Xebia

The Git you don’t know

More about git than you ever wanted to know

https://norasoft.eu/slides/unknown-git/en/

xebia.com

Michał Borkowski

wielki.borsuk@gmail.com

What we'll cover?

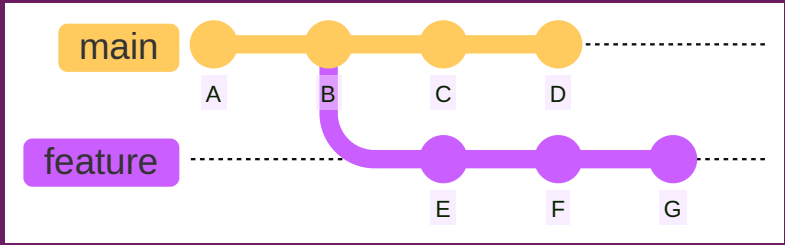
- change management
- large repositories
- objects outside of repository
- operations on git history
- tips & tricks

Git rebase

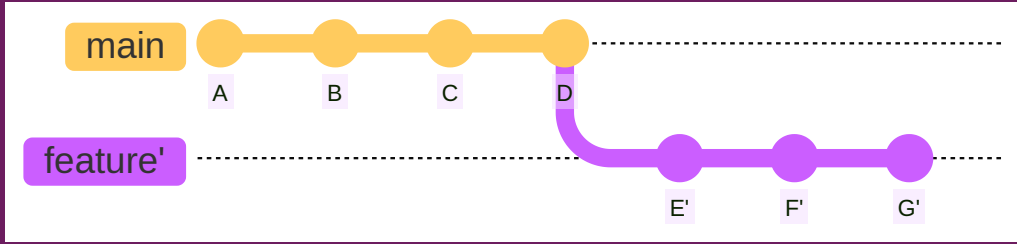
Changing my changes

Rebase

Like a merge, but better and can harm you

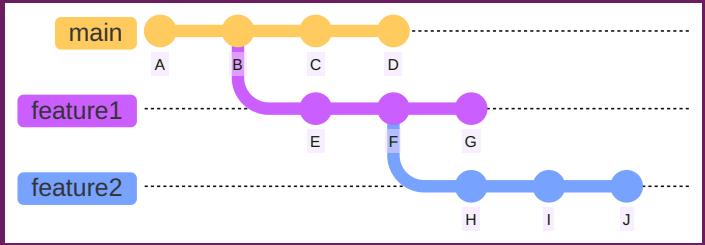



```
git checkout feature
git rebase main
```

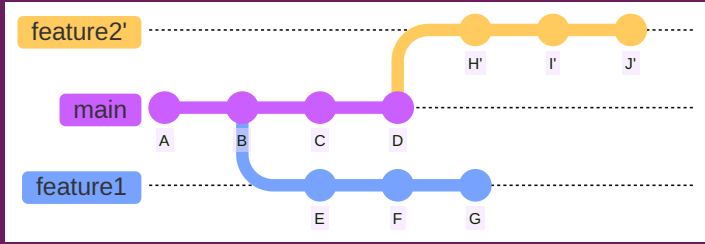


Rebase (–onto)

When you based a branch on a branch

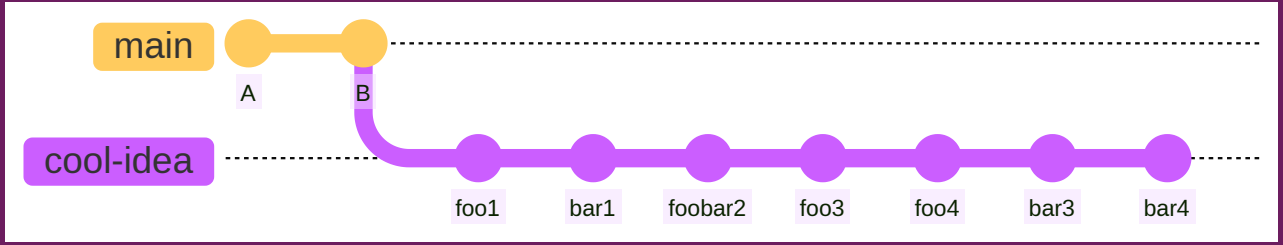



```
git checkout featureB
git rebase --onto main featureA
```



Interactive rebase

When you need to do ANYTHING else




```
git checkout cool-idea
git rebase --i main
```

Interactive rebase

```
pick 83148df foo1
pick cc4c748 bar1
pick b0a03c0 foobar2
pick a36f274 foo3
pick 8e4d113 foo4
pick a3d18de bar3
pick f5a4870 bar4

# Rebase c06a946..a3d18de onto c06a946 (6 commands)
#
# 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 [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# 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
```

Branch split

```
git checkout cool-idea
git checkout -b foo
git rebase -i main
```

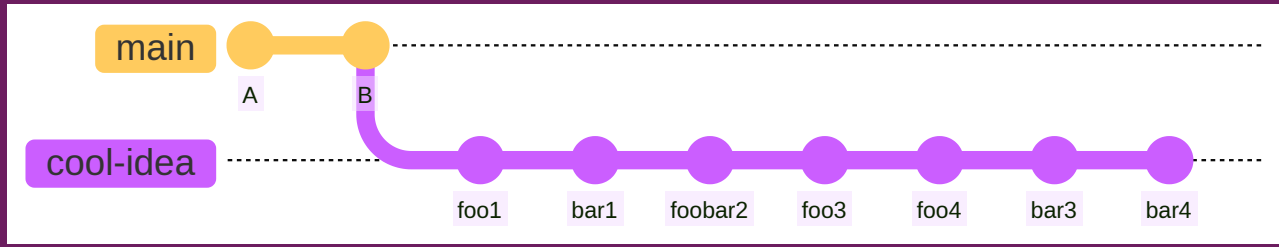
```
pick 83148df foo1
drop cc4c748 bar1
edit b0a03c0 foobar2
pick a36f274 foo3
pick 8e4d113 foo4
drop a3d18de bar3
drop f5a4870 bar4
```

Branch split

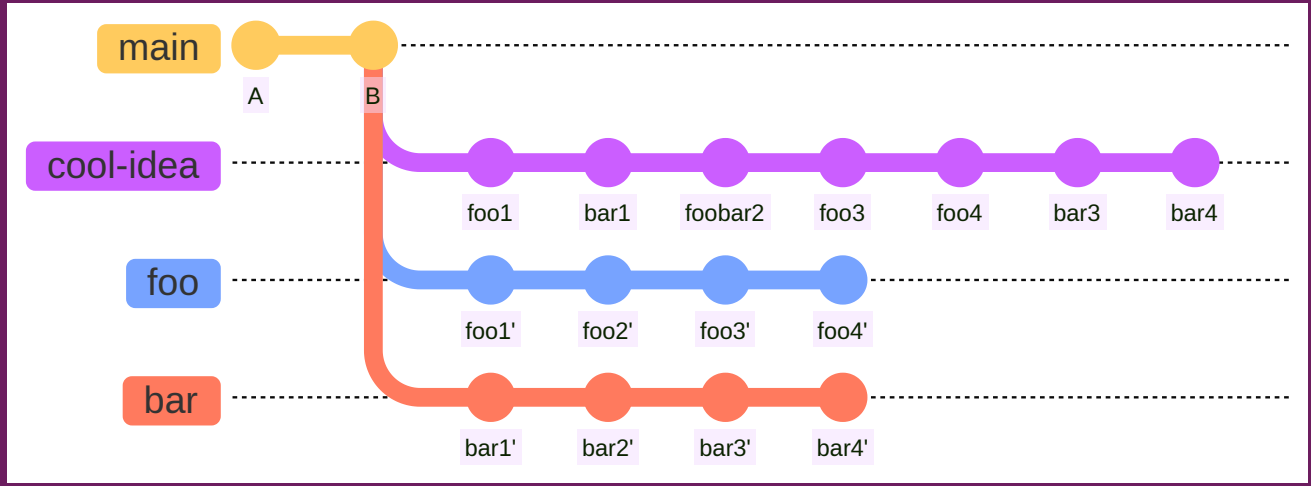
```
git checkout cool-idea
git checkout -b bar
git rebase -i main
```

```
drop 83148df foo1
pick cc4c748 bar1
edit b0a03c0 foobar2
drop a36f274 foo3
drop 8e4d113 foo4
pick a3d18de bar3
pick f5a4870 bar4
```

Before

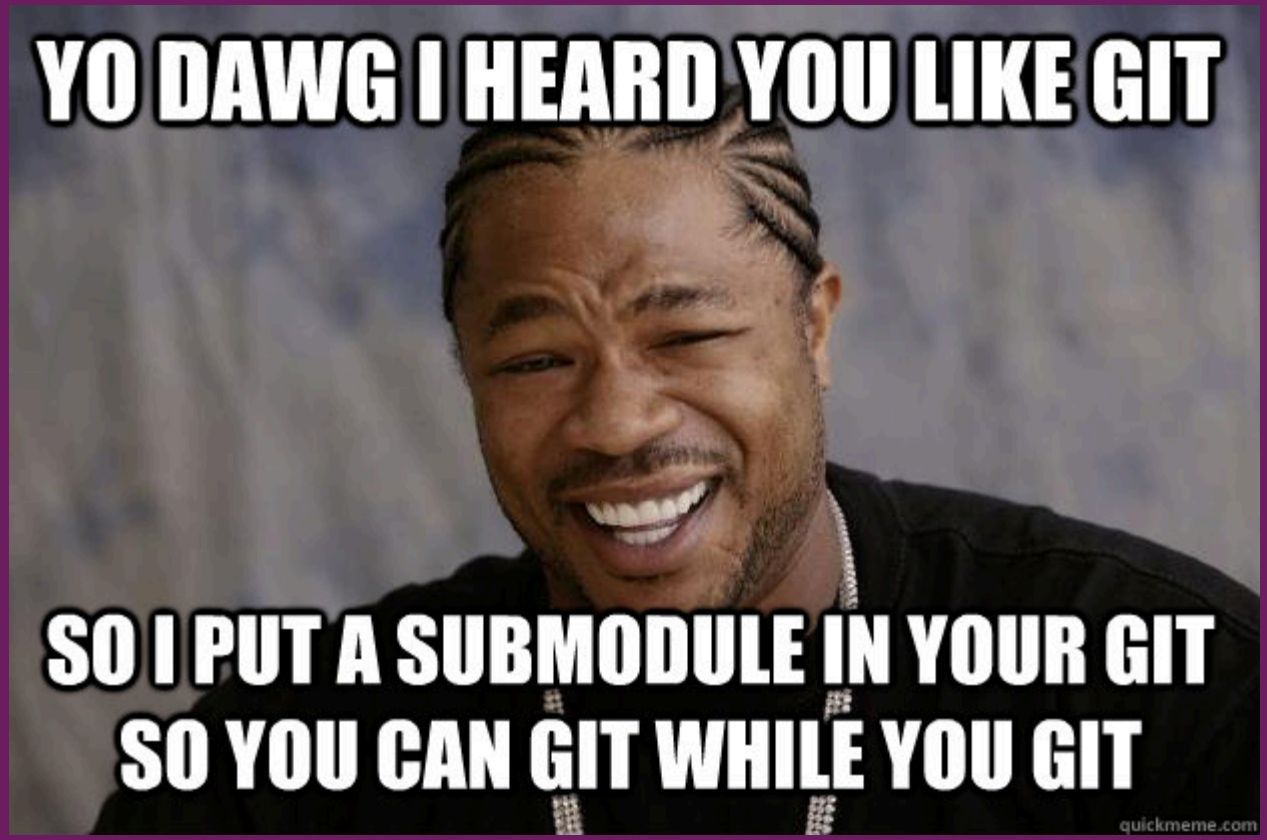


After



When the repository gets too large

Submodules


YO DAWG I HEARD YOU LIKE GIT
SO I PUT A SUBMODULE IN YOUR GIT
SO YOU CAN GIT WHILE YOU GIT
quickmeme.com

Beginning

Add a module

```
git clone git@github.com:user/project.git
cd project
git submodule add git@github.com:user/library.git library

git submodule update --init --recursive
```

Clone a repo with modules

```
git clone --recursive-submodules git@github.com:user/project.git

git pull --recursive-submodules
```

Magic

Make changes

```
cd project/library
git commit -m "library update"
git push

cd ..
git commit -m "update library reference"
git push
```

Get changes

```
cd project
git pull --recursive-submodules
```

Problem

Make changes

```
cd project/library
git commit -m "library update"

cd ..
git commit -m "update library reference"
git push
```

Get changes

```
cd project
git pull
```

Alternatives

```
# requirements.txt for pip
git+https://github.com/user/library@dev-branch
```

```
# Gemfile for ruby
gem 'library', git: 'https://github.com/user/library/', branch: 'dev-branch'
```

```
# main.tf for terraform
module "library" {
  source = "git@github.com:user/library.git?ref=dev-branch"
}
```

```
# go.mod for golang
require (
  github.com/user/library@dev-branch
)
```

Shallow clone

Those who forget the past...

Get only what you need

```
git clone --depth 1 -b main git@github.com/user/project.git
```

```
git fetch origin feature:feature --depth 1
git checkout feature
```

Send changes to the server

```
git fetch origin new-feature:new-feature --depth 1
git checkout new-feature
git commit -m "new feature implementation"
git push
```

Merge when needed

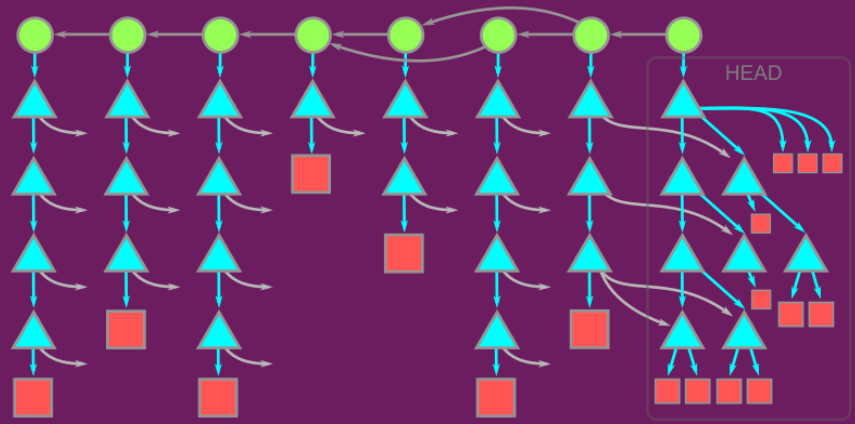
```
git checkout main
git merge --no-ff new-feature
```

```
fatal: refusing to merge unrelated histories
```

```
git fetch mixed-feature --deepen 10
git merge --no-ff new-feature
git push
```

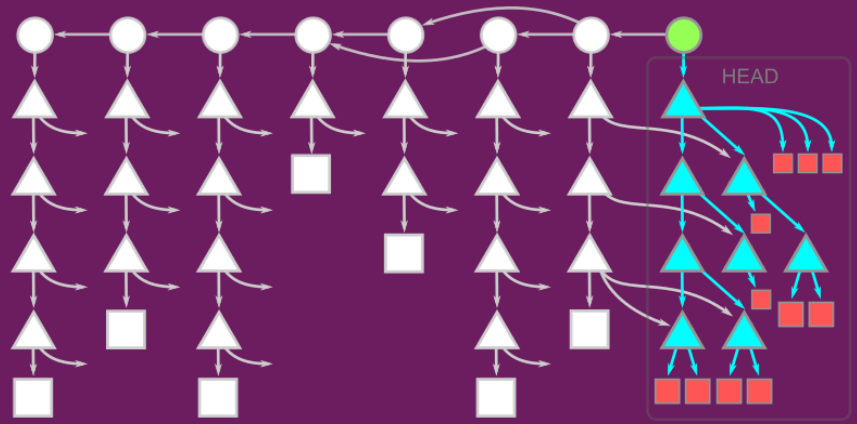
Full clone

```
git clone git@github.com/user/project.git
```



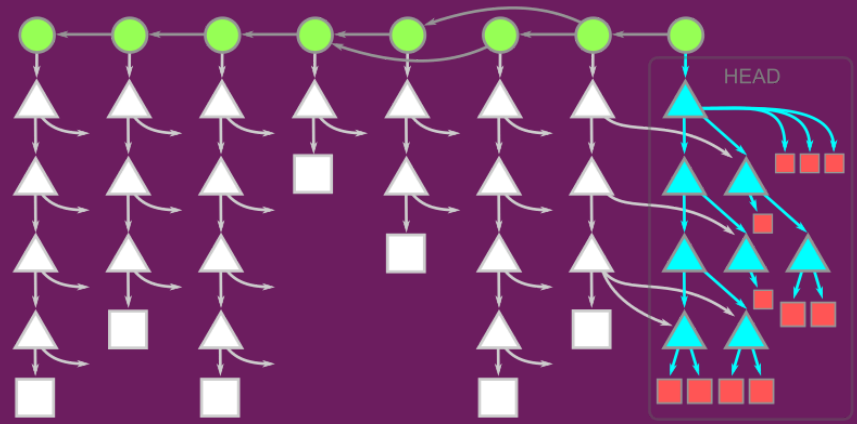
Clone without history

```
git clone --depth 1 \
  git@github.com/user/project.git
```



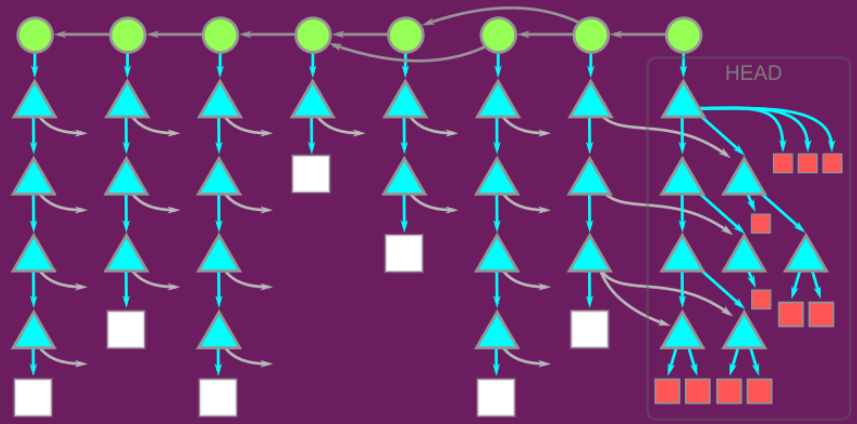
Clone without content

```
git clone --filter=tree:0 \
  git@github.com/user/project.git
```



Clone without objects

```
git clone --filter=blob:none \
  git@github.com/user/project.git
```



Git LFS

So you say your repository is too big?

Where to start?

Installation

```
wget https://github.com/git-lfs/releases/v3.6.1/git-lfs-linux-amd64-v3.6.1.tar.gz
tar xzf git-lfs-linux-amd64-v3.6.1.tar.gz
cd git-lfs-linux-amd64-v3.6.1
./install.sh

git lfs install
```

Activate in your project

```
cd project
git lfs track "*.psd"
git lfs track "*.m4a"
git add .gitattributes
```

How it works?

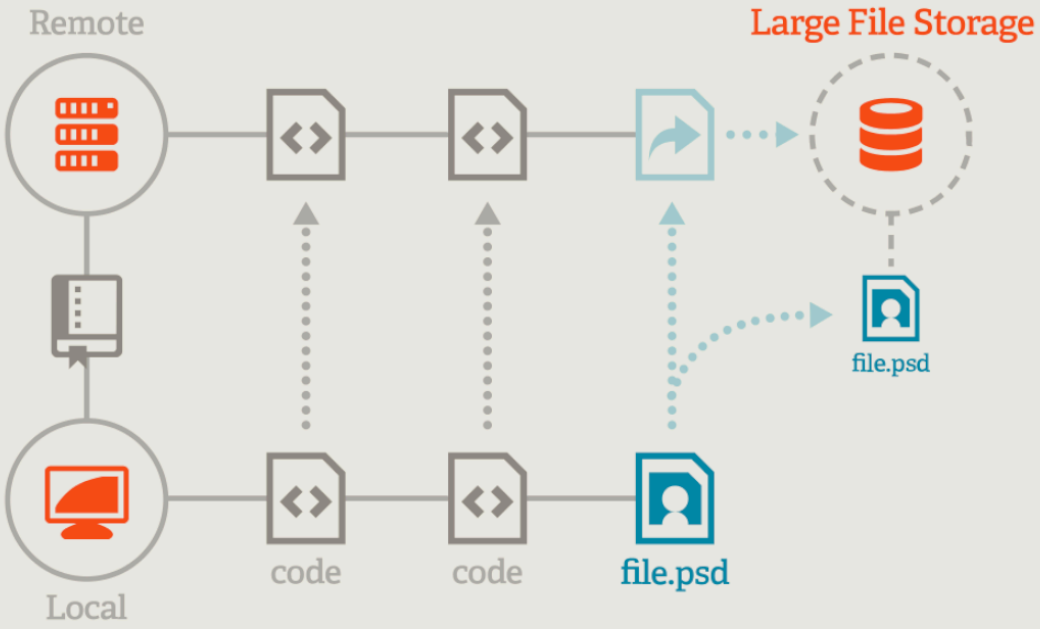
After you add a file

```
cd project
git add assets/sound.m4a
git commit -m "add audio files"
git push
```

What does the file contain?

```
# assets/sound.m4a
+version https://git-lfs.github.com/spec/vi
+id sha256:7194bdd797bde471a6e29b4fa9c8c2278b3c4dadfc5cb2c36d7f4531dc6cb8f
+size 17330
```

Where are the objects then?



Your repo is already huge?

Backward big file management

```
git lfs migrate import --everything --include "*.psd" --include "*.m4a"
```

Only import the biggest files

```
git lfs migrate import --everything --above "20 Mib"
```

Only import local changes

```
git lfs migrate import --exclude-ref="origin/master" \
  --include-ref="master" --above "20 Mib"
```


Submit your changes "Dracarys"
git push --all --force --tags

Safety measures

```
git clone --mirror git@github.com/user/project.git

# if needed
git lfs fetch --all
```

What else you could do with it?

```
# project/.git/config

[lfs]
  url = https://my-s3-proxy.example
  locksverify = false
```

```
# project/storage-submodule-1/.git/config

[lfs]
  url = https://my-backblaze-proxy.example
  locksverify = false
```

```
# project/storage-submodule-2/.git/config

[lfs]
  url = https://my-wasabi-proxy.example
  locksverify = false
```

When things go wrong

Git filter-branch

History is written by the victors

When something should disappear

```
git filter-branch --tree-filter 'rm -f .awscredentials'
git filter-branch --tree-filter 'rm -f silly-movie.mp4'
```

```
git filter-branch --tree-filter \
  'sed -i readme.md "s/embarrassing secret/<secret placeholder>/"'
```

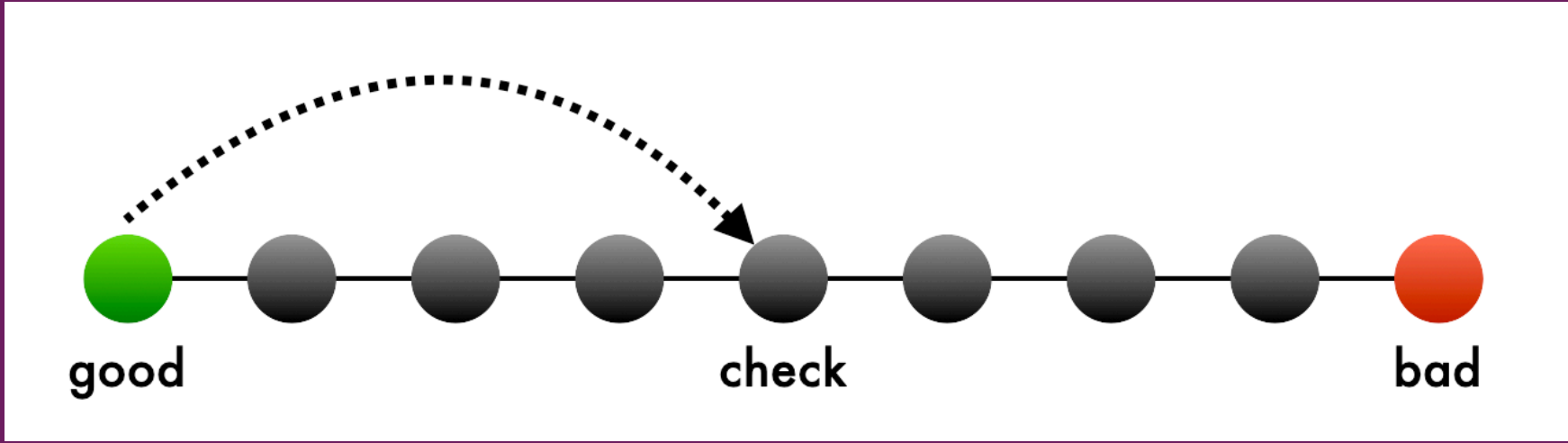
When someone should disappear

```
git filter-branch --commit-filter '
  if [ "$GIT_COMMITTER_NAME" = "MyEx" ];
  then
    GIT_COMMITTER_NAME="Me";
    GIT_AUTHOR_NAME="Me";
    GIT_COMMITTER_EMAIL="me@email.com";
    GIT_AUTHOR_EMAIL="me@email.com";
    git commit-tree "$@";
  else
    git commit-tree "$@";
  fi' HEAD
```

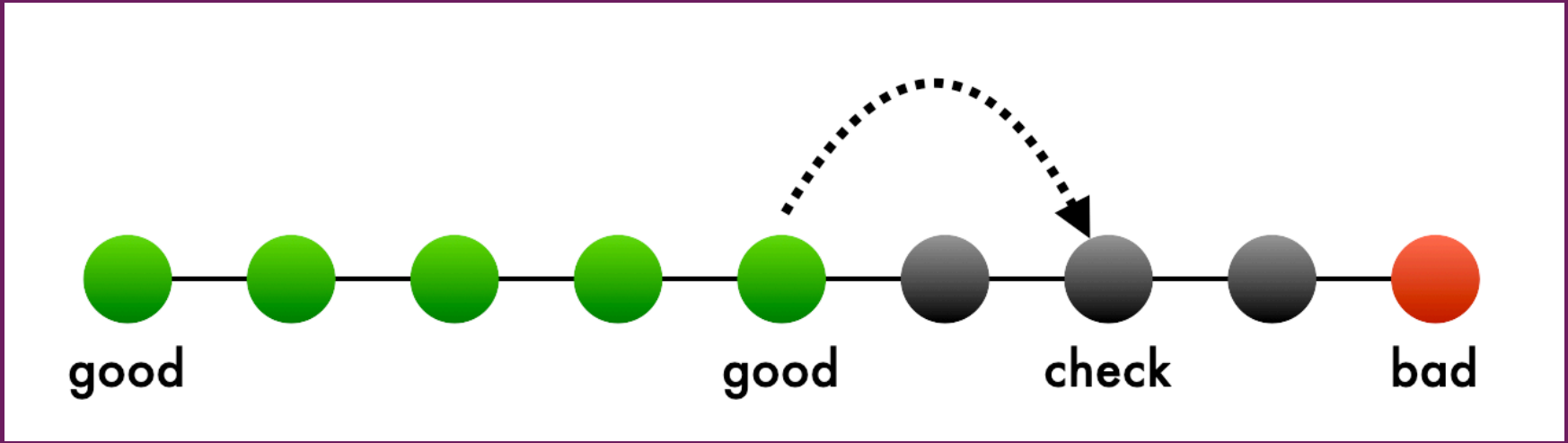
Git bisect

Blame assignment for advanced users

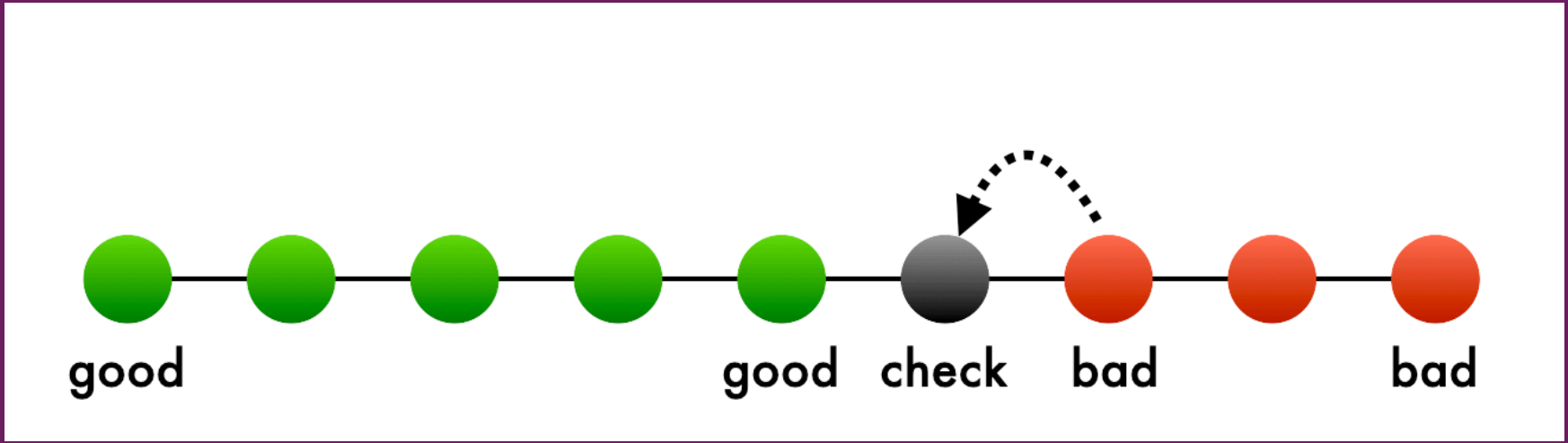
Procedure



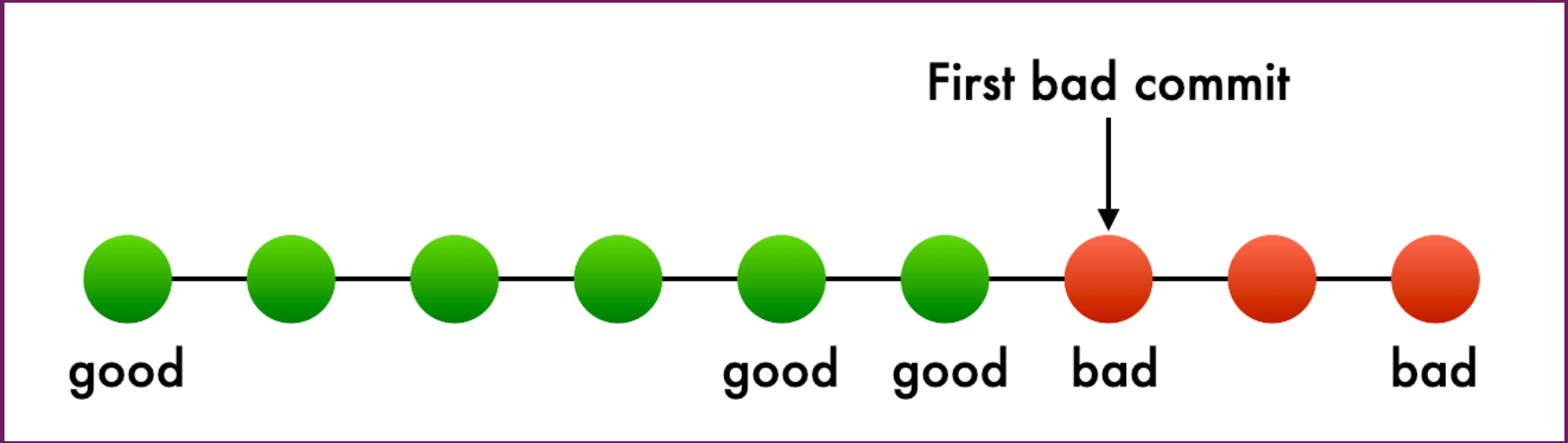
Procedure



Procedure



Procedure



How to use it?

```
# Slowly
git bisect start
git bisect bad
git bisect good origin/main

rake test
git bisect good
rake test
git bisect bad
rake test
git bisect bad
...

git bisect reset
```

```
# Fast
git bisect start HEAD origin/main
git bisect run rake test
```

When were credentials exposed?

```
git bisect start HEAD <initial commit>
git bisect run ./check-for-absence.sh
```

HEAD always needs to be marked as "BAD" so the test needs to fail for current code

```
# ./check-for-absence.sh
grep -R "secret password value" && exit 1 || exit 0
```

It's better to keep your custom script outside the repo

```
git bisect run ../check-for-absence.sh
```

Is there anything else?

Tagged object

Why commit your files?

When you can tag anything

Not only commits but any hash that exists in your repository

```
cd project
new_file_id=$(git hash-object -w <file>)
git tag hidden-file $new_file_id
git push --tags
```

```
git tag
file_id=$(cat .git/refs/tags/hidden-file)
git cat-file -p $file_id
```

Git worktree

When a single branch is simply not enough

```
cd project
git worktree add ../project-main main
git worktree add ../project-feature2 feature2
```

```
cd ../project-main
ls .git/

  > ls: cannot access '.git/': Not a directory
```

```
cat .git

  > gitdir: /<absolute-path-to-repo>/project/.git/worktrees/project-main
```

Who needs git stash, when you can check out multiple branches at once.

References for insomniacs

- https://blog.entrostat.com/breaking-up-a-git-branch-into-multiple-features/ - split branch with rebase
- https://dbushell.com/2024/07/15/replace-github-lfs-with-cloudflare-r2-proxy/ - git lfs configured for different backends
- https://git-scm.com/book/en/v2/Git-Internals-Git-References#_tags - low level mechanisms around tagging
- https://github.blog/open-source/git/get-up-to-speed-with-partial-clone-and-shallow-clone/ - different clone types and how to use them
- https://git-scm.com/docs/git-worktree - one repo, multiple branches

Xebia

Thank you!

https://norasoft.eu/slides/unknown-git/en/

wielki.borsuk@gmail.com