

**DONT
UNDERESTIMATE
THE push --force**



Noaa Barki



BarkiNoaa



PROBLEM

Speed vs. Control

How to coordinate and ensure alignment among developers without implementing excessive (heavyweight) control?



datree.io

**Datree enable modern software development
without compromising on operational efficiency**

How Datree works

1

SCAN

Identify issues in your code repositories

2

SET

Set policy rules to prevent future occurrences

3

ENFORCE

Automatically run policy checks on pull requests

Enforce Best Practices throughout your Tech Stack



DOCKER



KUBERNETES



SERVERLESS



CI/CD



JENKINS



GIT FLOWS



ANY CUSTOM POLICY...

12

Example - Docker rules pack:

Policy rules

☒ Docker: LABEL: Maintainer: property should exist

✗ The following properties don't match the required pattern:

• Dockerfile

☒ Docker: USER property should exist

✗ The following properties don't match the required pattern:

• Dockerfile

☒ Docker: apt: installed packages should have a version

✗ The following properties don't match the required pattern:

• Dockerfile RUN apt-get install --no-install-recommends postgresql-client

☒ Docker: USER property should not be root

✗ The following properties don't match the required pattern:

• Dockerfile

☒ Docker: FROM: every image should be pulled from datree-docker:frog.io

✗ The following properties don't match the required pattern:

• Dockerfile FROM node:latest

☒ Docker: FROM: should have a tag and it shouldn't be latest

✗ The following properties don't match the required pattern:

• Dockerfile FROM node:latest



datree.io

How Datree works

1

SCAN

Identify issues in your code repositories

2

SET

Set policy rules to prevent future occurrences

3

ENFORCE

Automatically run policy checks on pull requests

The screenshot displays the Datree.io platform interface across three main sections:

- Dashboard:** Shows a sidebar with navigation links: Dashboard, POLICY (Reports, Policy Management, Pull Requests, Notifications), VISIBILITY (Code Components, People, Repositories), and a search bar.
- Smart Policy Management:** A central panel titled "Smart Policy Management" featuring five policy cards:
 - Separate secret credentials from source code (Security, Docker, Kubernetes) - 24 red, 51 green
 - Separate personal config files from source code and other stuff (Security, Docker, Kubernetes) - 96 red, 21 green
 - Separate secret credentials from source code (Security, Docker, Kubernetes) - 87 red, 33 green
 - Ensure a .gitignore file is included in projects (Security, Docker, Kubernetes) - 63 red, 57 green
 - Ensure CODEOWNERS defined in projects (Security, Docker, Kubernetes) - 34 red, 67 green
- Repository Enforcement:** A right-hand panel titled "Separate secret credentials from source code" which is "ENABLED". It lists repositories being monitored:
 - arbel/figma-plugins (PASSED)
 - arbel/figma-plugins (FAILED)
 - arbel/figma-plugins (PASSED)
 - arbel/figma-plugins (PASSED)A checkbox at the bottom enables enforcement on new repositories by default.

How Datree works

1

SCAN

Identify issues in your code repositories



2

SET

Set policy rules to prevent future occurrences

3

ENFORCE

Automatically run policy checks on pull requests

A screenshot of a GitHub pull request page. At the top right, there's a 'Review required' badge with a red 'X' and the text 'At least 1 approving review is required by reviewers with write access. [Learn more.](#)' Below it is a 'Some checks were not successful' badge with a green circle and red text, indicating '1 failing and 1 successful checks'. Two specific checks are listed: 'Datree Smart Policy' (failing after 5s — Best Practices Verification) and 'Datree insights' (successful in 6s — datreeio insights events). Both of these have 'Required' status and 'Details' links. At the bottom, there's a 'Merge pull request' button with a dropdown arrow, and a note: 'As an administrator, you may still merge this pull request.' To the right of the button, it says 'You can also open this in GitHub Desktop or view command line instructions.'

- Review required**
At least 1 approving review is required by reviewers with write access. [Learn more.](#)
- Some checks were not successful**
1 failing and 1 successful checks
- Datree Smart Policy** Failing after 5s — Best Practices Verification
Required [Details](#)
- Datree insights** Successful in 6s — datreeio insights events
Required [Details](#)
- Merging is blocked**
Merging can be performed automatically with 1 approving review.

As an administrator, you may still merge this pull request.

Merge pull request ▾ You can also open this in GitHub Desktop or view command line instructions.

Agenda

1. The git push command - How it works
2. The --force flag - Why and when to use and
3. The --force flag - Why it got bad reputation
4. Alternatives for safer push --force
5. How to recover for destructive use of push --force





\$ push



\$ rebase



\$ push



\$ push --force





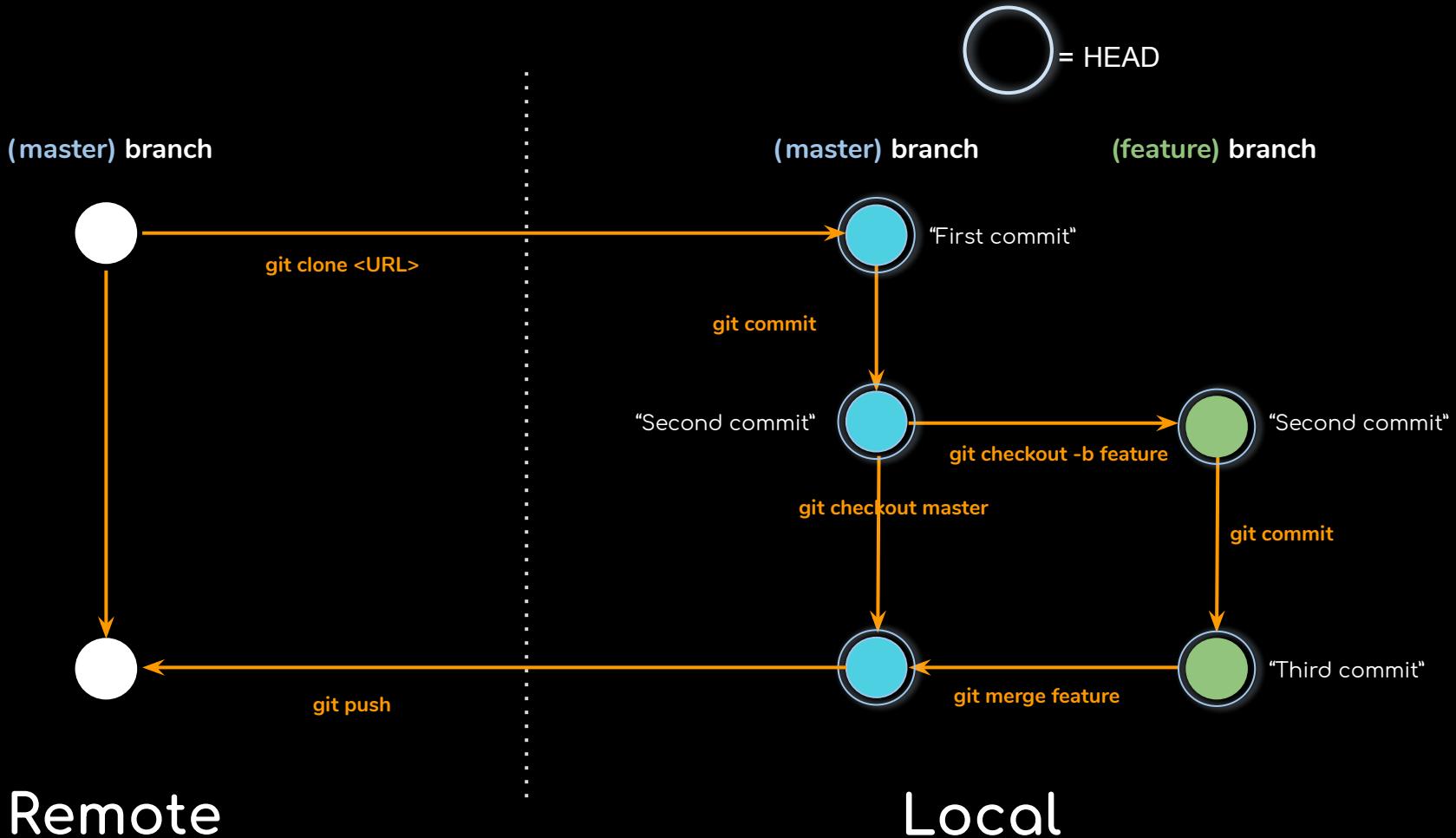
**Get Push
To
Share
Your work
With the world**



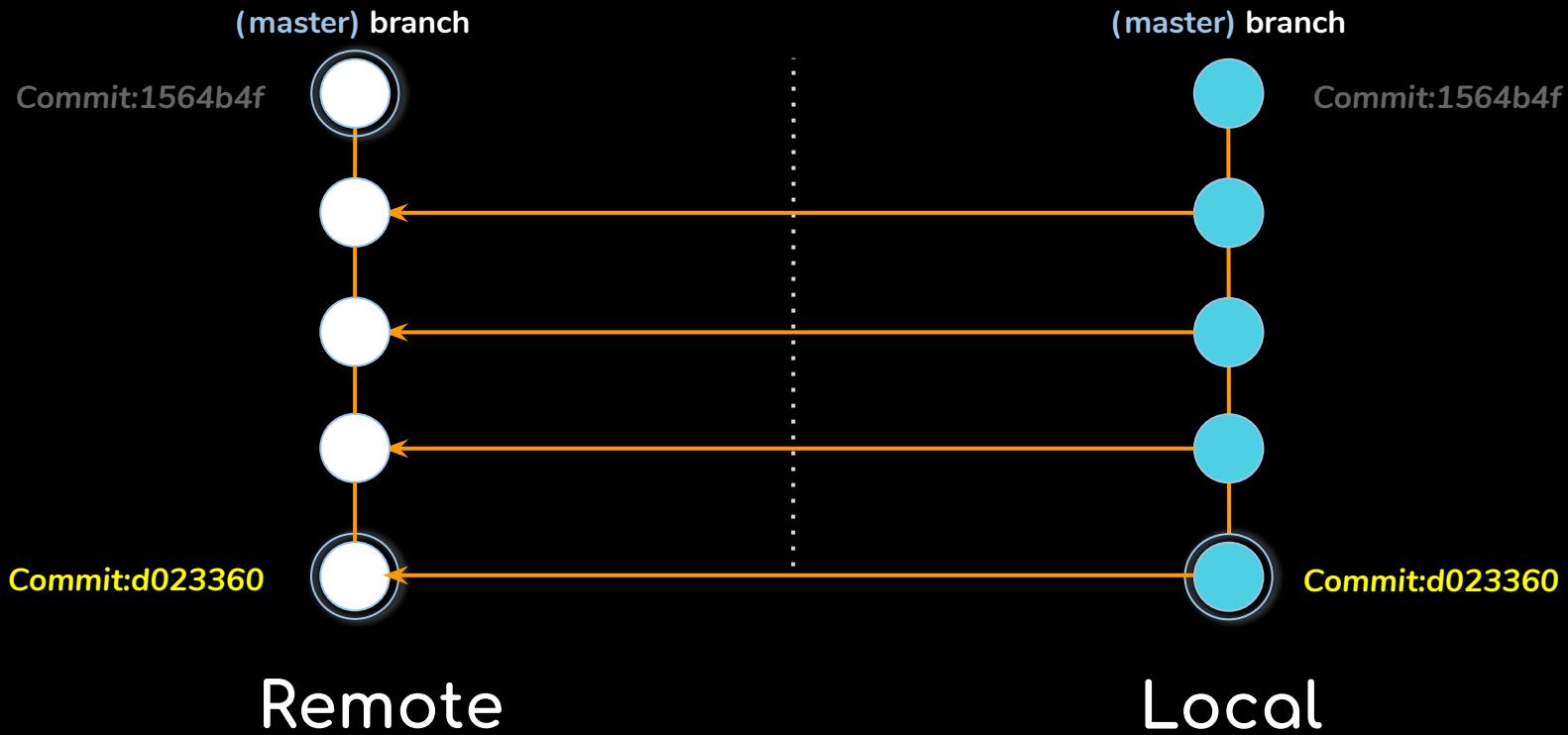
git push

*"updates remote refs using local refs,
while sending objects necessary to
complete the given refs."*

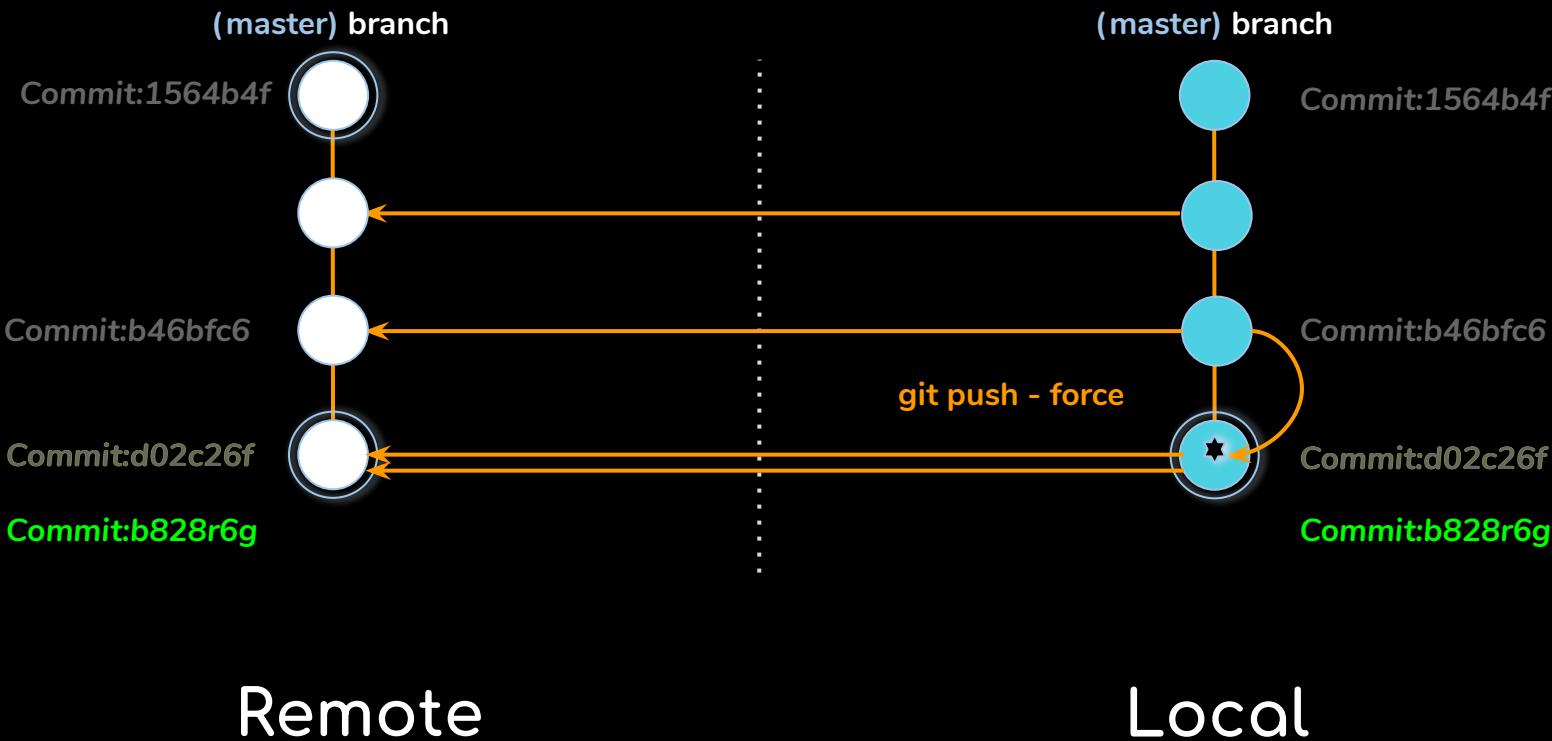
Git Documentation



git push



git commit --amend



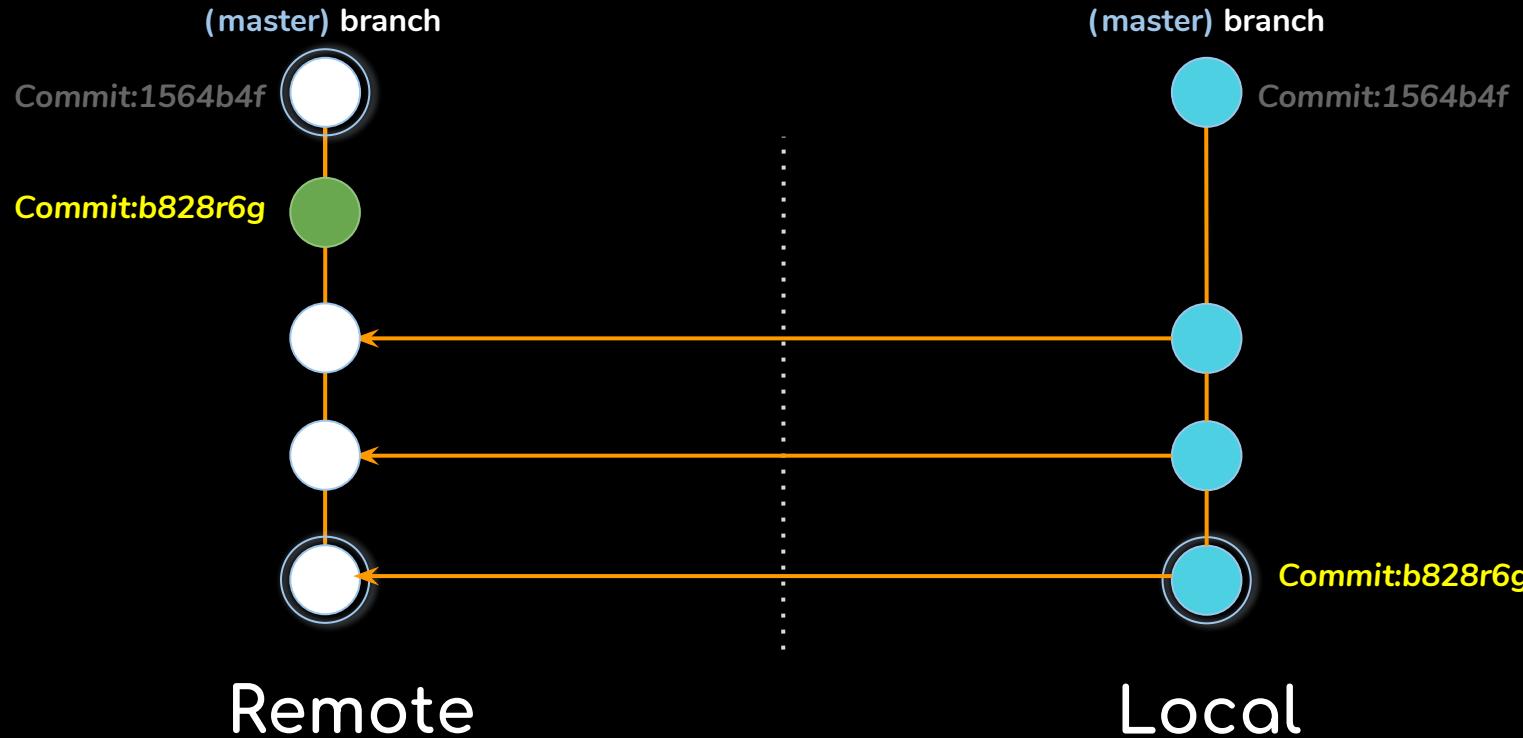
--force **MATCHES** the remote
branch with the local

A close-up photograph of an owl's face. The owl has white and brown feathers. Its left eye is wide open, showing a bright orange iris, while its right eye is closed. The beak is dark and slightly open.

So --force it

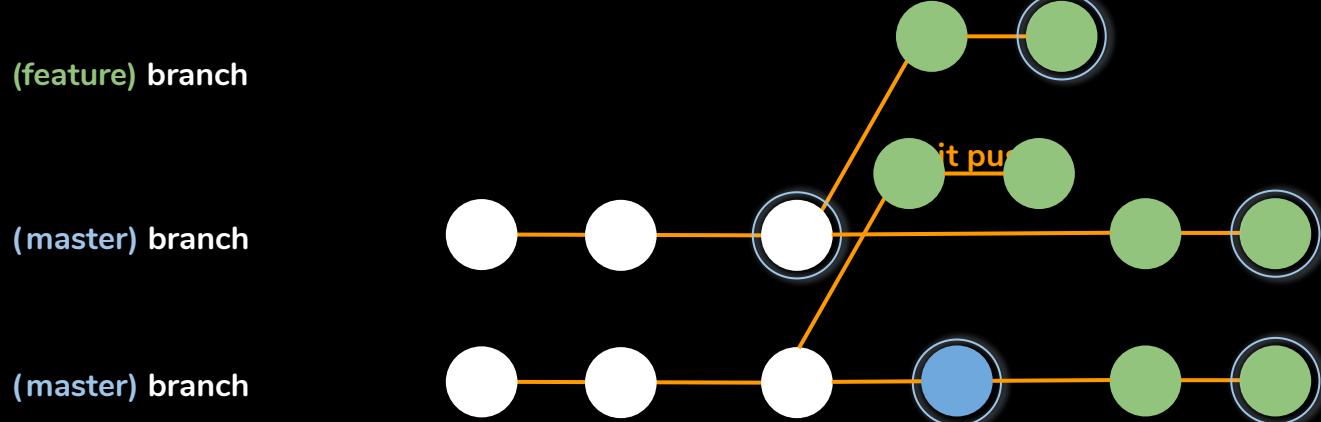
But be wise

git push --force

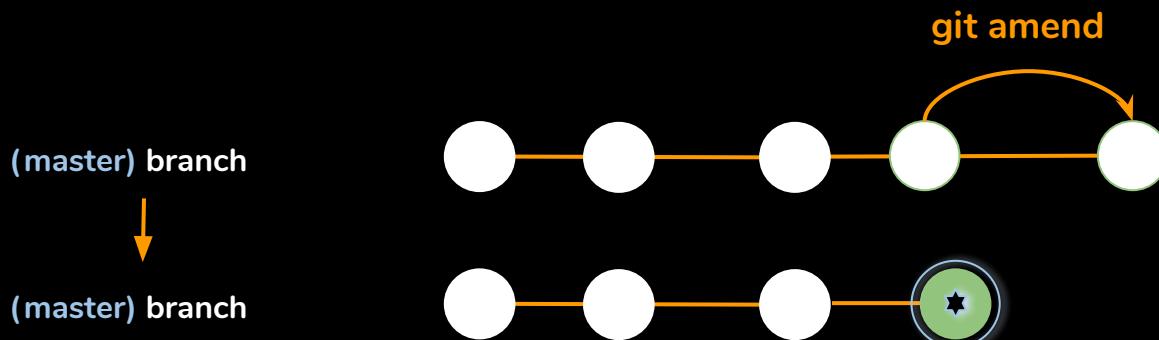


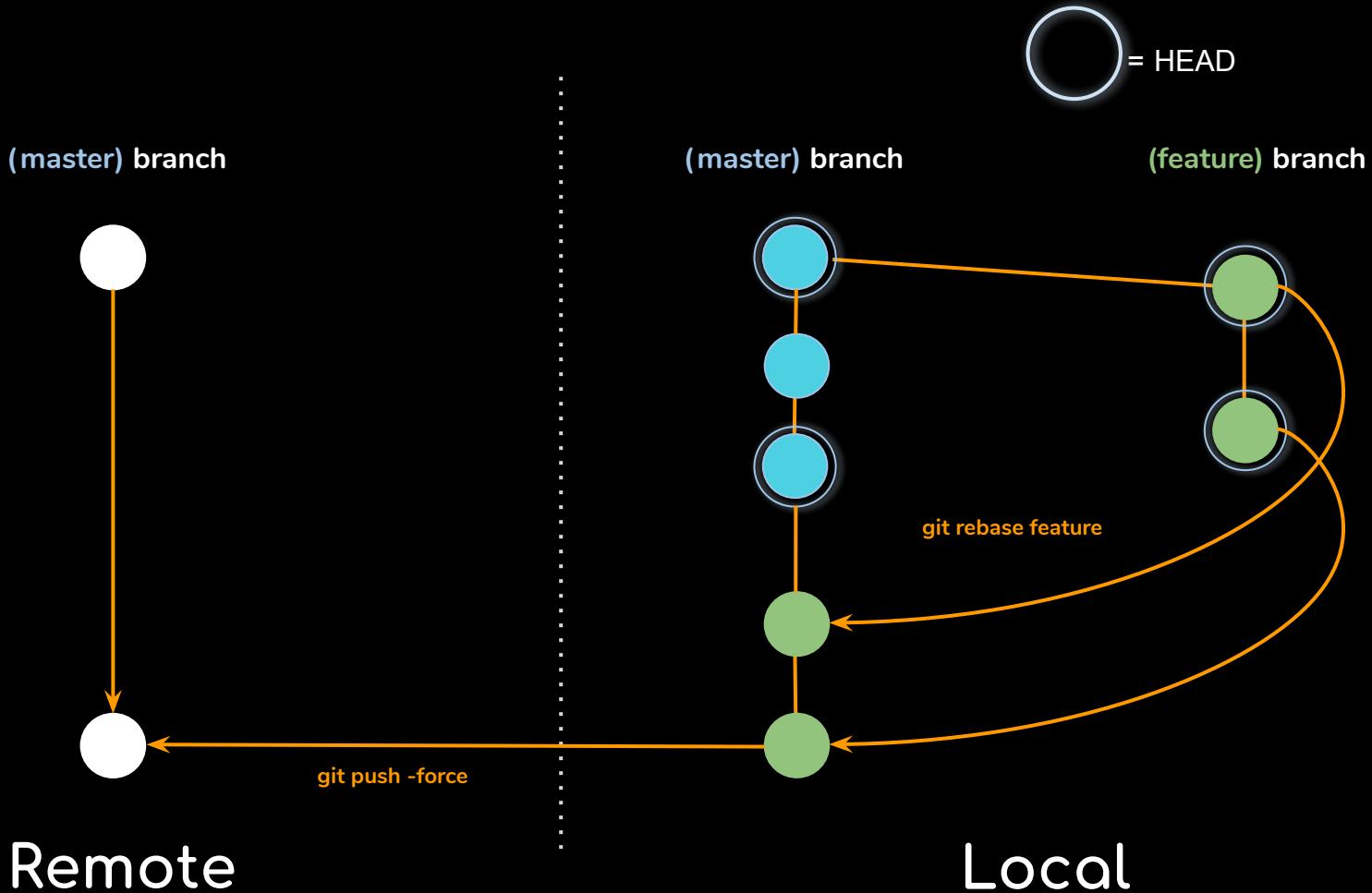
fast forward

shift the branch HEAD



commit --amend





But still,Bounce it

--force-with-lease

-force-with-lease

--force-with-lease

--force-with-lease=<refname>

Example: **--force-with-lease=master:#tag master**

refname

expected

\$ rebase



\$ push



\$ push --force



\$ push --force-with-lease



A close-up photograph of a dog's muzzle, showing its dark, textured fur and a small, bright orange tick on its chin. The dog has dark eyes and a slightly open mouth. Overlaid on the image is white text in a sans-serif font.

So --force-with-lease it

But be wise



 someone pulled a recent version of the master just before?

 push --force the recent version



You were the last to push to master?

- 👉 Do not clear your terminal
- 😢 Confess
- 🗣 Ensure no one mess with repo

```
Welcome@Welcome-PC MINGW64 /e/git-demos/push-test (master)
$ git push origin master --force
Counting objects: 2, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 238 bytes | 238.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/git-test-jaz/push-tst.git
 + 42884b4..a4ee42d master -> master (forced update)
```



push --force origin 42884b4:master

 I accidentally --force my repo, and I want to go back to the previous version. What do I do?



git refLog

git refLog

```
1. 1b46bfc65e (HEAD -> test-branch) HEAD@{0}: rebase -i (finish): returning to  
refs/heads/test-branch  
2. b46bfc65e (HEAD -> test-branch) HEAD@{1}: rebase -i (squash): a  
3. dd7906a87 HEAD@{2}: rebase -i (squash): # This is a combination of 2 commits.  
4. a3030290a HEAD@{3}: rebase -i (start): checkout refs/heads/master  
5. 0c2d866ab HEAD@{4}: commit: c  
6. 6cab968c7 HEAD@{5}: commit: b  
7. a3030290a HEAD@{6}: commit: a  
8. c9c495792 (origin/master, origin/HEAD, master) HEAD@{7}: checkout: moving from master to  
test-branch  
9. c9c495792 (origin/master, origin/HEAD, master) HEAD@{8}: pull: Fast-forward
```

👉 reset --hard HEAD@{4}

👊 push --force origin test-branch



General recover

1. Get the previous commit via terminal, refLog...
 2. Create branch or reset to the previous commit
 3. Push --force origin master
- 👉 Reset --hard origin/<new-branch-name>



You saved the day!



Restore after intentionally push --force

Let's say...

-  You own a Git server repository server
-  You had a developer that wrote a project for you
-  For some reason the developer got really angry
-  The developer deleted all repos, and --forceed “*Ha Ha The project was here*”
-  The developer escaped from the country
-  Leaving you without any code and you have never cloned the repo before



Restore after intentionally push
--force

✗ Sadly git log won't work for us



Look for unreachable commits - Dangling Commits!

👉 Run `git fsck --lost-found`

👉 Check it out by run `git show <commit>`

👉 Finish with following 'General Recover'



KEEP
CALM
AND

MARK PROTECTED BRANCHES

It can happen to anyone



ask Jenkins

To sum up...

1. How the push command works
2. When to use push --force
3. Why --force consider dangerous
4. Safer alternatives for push --force
5. How to recover from destructive use of push --force

Thank you!



BarkiNoaa



noaa@datree.io



noaabarki