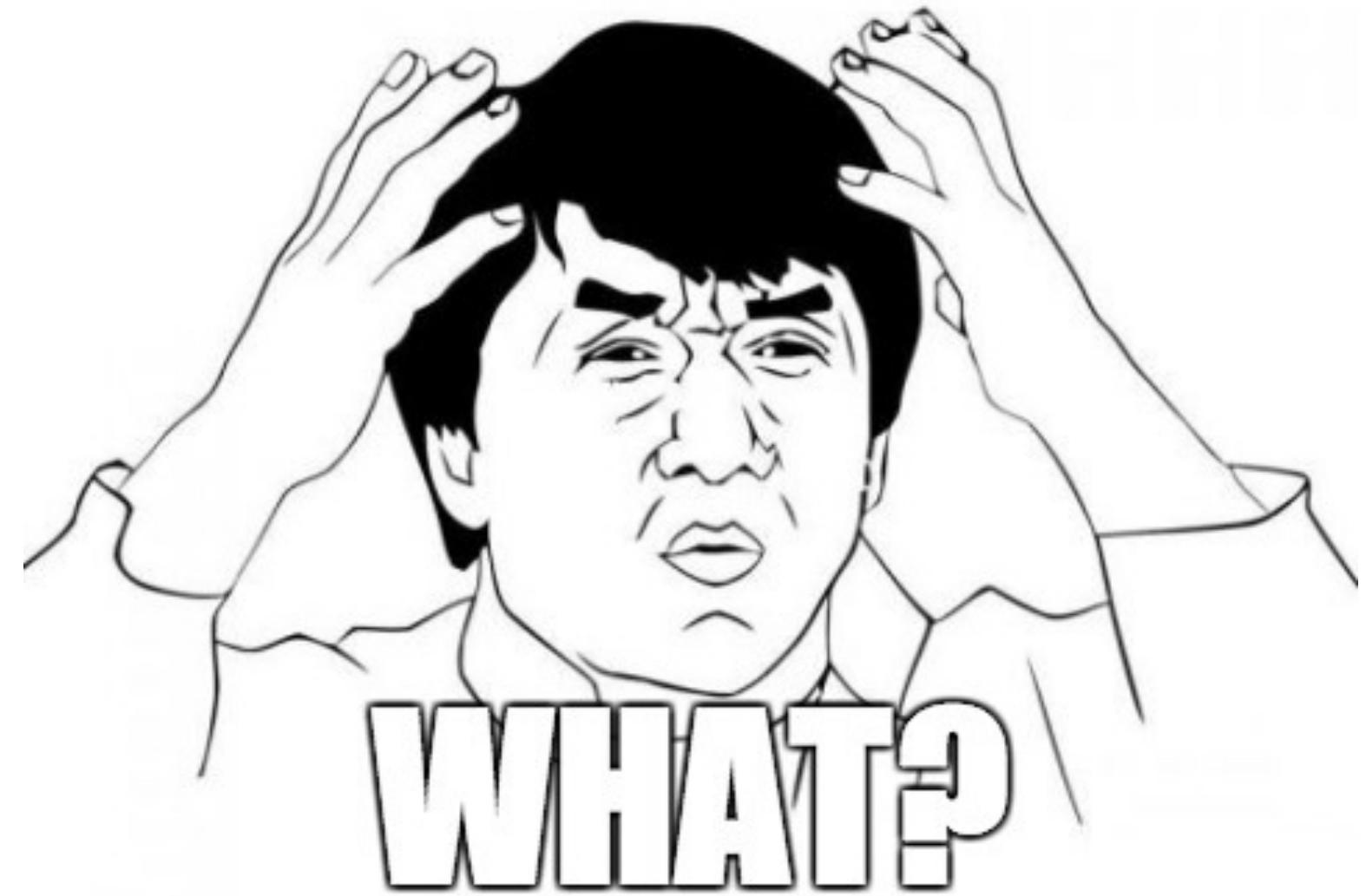
Clean history (branch types)

Public

- Immutable
- Every commit well documented
- Includes: master, develop, release*, hotfix*
- No cherrypicking between public branches

Private

- Scratchpads
- Local, no forks possible
- History rewrite is plausible
- Can cherrypick from private branches



history rewrite?



**KEEP
CALM
AND
REBASE -I**

***if you work on a private branch**

History rewrite tips

Rewriting history

It's all about git rebase:

```
git rebase -i HEAD~N  
git rebase -i SHA
```

where:

- **N**, number of commits we want to go back to rewrite history
- **SHA** of previous commit before the commit we want to include in our history rewrite

git rebase -i HEAD~6

HEAD

```
pick e9c2acd SettingsCell: Fixing ambiguous constraints
pick ae1ddaf SettingsCell: Removed duplicated constraint
pick 16c3e9d Allow to put an icon on a Screen
pick f9cd935 Release 0.0.2
pick 39069aa podspec: Updated version to 0.0.2
pick b349543 Tests: Removed OptionsTest conflict file

# Rebase 5c8c4d1..b349543 onto 5c8c4d1 (6 command(s))
#
# 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
# d, drop = remove commit
```

Squashing with **fixup!** (1)

- add commit that is to be squashed with one of the commits from history
- prefix it with **fixup!** and add the same commit message as the commit that is rewritten

Subject	Author	Date
develop fixup! adding feature B in file1	Tomasz Gebarowski	2015-12-30 09:55:36
Revert "Merge branch 'feature/A' into develop"	Tomasz Gebarowski	2015-12-29 09:24:39
feature/B adding feature B in file1	Tomasz Gebarowski	2015-12-28 17:17:49
master Merge branch 'feature/A' into develop	Tomasz Gebarowski	2015-12-28 17:16:28
feature/A adding feature A in file2	Tomasz Gebarowski	2015-12-28 17:15:42
adding feature A in file1	Tomasz Gebarowski	2015-12-28 17:15:24
Initial commit	Tomasz Gebarowski	2015-12-28 17:13:53

Squashing with fixup! (2)

- rebase with autosquash option

```
git rebase -i --autosquash HEAD~3
```

```
pick 8844119 adding feature B in file1
fixup ed6d885 fixup! adding feature B in file1
pick 5a29779 Revert "Merge branch 'feature/A' into develop"
```

```
# Rebase a9c93b9..ed6d885 onto a9c93b9 (            3 TODO item(s))
```

Amending merge commit

git commit --amend -p (or --preserve-merges)

git rebase -i -p

git will try to preserve the merges when rebasing, rather than linearizing the history

Reverting Merge Commit

Subject	Author	Date
develop Merge branch 'feature/B' into de...	Tomasz Gebarowski	2015-12-28 17:18:38
feature/B adding feature B in file1	Tomasz Gebarowski	2015-12-28 17:17:49
master Merge branch 'feature/A' into develop	Tomasz Gebarowski	2015-12-28 17:16:28
feature/A adding feature A in file2	Tomasz Gebarowski	2015-12-28 17:15:42
adding feature A in file1	Tomasz Gebarowski	2015-12-28 17:15:24
Initial commit	Tomasz Gebarowski	2015-12-28 17:13:53

```
git revert -m 1 a9c93b99d
```

where: *a9c93b99d* is Merge Request hash

Reverting Merge Commit

After:

Subject: Revert "Merge branch 'feature/A' into develop"
Author: Tomasz Gebarowski 2015-12-29 09:24:39
develop Merge branch 'feature/B' into develop
feature/B adding feature B in file1
master Merge branch 'feature/A' into develop
feature/A adding feature A in file2
adding feature A in file1
Initial commit

SHA: 5be26e92f3376a5451c9470c1e8e015568d0bc34
Author: Tomasz Gebarowski <tomasz.gebarowski@partners.mbank.pl>
Date: Tue Dec 29 2015 09:24:39 GMT+0100 (CET)
Subject: Revert "Merge branch 'feature/A' into develop"
Parent: [37239923020f896c702a65a17bdd50620f461c49](#)

Revert "Merge branch 'feature/A' into develop"
This reverts commit a9c93b99d8ba75cc31412c2088c8cb593a684677, reversing
changes made to 0e52bef482cb2273f2e2abb3ef9d55d9d99535f7.

changed file1
deleted file2

file1
... ... @@ -1,2 +1 @@
1 -A
2 1 B

file2
... ... @@ -1 +0,0 @@
1 -A

Gist it ?

Diff after rebase (?)

Before each rewrite it is advised to tag your current HEAD:

```
git tag v1
```

then after:

```
git rebase -i HEAD~4
```

you can compare what changed after rewrite:

```
git diff v1 HEAD
```

Diff after rebase (2)

What if you forgot to tag before rebase -i?

```
git tag v2 tomasz/feature/rccs
```

(before history got rewritten in remote)

Pitfalls

- Changing commits may result in conflicts
- It takes time to rearranged already committed code

Benefits

- Commits treated as code documentation
- Well organized commits are easier to review
- No problems with git bisect

Clean Git History

Regression tracking

Several working trees

Branch management
pro tips

Regression tracking with git bisect

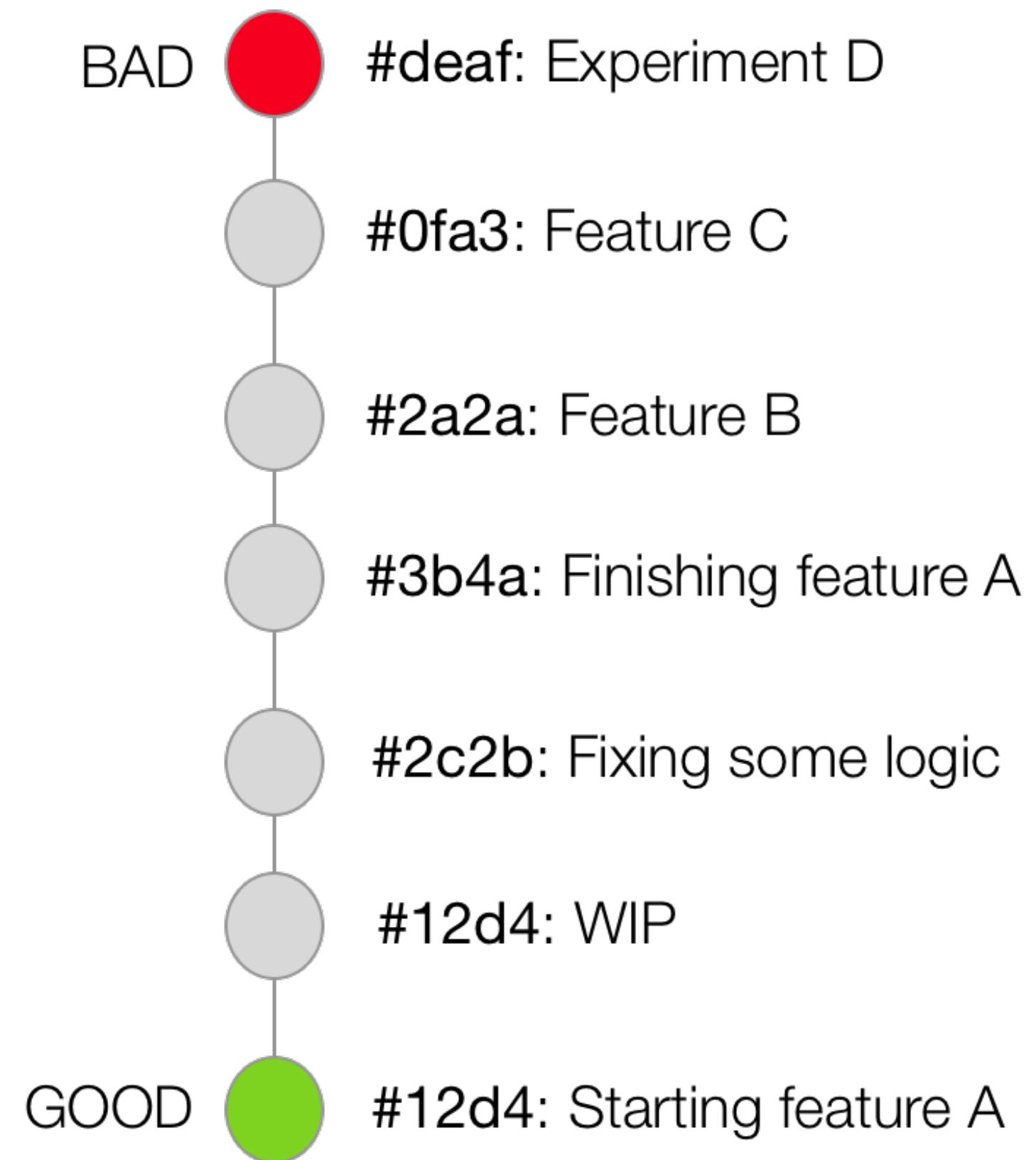
Binary search of commit that caused error

Regression tracking: git bisect

- **Setup**

```
$ git bisect start  
$ git bisect bad  
$ git bisect good 12d4
```

Bisecting: 3 revisions left to test after this (roughly 2 steps)

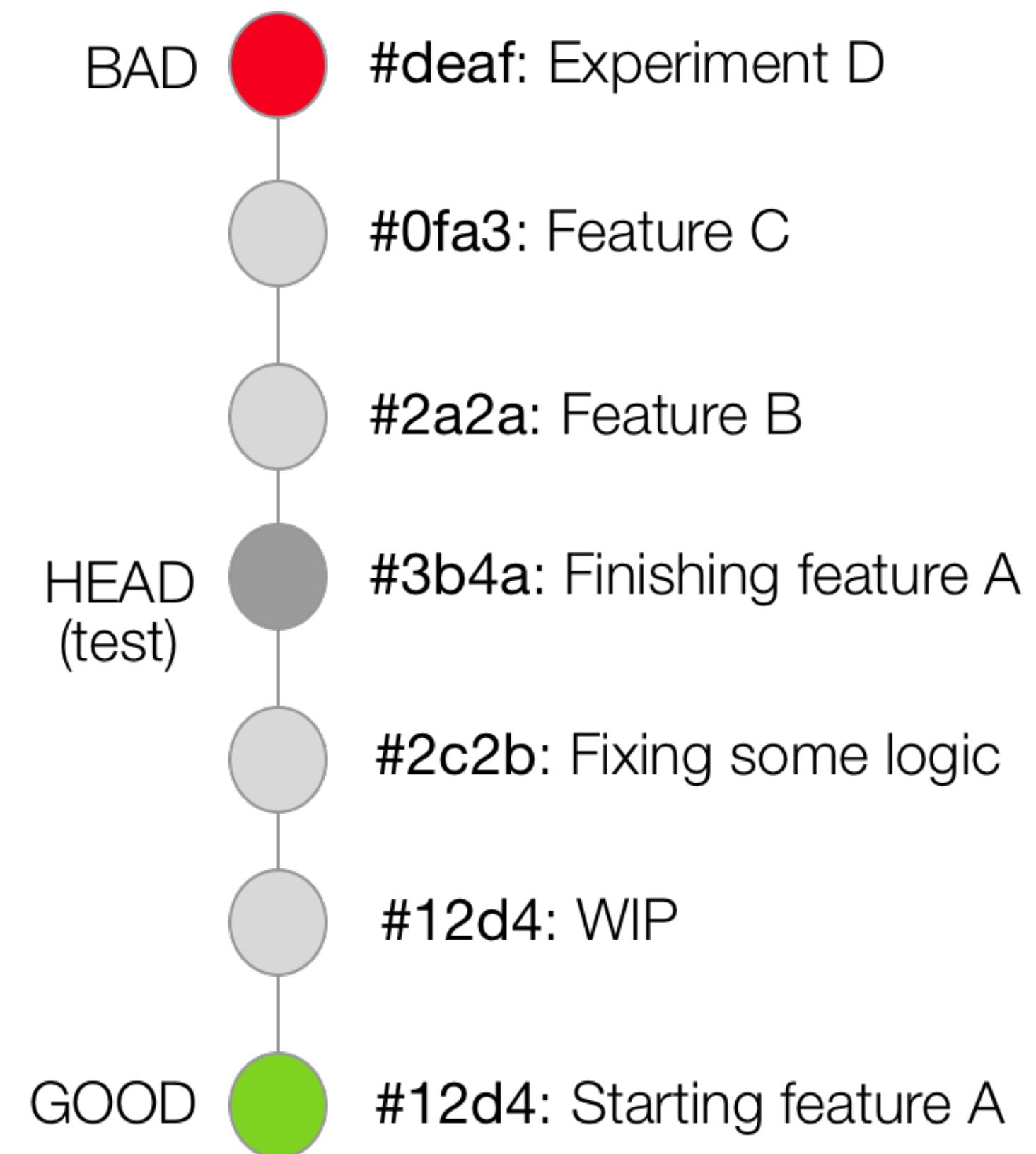


Regression tracking: git bisect

- **Test 1**

```
git bisect bad
```

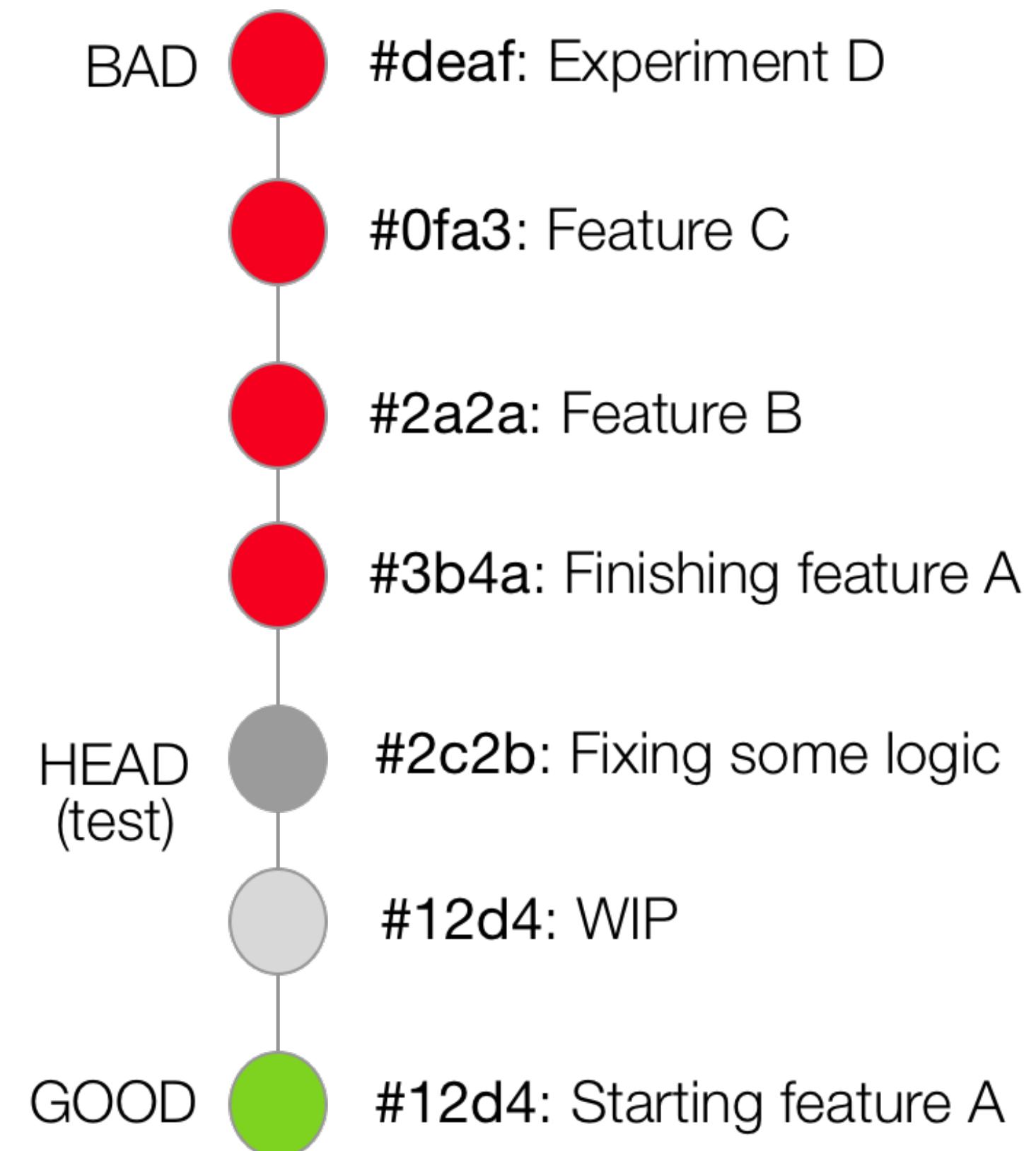
Bisecting: 0 revisions left to test after this (roughly 0 steps)



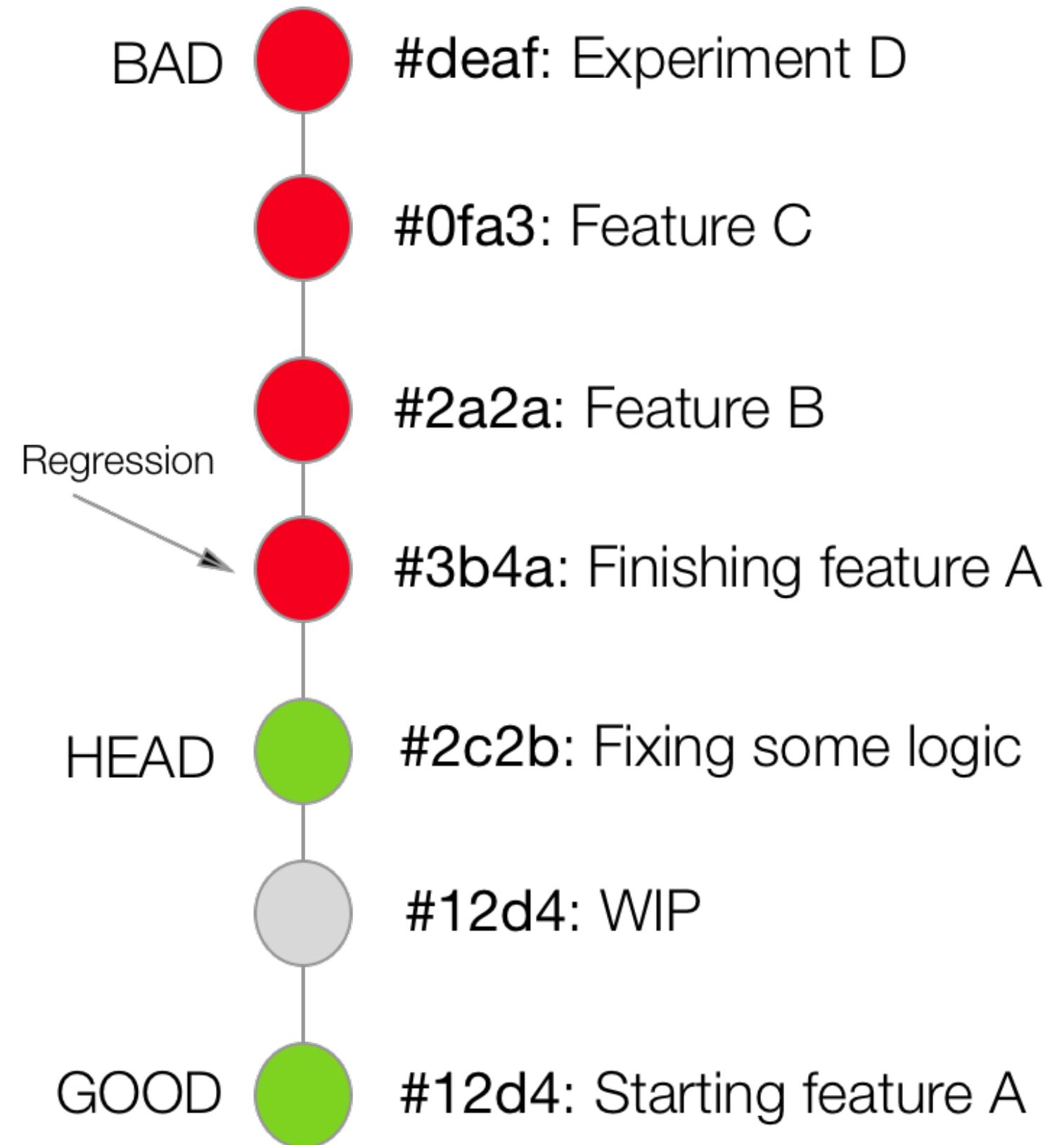
Regression tracking: git bisect

- **Test 2**

```
$ git bisect good
```



Regression tracking: git bisect



- **Results**

3b4a is first bad commit

Remember to call `git bisect reset`, to bring back original
HEAD state

Automatic regression tracking

git bisect start

git bisect bad

git bisect good 12d4

git bisect run ./test.sh

test.sh:

```
xcodebuild -project TestProject.xcodeproj -scheme TestProject -sdk iphonesimulator  
-destination 'platform=iOS Simulator,name=iPhone 6,OS=9.3' test
```

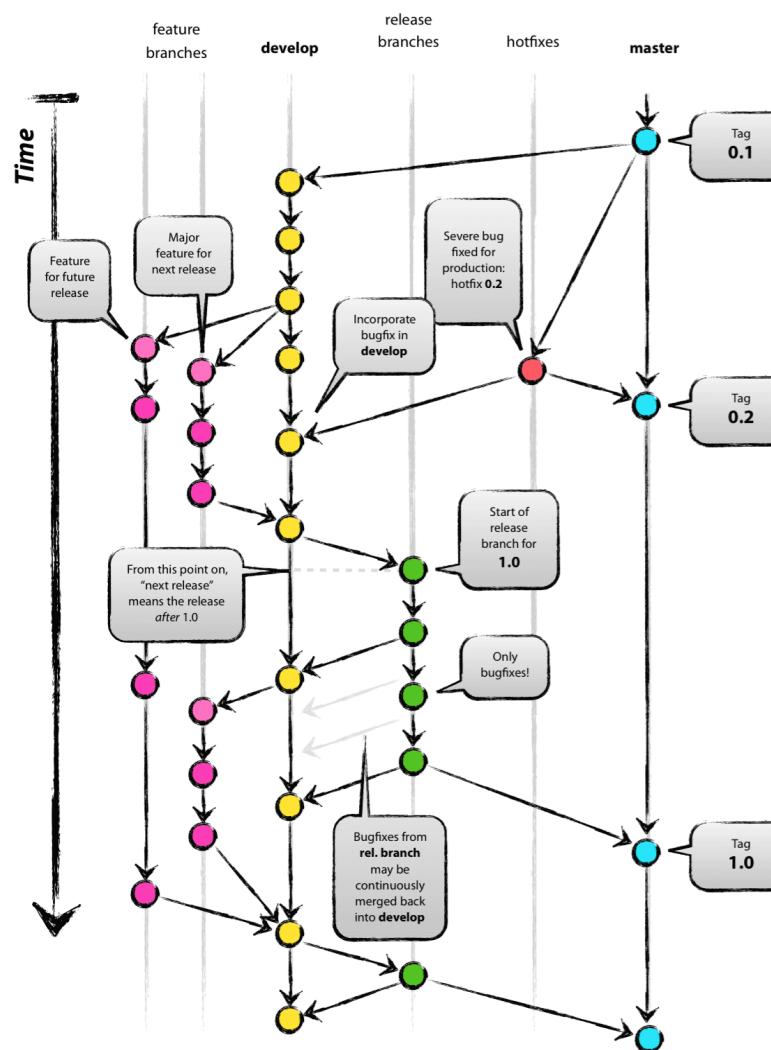
Clean Git History

Regression tracking

Several working trees

Branch management
pro tips

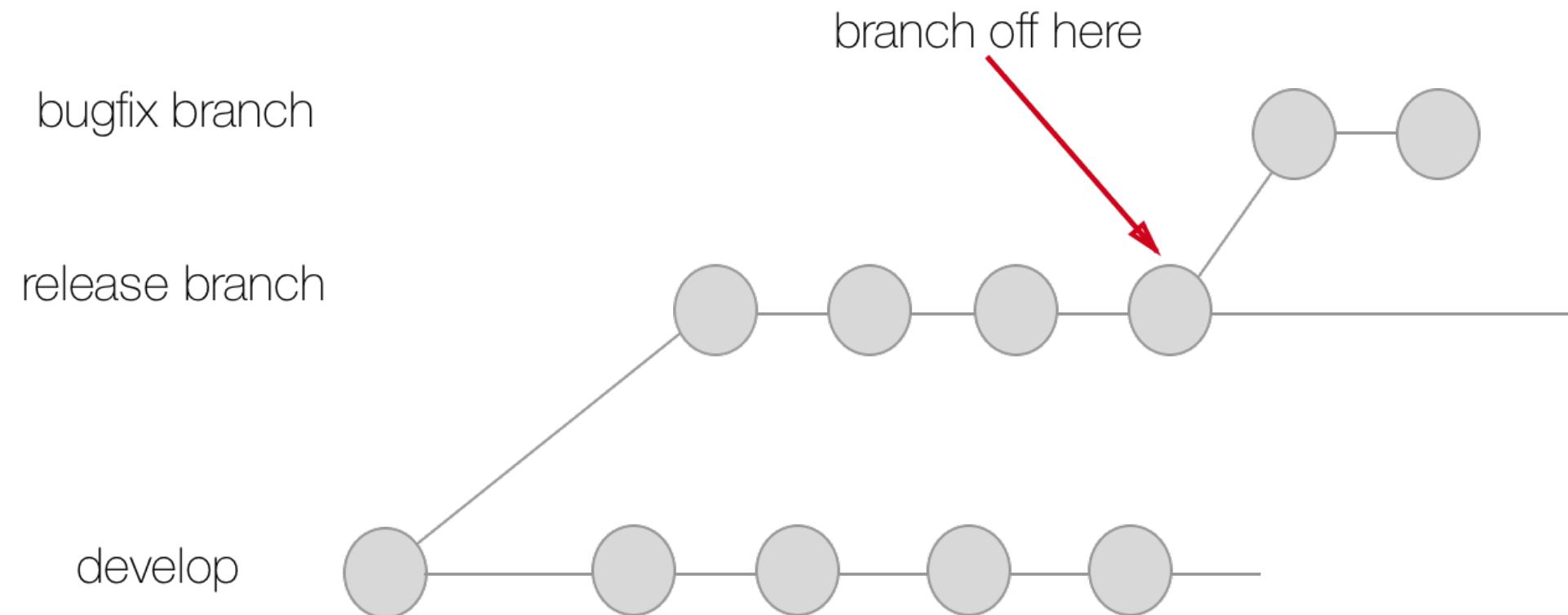
Successful Git Branching Model



Source: <http://nvie.com/posts/a-successful-git-branching-model/>

Proper starting point

- Always branch off from oldest ancestor, to incorporate feature in all children branches

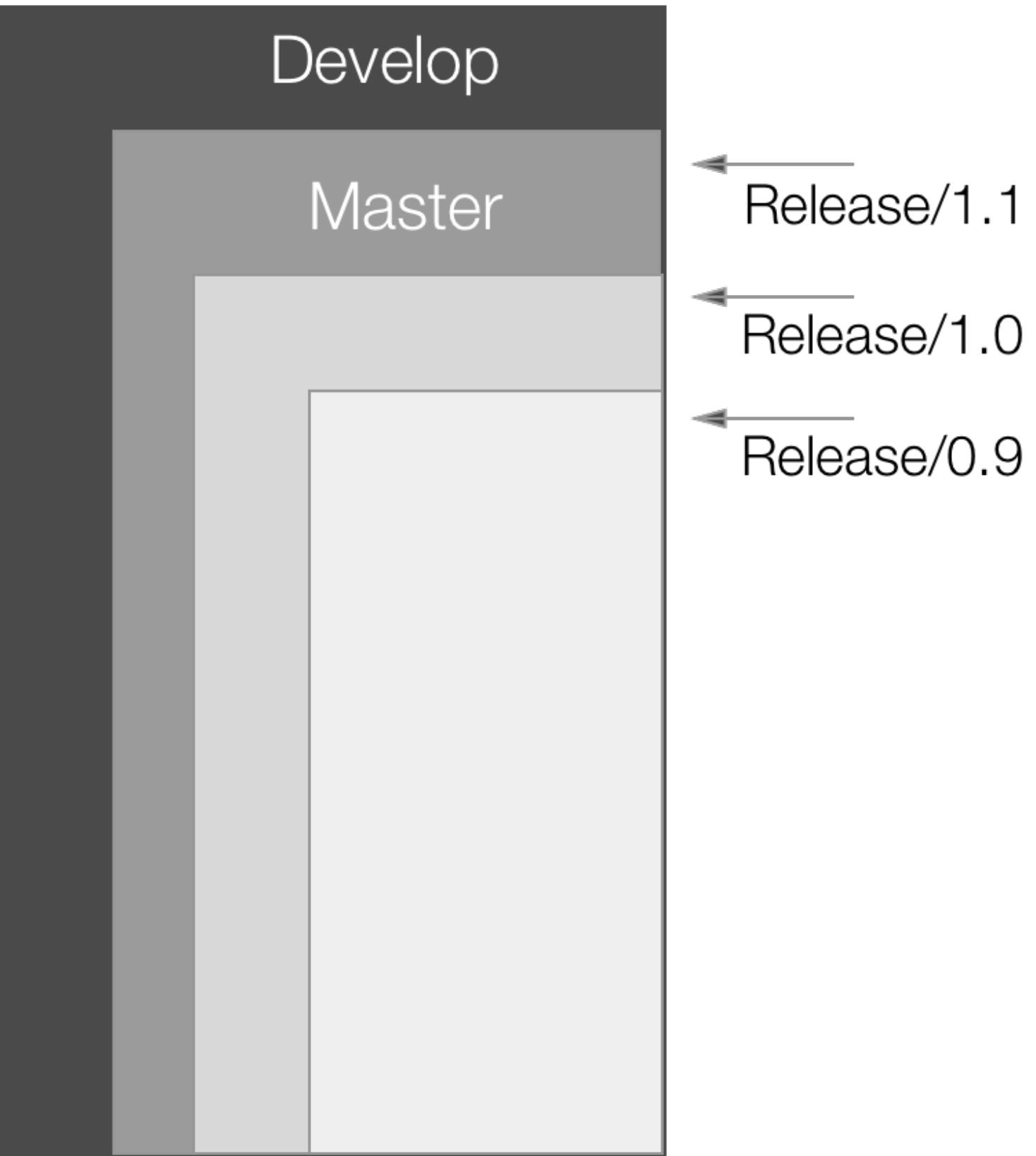


Branch difference tracking (1)

After incorporating changes, this should
ALWAYS return empty results:

```
git log --no-merges  
develop..master
```

```
git log --no-merges  
develop..release-1.8
```



Branch difference tracking (2)

Lists commits that are in release-1.2 but not in master:

```
git log --no-merges master..release-1.2
```

Lists commits that are in develop but not in master:

```
git log --no-merges master..develop
```

Listing not merged branches

Remote branches not merged with develop

```
git branch -r --no-merged origin/develop
```

Local branches not merged with develop

```
git branch --no-merged develop
```

beware of cherry-pick



- Cherry-pick is not a silver bullet
- It generates new hash every time
- Do not cherry-pick changes between develop, master and release- branches because git will treat those changes as independent, not related commits

when to use cherry-pick?

- when there is no other choice and changes can't be integrated easily (backporting)
- when working with your local branches

Clean Git History

Regression tracking

Several working trees

Branch management
pro tips

Several worktrees (1)

Consider following scenario:

- Working on a feature branch
- Urgent hotfix on a master branch

Several worktrees (2)

What would you do?

git stash

git checkout master

git checkout -b hotfix

Several worktrees (3)

There is better way:

```
git worktree add -b hotfix ../hotfix origin/master  
cd ../hotfix  
(bugfixing)
```

```
git push origin hotfix  
(pull request)
```

```
cd ../main  
rm -rf ../hotfix  
(cleanup)
```

what about live demo?



Thank You

Tomasz Gebarowski

@tgebarowski

github.com/tgebarowski/

presentation:

<https://github.com/tgebarowski/mobile-warsaw-git-presentation>