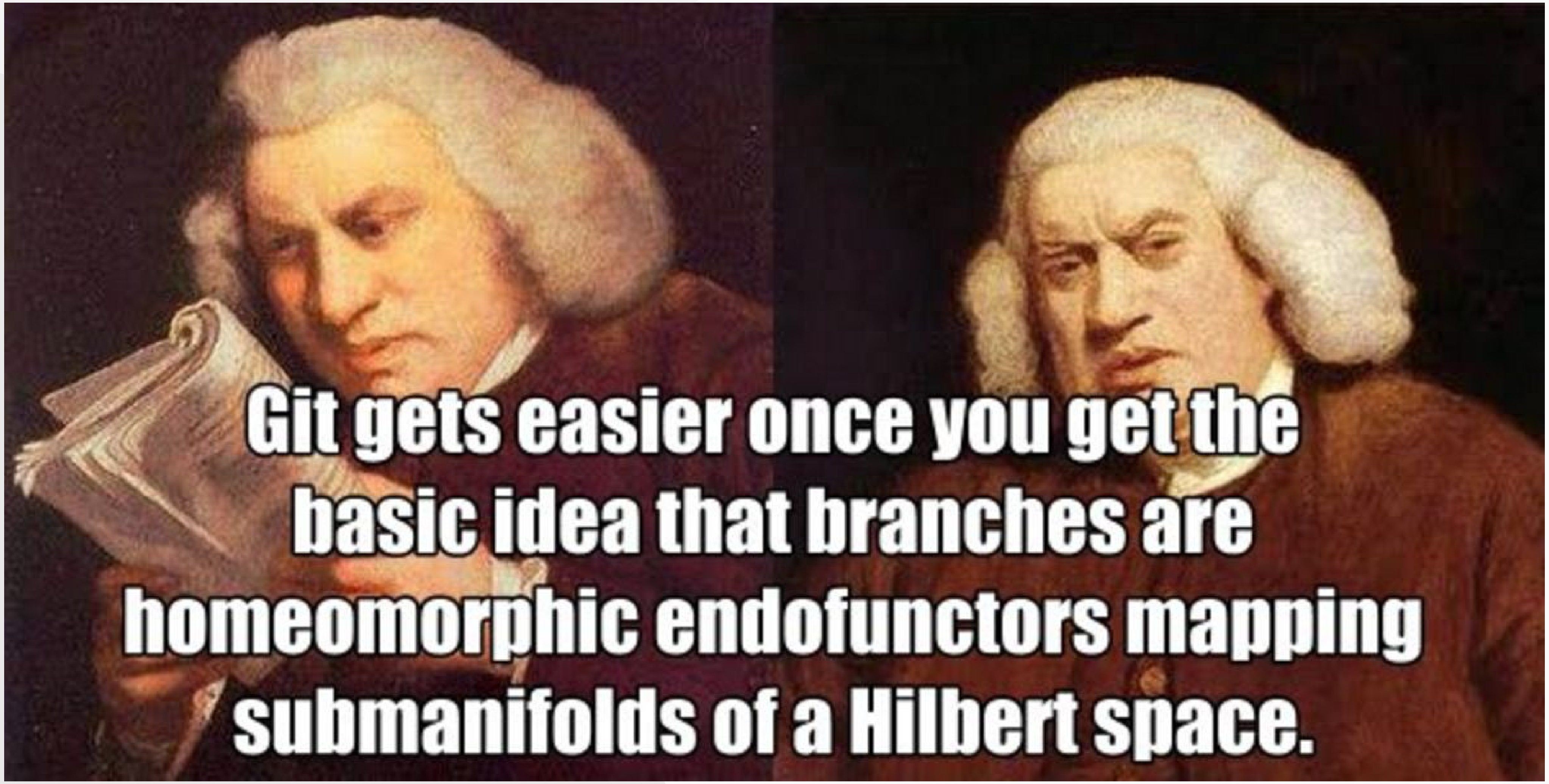
Google Cloud

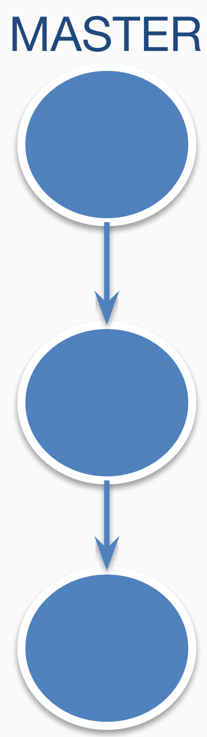
# Edge

## Git Branching Strategy

## Git Branching

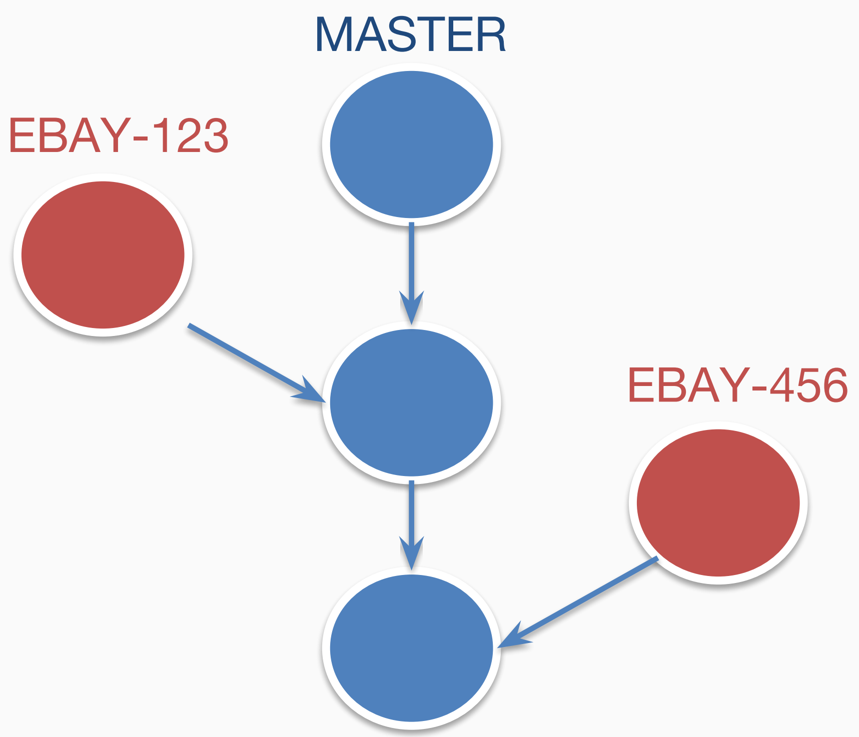

Git gets easier once you get the basic idea that branches are homeomorphic endofunctors mapping submanifolds of a Hilbert space.

## Master Branch



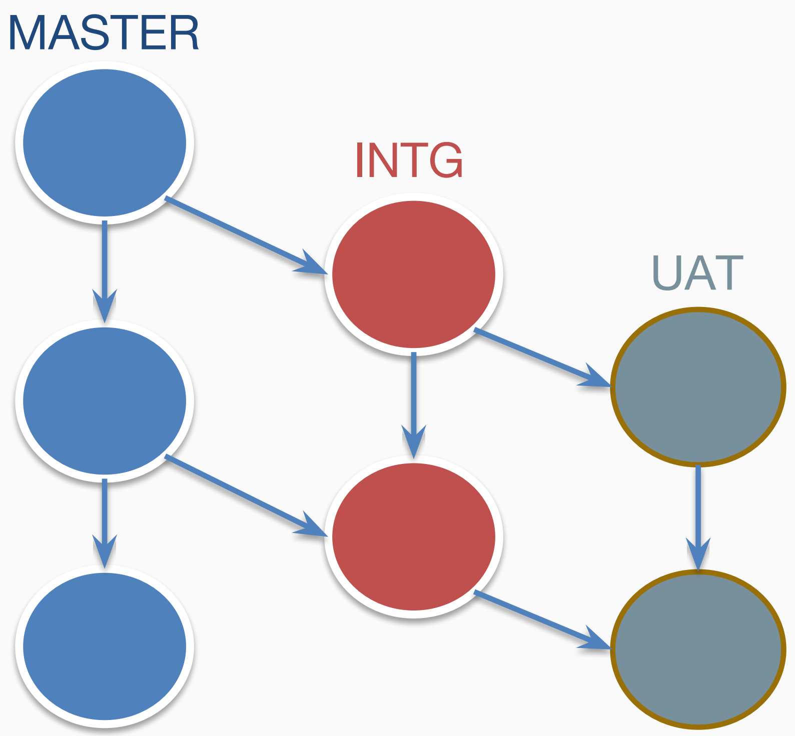
- convention to call default branch master
- most tooling accommodate this concept
- branch from and merge to this

## Feature Branches



- short lived – a day
- named after JIRA issue
- ensures issue tracking
- link back to issue

## Environment Branches



- when deploying to other environment, just merge to downstream branch
- commits flow downstream – ensure every commit follows SCM
- use ‘git diff’ to see changes
- see release history using ‘git log’

## Hotfixes

Scenario 1 – production branch is close to master

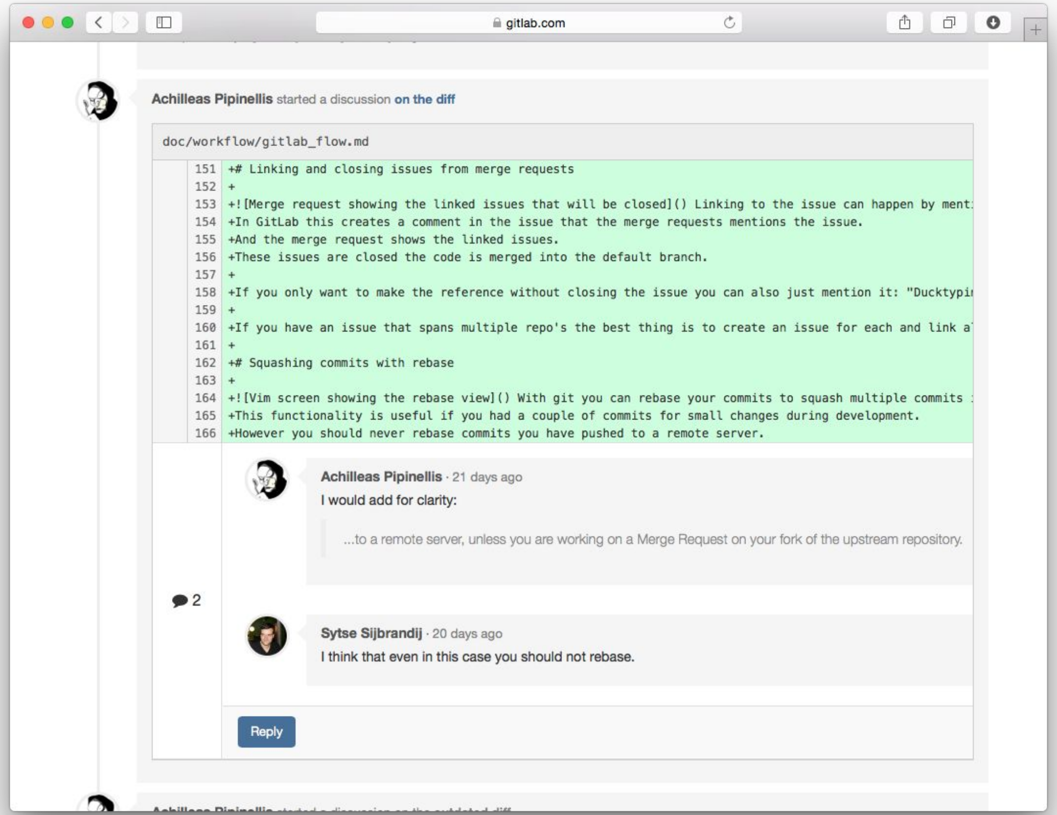
- hotfix on feature branch – tracked with JIRA
- merge to master – test
- cherry-pick to other environments, do work locally
- similar to Linux kernel development
  - Linux -> Subsystem (netdev) -> Device

## Hotfixes

Scenario 2 – hotfix on staging

- hotfix on staging branch – deployed to staging via CI
- merge to production branch – deployed to prod via CI
- cherry-pick back to other environments, do work locally

## Merge Requests



Achilleas Pipinellis started a discussion on the diff

doc/workflow/gitlab_flow.md

```
+# Linking and closing issues from merge requests
+
+![Merge request showing the linked issues that will be closed]() Linking to the issue can happen by ment
+In GitLab this creates a comment in the issue that the merge requests mentions the issue.
+And the merge request shows the linked issues.
+These issues are closed the code is merged into the default branch.
+
+If you only want to make the reference without closing the issue you can also just mention it: "Ducktypi
+
+If you have an issue that spans multiple repo's the best thing is to create an issue for each and link a
+
+# Squashing commits with rebase
+
+![Vim screen showing the rebase view]() With git you can rebase your commits to squash multiple commits
+This functionality is useful if you had a couple of commits for small changes during development.
+However you should never rebase commits you have pushed to a remote server.
```

Achilleas Pipinellis · 21 days ago

I would add for clarity:

> ...to a remote server, unless you are working on a Merge Request on your fork of the upstream repository.

2

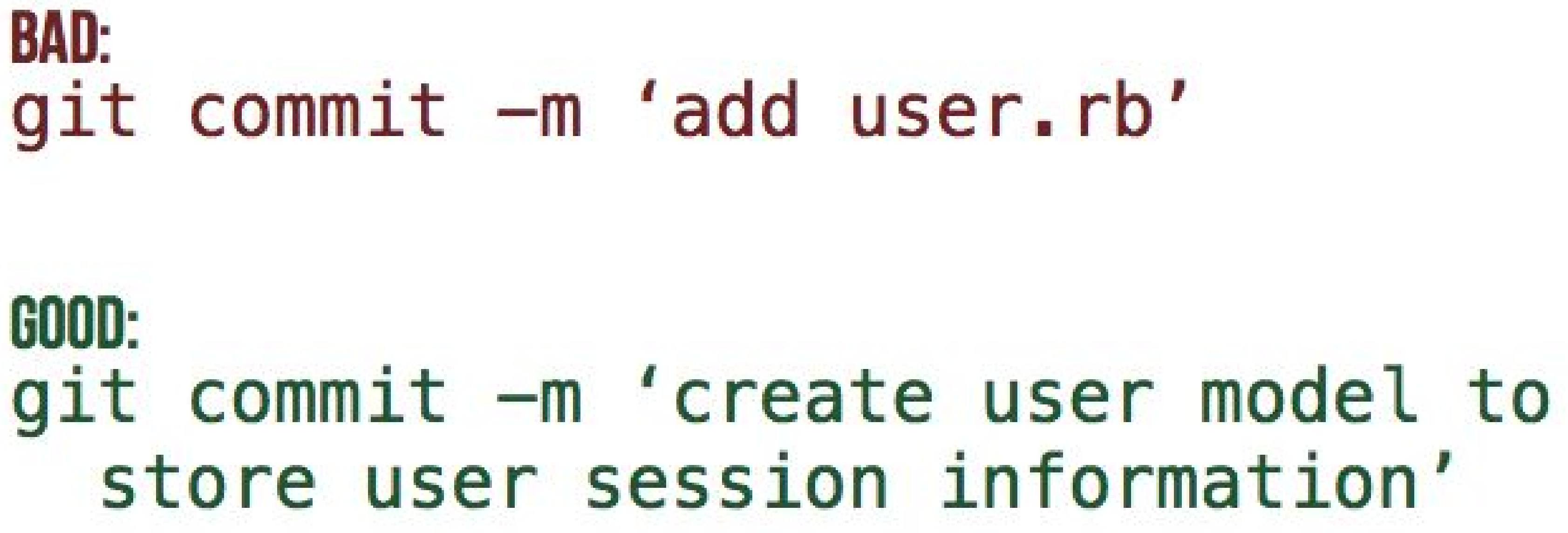
Sytse Sijbrandij · 20 days ago

I think that even in this case you should not rebase.

- online place to discuss and review code
- share your in-progress feature, ask for a review
  - if you work on a branch for more than a few hours
  - Do MR without assigning to anyone. Mention people in comment.
  - code is not ready for merge, but feedback welcome
  - anybody can comment or push changes
- ask for a merge
  - assign MR to somebody

## Commit Often with the Right Message

- commit message to reflect intention, not contents of the commit
    - contents are already visible
    - more important than what is why
- stay away from: change, improve, refactor, fix



**BAD:**

```
git commit -m ‘add user.rb’
```

**GOOD:**

```
git commit -m ‘create user model to
   store user session information’
```

## Merge Master Into Feature to Keep Feature Branches Up-to-date

- why
  - leverage code pushed to master
  - solve merge conflicts
- how
  - cherry-pick individual commit
  - merge master to feature
- when
  - start of the day – CI strategy
  - well-defined points – synchronization point strategy when the state of the code is better known
  - don’t merge at random points – don’t litter the history (Git rebase is your friend)

## Merge Feature into Master when Dev on Feature is Done

- Testing the feature branch on its own may not be enough.  It is not the merged result.
- Merge your feature branch into master and test locally.  Solve conflicts locally.
- CI to incorporate merge requests
- short-lived feature branches help a lot to minimize conflicts