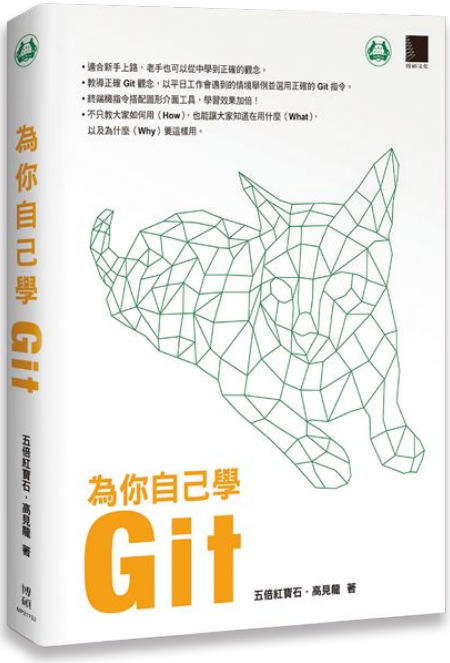
Git Cookbook

高誌佑

Reference

- https://git-scm.com/doc
- https://www.atlassian.com/git/tutorials
- 為你自己學Git



Outline

- Git Basic
- Working with Remotes
- Branch Usage
- Undoing Changes
- Debugging
- Commit Spoofing and Signing

Git Basic

First-Time Git Setup

- Set user name and email

```
> git config --global user.email "lcd010308@gmail.com"
> git config --global user.name "Chih-Yu Kao"
```

- List your config setting

```
> git config --list
user.email=lcd010308@gmail.com
user.name=Chih-Yu Kao
```

Get a Git Repository

- Initialize from a directory: **git init**

```
> cd my_project
> git init
Initialized empty Git repository
> ls -a
.  ..  .git
```

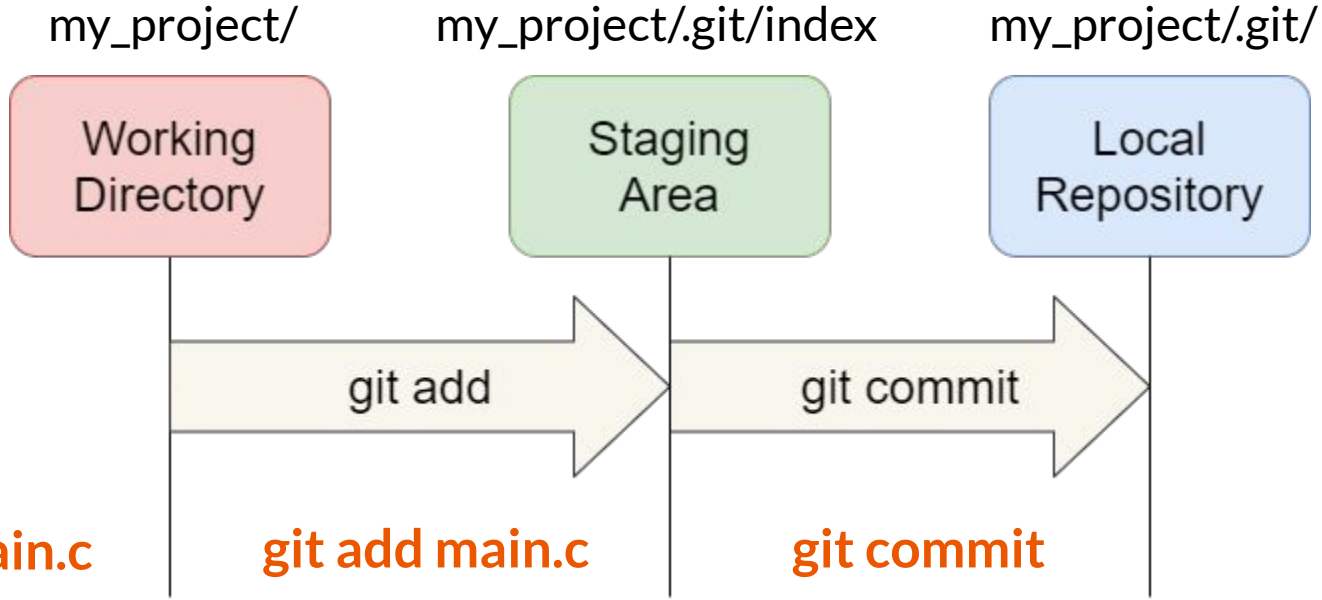
- Clone from an existing repository: **git clone**

```
> git clone https://github.com/libgit2/libgit2
```

Create a Commit

```
> touch main.c
>
> git add main.c
>
> git commit -m "First commit"
[master (root-commit) a6623aa] First commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 main.c
```

Create a Commit



e.g.

touch main.c

git add main.c

git commit

Commit History

- git log

new

old

```
commit 8618f968337dd5538928630e41ffbdf17f7b0d62 (HEAD -> master)
Author: Chih-Yu Kao <lcd010308@gmail.com>
Date:   Sun Mar 7 01:53:42 2021 +0800

    Third commit

commit d2ea4298a23b449509aedb59959b28afd1d89718
Author: Chih-Yu Kao <lcd010308@gmail.com>
Date:   Sun Mar 7 01:53:35 2021 +0800

    Second commit

commit 0ca319e23ddd26e4f315b7746683548527f0e393
Author: Chih-Yu Kao <lcd010308@gmail.com>
Date:   Sun Mar 7 01:53:28 2021 +0800

    First commit
```

Commit ID

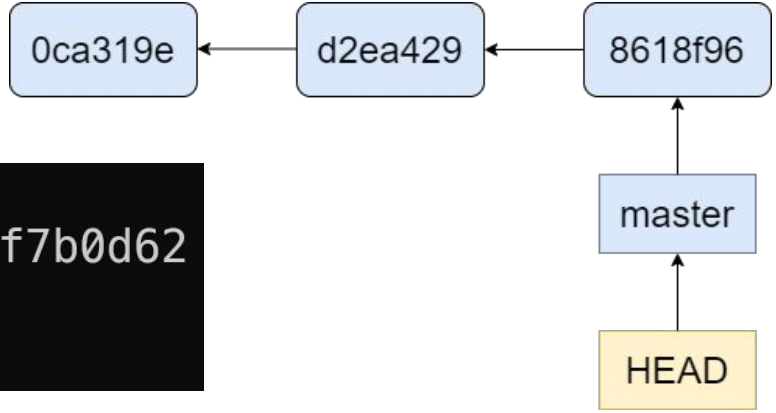
- Each commit ID is a SHA-1 hash

```
commit 8618f968337dd5538928630e41ffbdf17f7b0d62 (HEAD -> master)
Author: Chih-Yu Kao <lcd010308@gmail.com>
Date:   Sun Mar 7 01:53:42 2021 +0800

    Third commit

commit d2ea4298a23b449509aedb59959b28afd1d89718
Author: Chih-Yu Kao <lcd010308@gmail.com>
Date:   Sun Mar 7 01:53:35 2021 +0800

    Second commit

commit 0ca319e23ddd26e4f315b7746683548527f0e393
Author: Chih-Yu Kao <lcd010308@gmail.com>
Date:   Sun Mar 7 01:53:28 2021 +0800

    First commit
```

Ref

- Ref is an indirect way of referring to a commit
  - master: the master branch
  - HEAD: the currently checked-out commit / branch

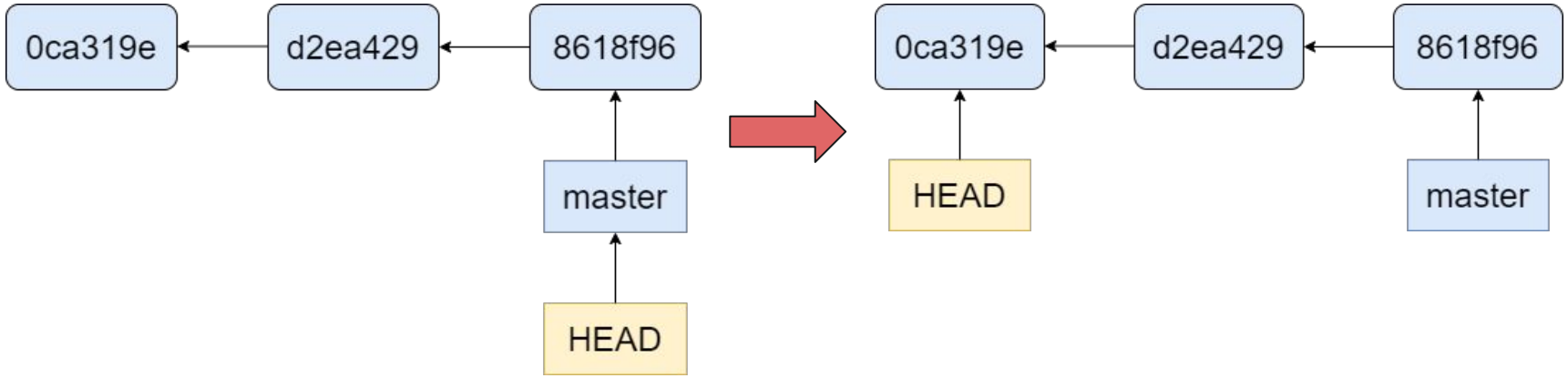
```
> cat .git/refs/heads/master
8618f968337dd5538928630e41ffbdf17f7b0d62
> cat .git/HEAD
ref: refs/heads/master
```



Switch to Other Commit

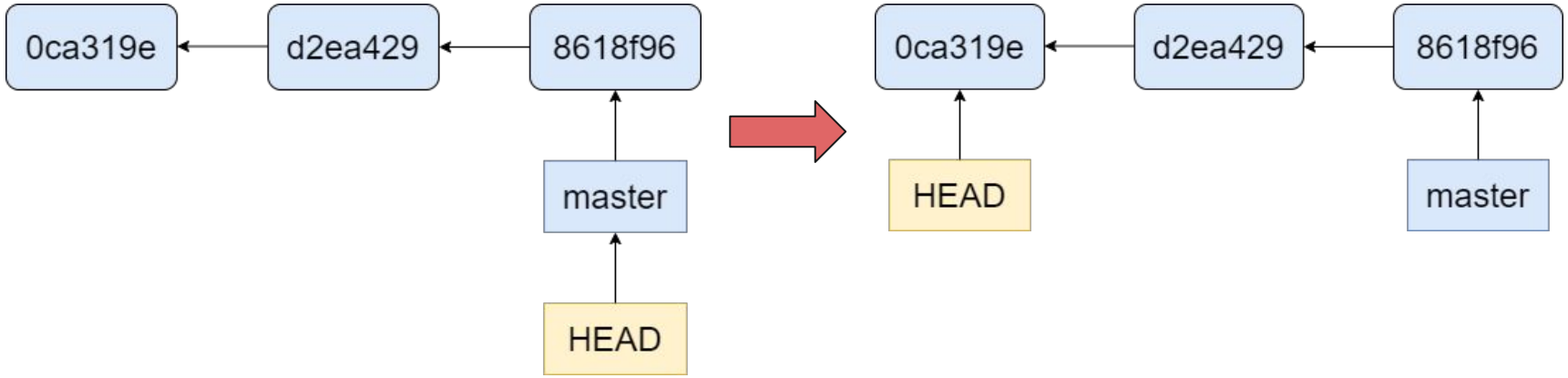
- git checkout - Switch between branches or commits

```
> git checkout 0ca319e
Note: switching to '0ca319e'.
```



Switch to Other Commit

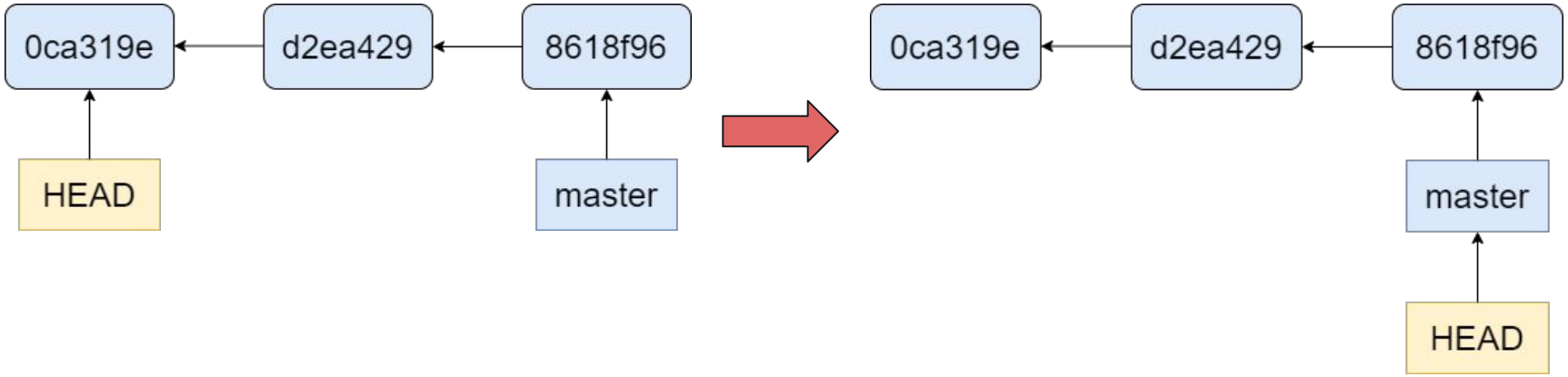
- git checkout - Switch between branches or commits
  - git checkout 0ca319e
  - git checkout HEAD^^
  - git checkout HEAD~2



Switch to Other Commit

- git checkout - Switch between branches or commits

```
> git checkout master
Previous HEAD position was 0ca319e First commit
Switched to branch 'master'
```



File Ignoring

Ignore Files

- Create a file named **.gitignore**

```
# secret file
secret.yml

# C/C++
*.o
*.obj
*.exe

# Python
__pycache__/
*.egg-info/

# VSCode
.vscode/

# Mac OS X
.DS_Store
__MACOSX
```

Ignore Files

- Before ignoring

```
> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .vscode/
```

- After ignoring

```
> git status
On branch master
nothing to commit, working tree clean
```

Working with Remotes

Add Remote Repository

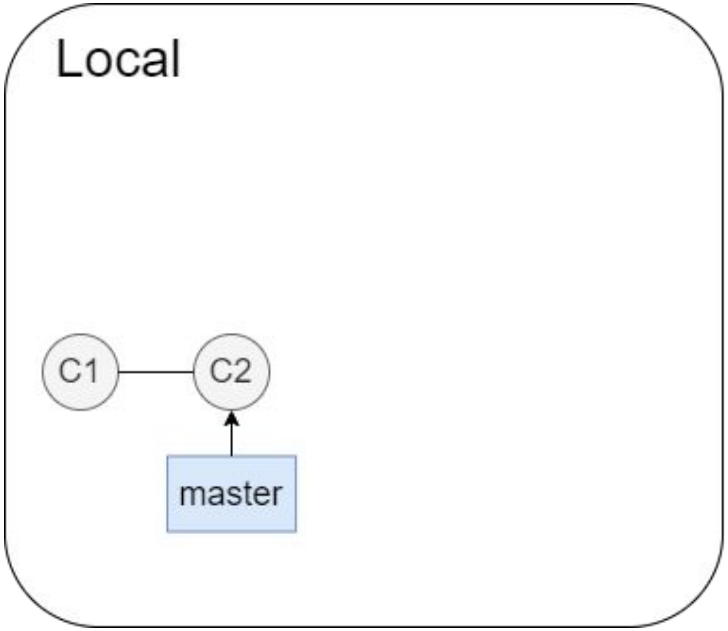
- If you **git clone** an existing repository, the **origin** remote is automatically set

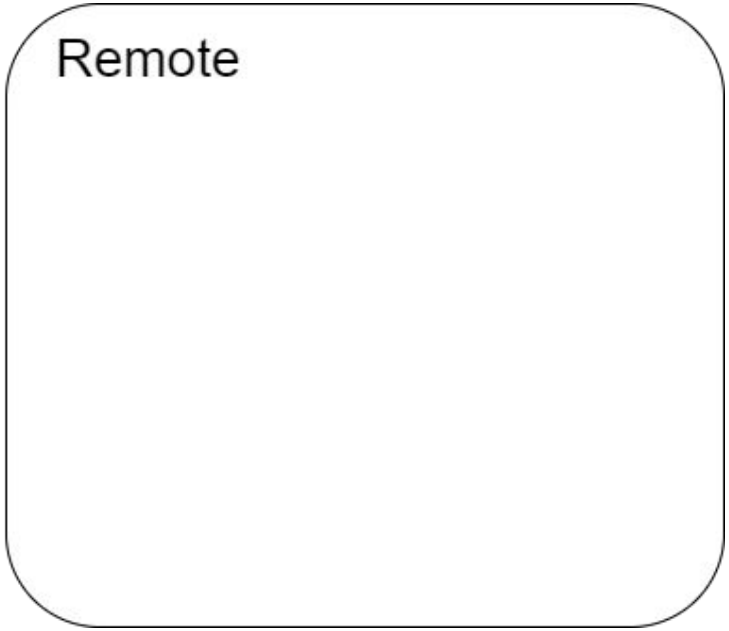
name url

```
> git remote add origin https://github.com/kaocy/git-test.git
> git remote -v
origin  https://github.com/kaocy/git-test.git (fetch)
origin  https://github.com/kaocy/git-test.git (push)
```

Synchronize to Remote

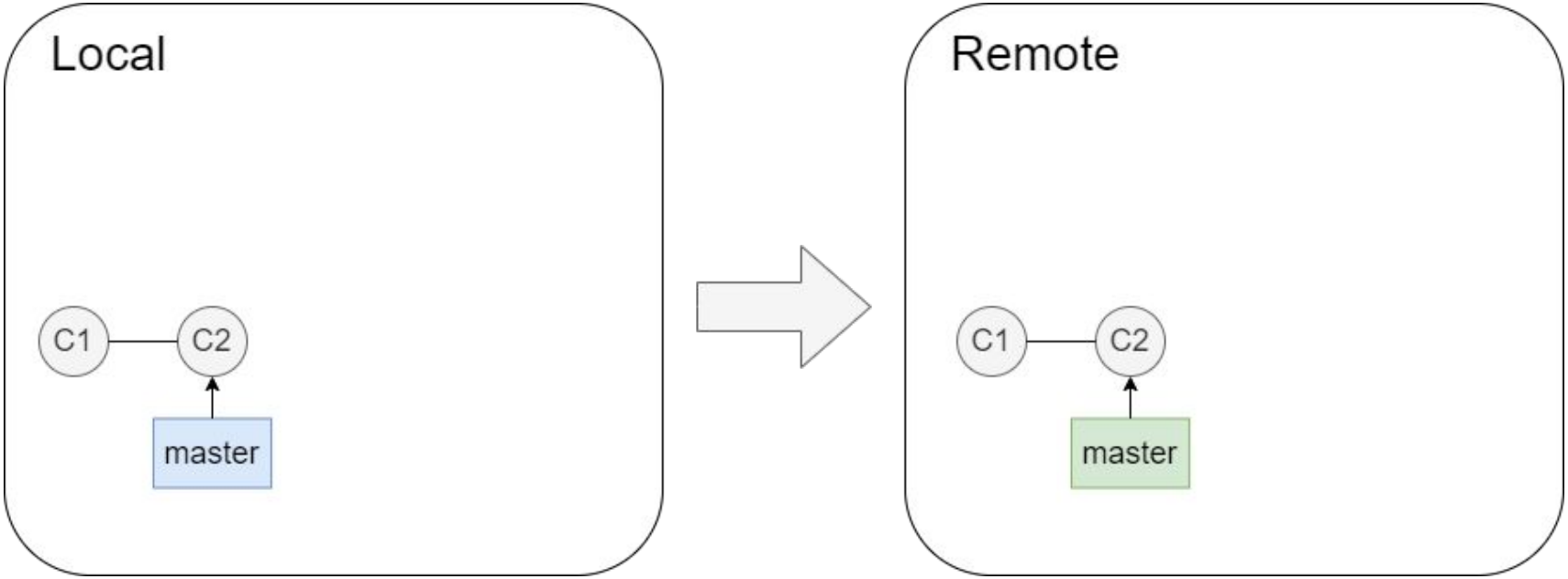
- There are two commits in local repo





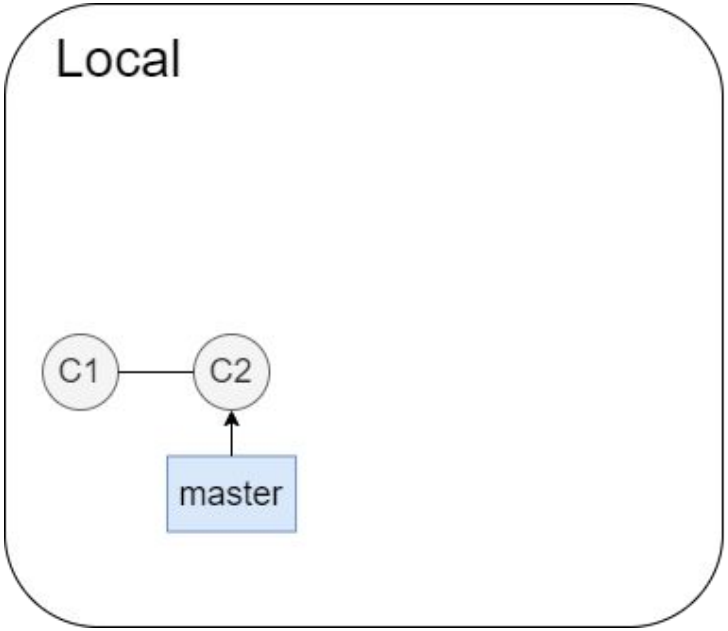
Synchronize to Remote

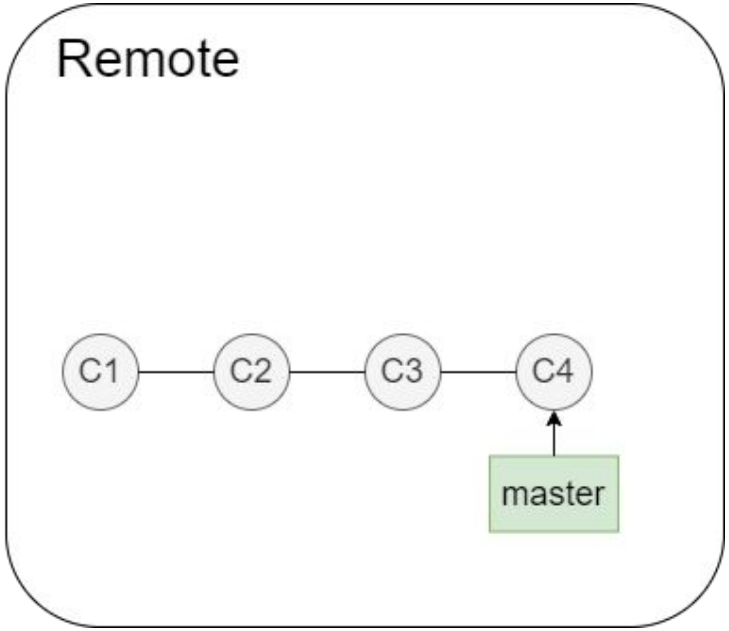
- git push origin master



Synchronize to Remote

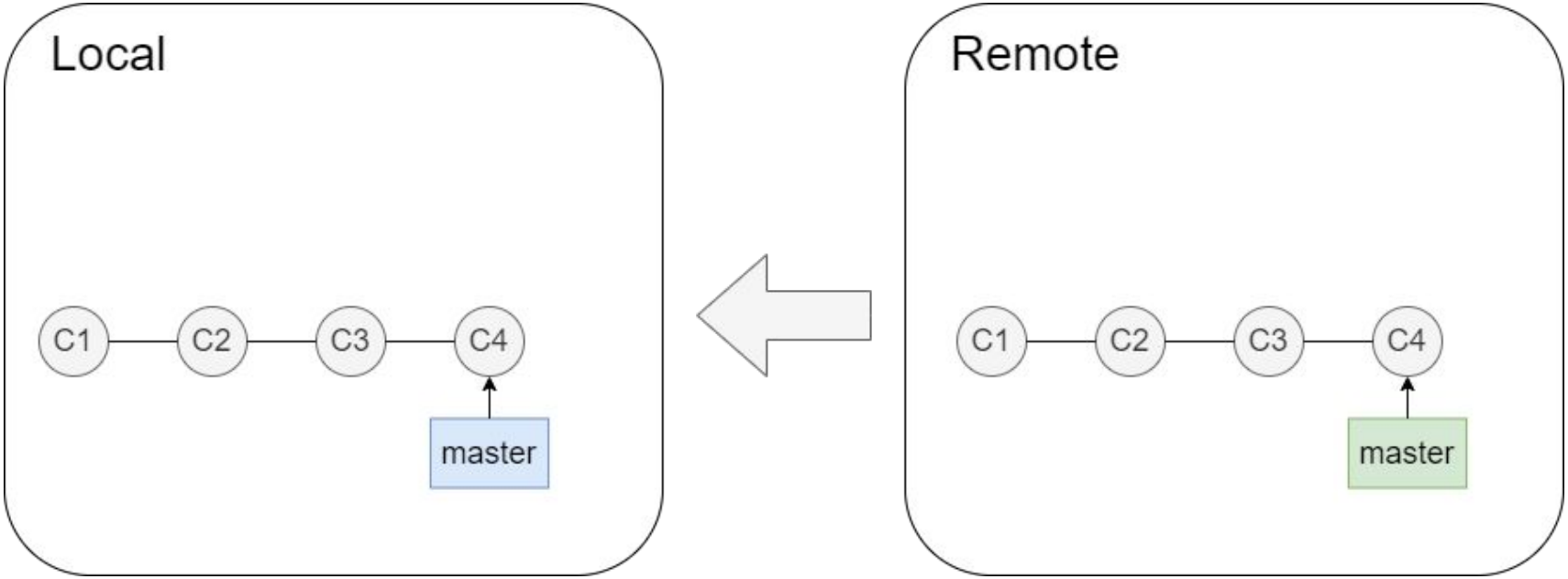
- There are two new commits in remote repo





Synchronize to Remote

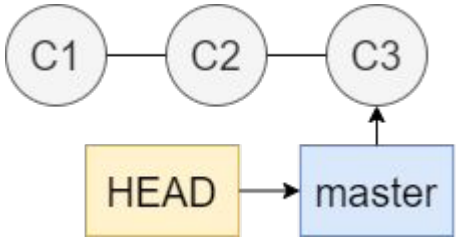
- git pull origin master



Branch Usage

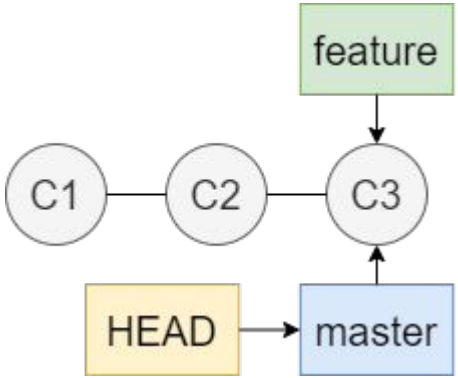
Branch

- Suppose there are 3 commits originally



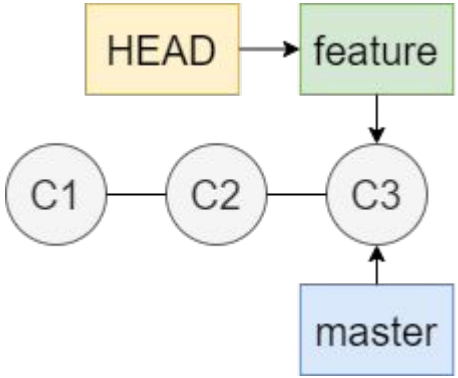
Branch - Develop Features

- git branch feature



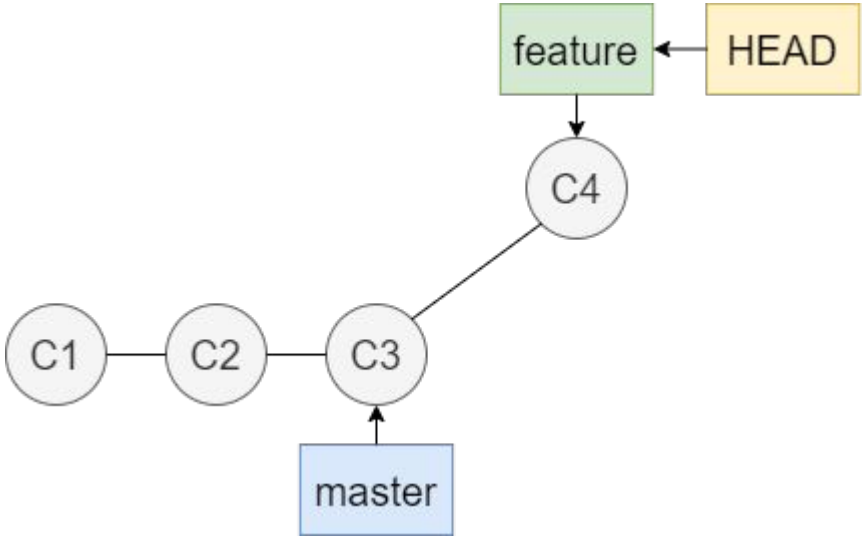
Branch - Develop Features

- git checkout feature



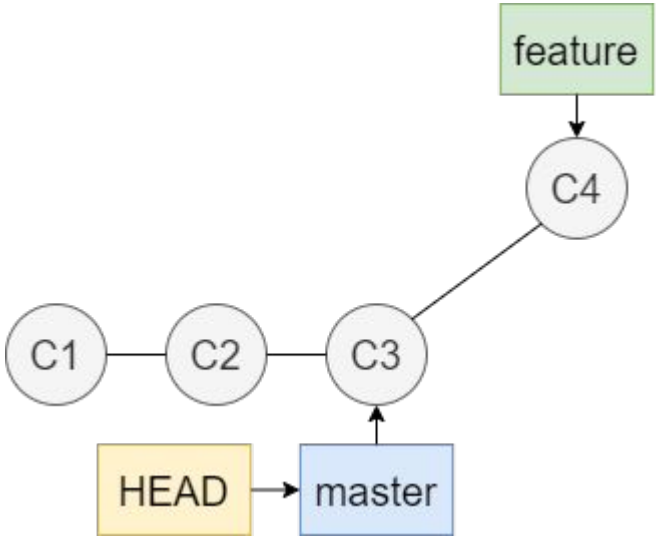
Branch - Develop Features

- git commit



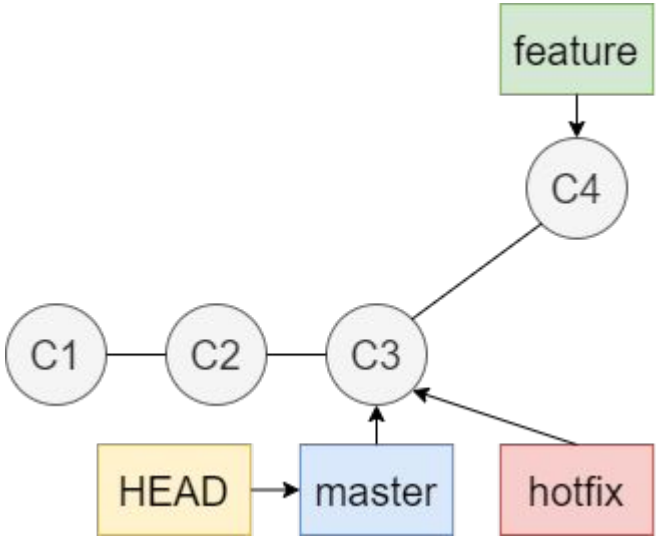
Branch - Fix Bugs

- git checkout master



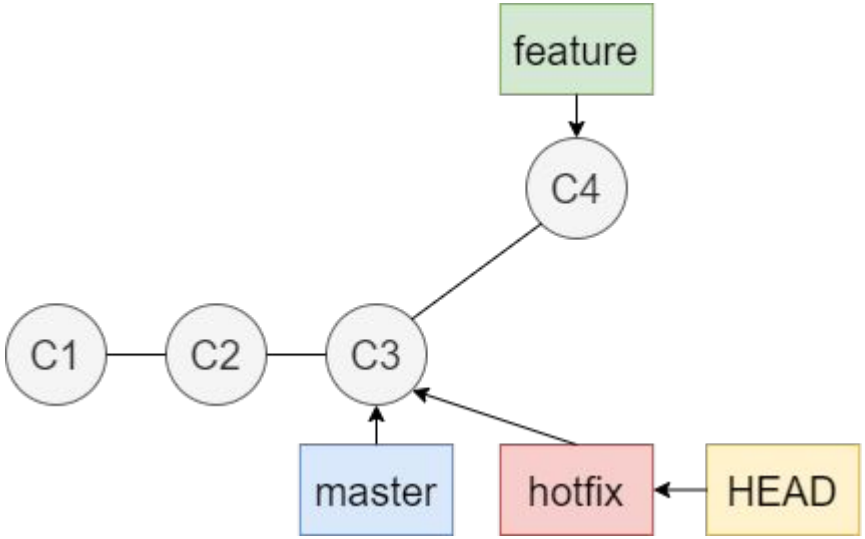
Branch - Fix Bugs

- git branch hotfix



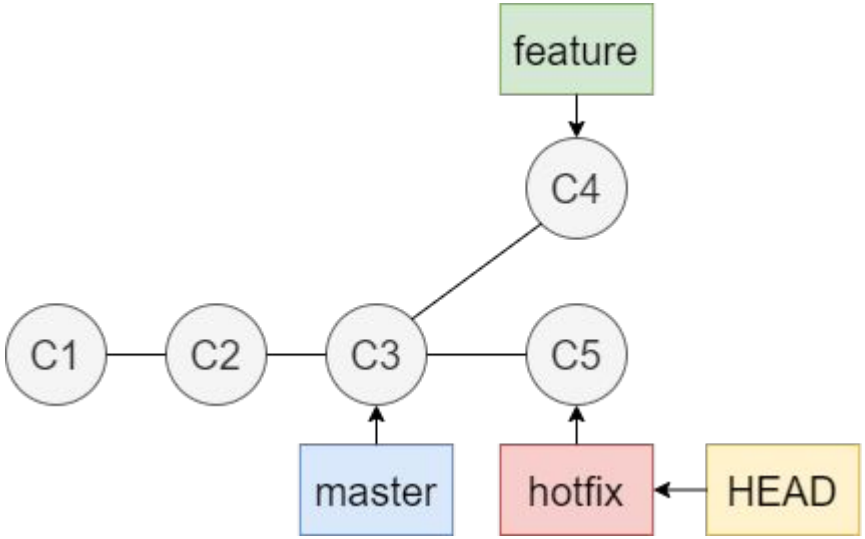
Branch - Fix Bugs

- git checkout hotfix



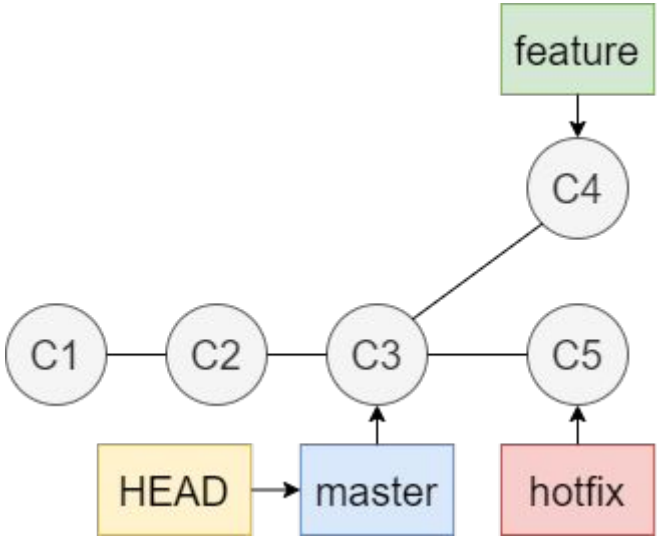
Branch - Fix Bugs

- git commit



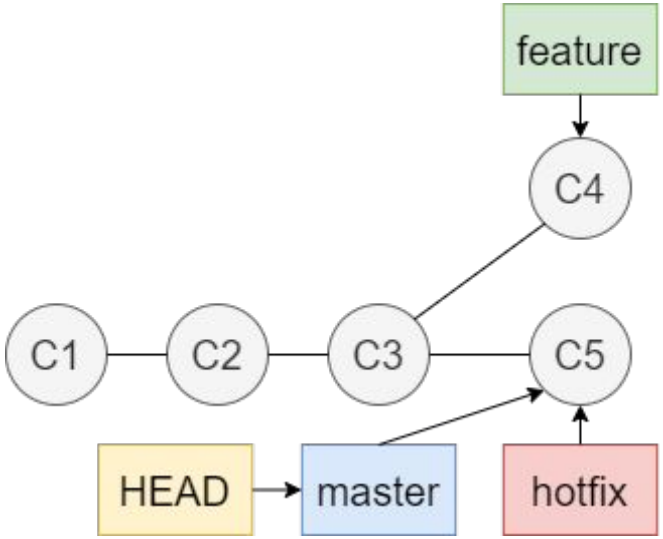
Branch - Merge to master

- git checkout master



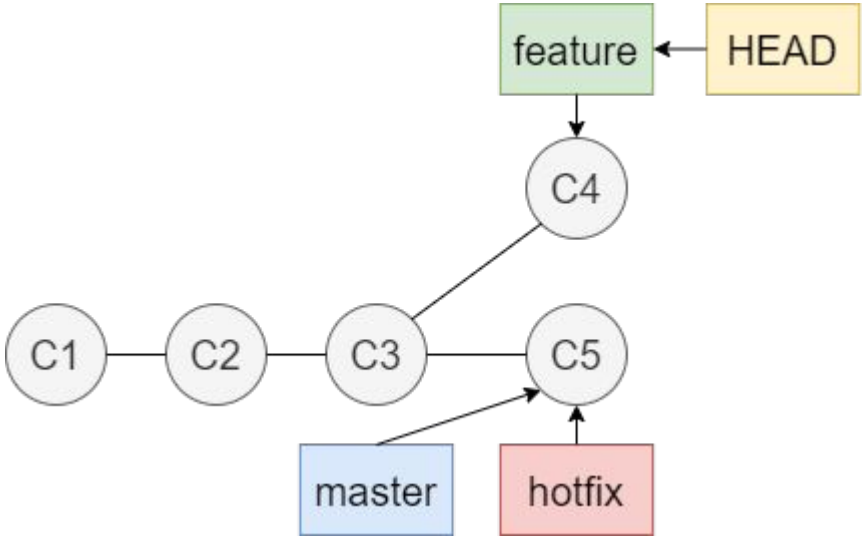
Branch - Merge to master

- git merge hotfix



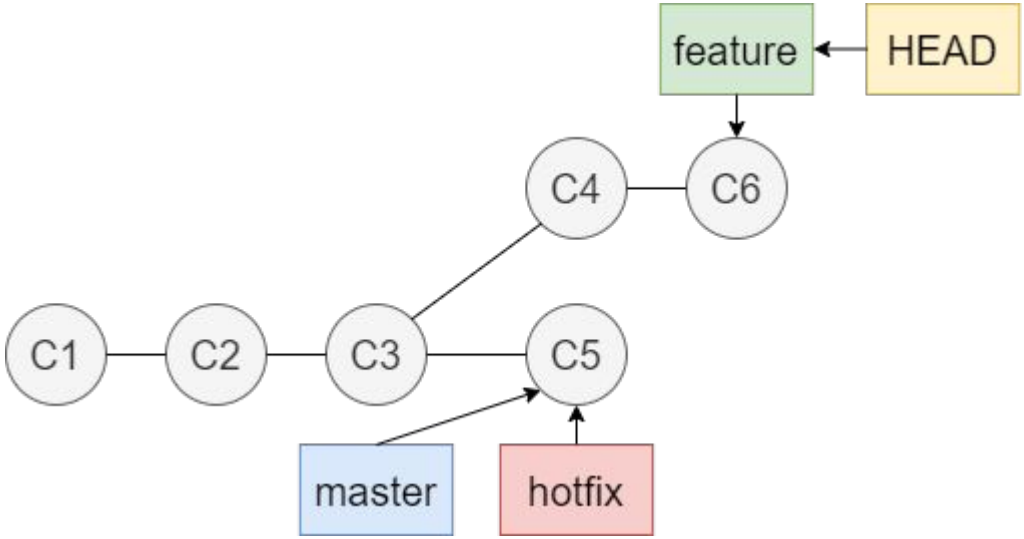
Branch - Develop Features

- git checkout feature



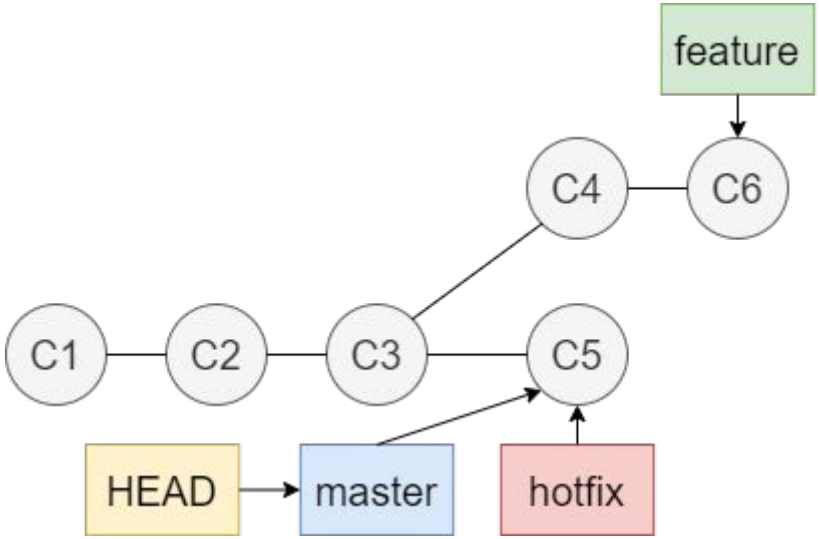
Branch - Develop Features

- **git commit**



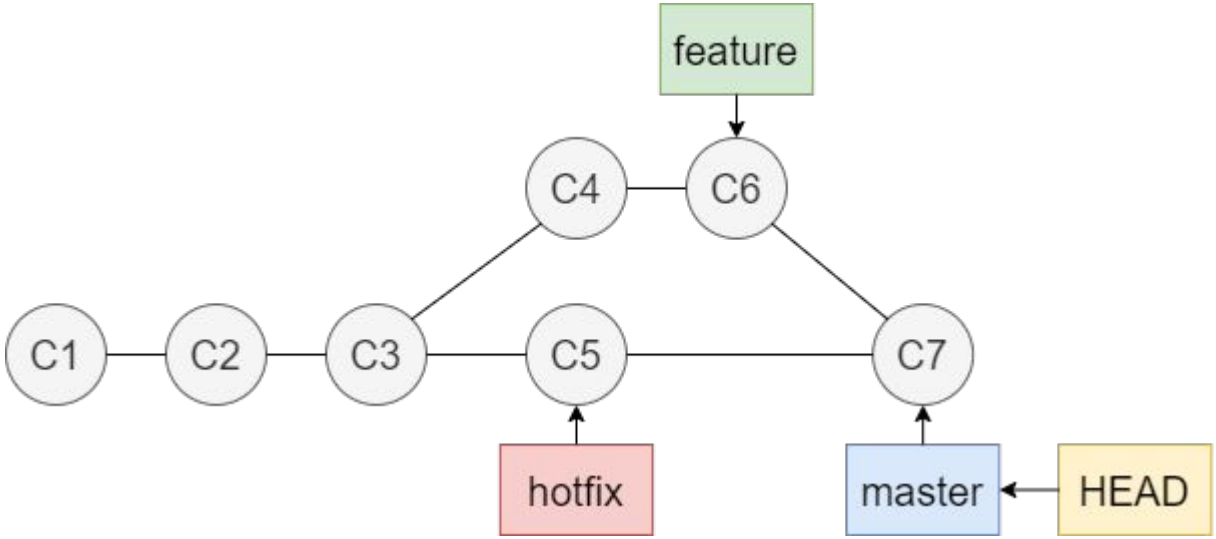
Branch - Merge to master

- git checkout master



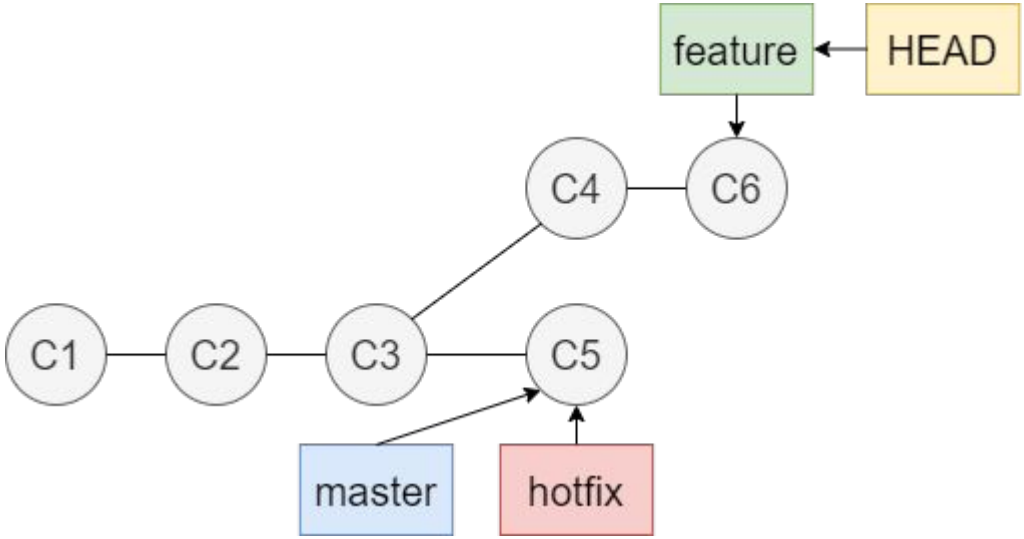
Branch - Merge to master

- git merge feature



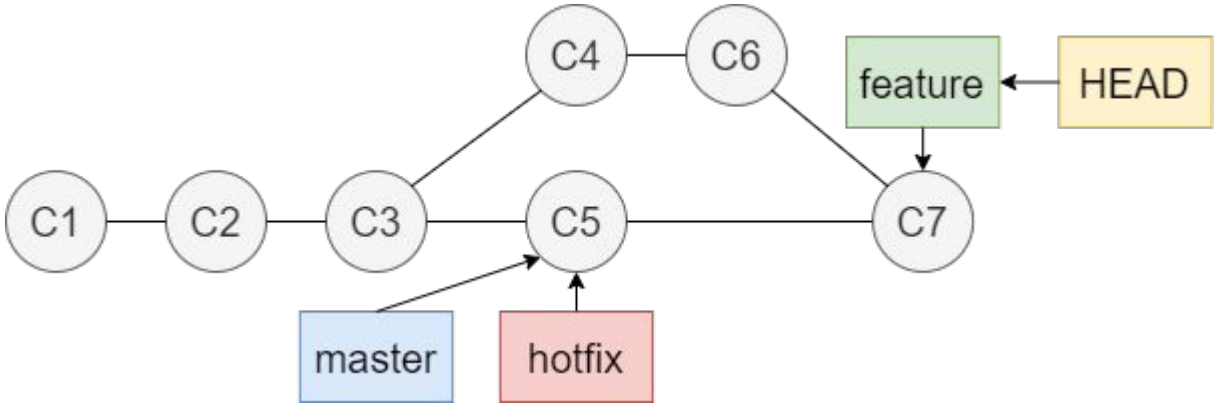
Branch - Merge to feature

- git checkout feature



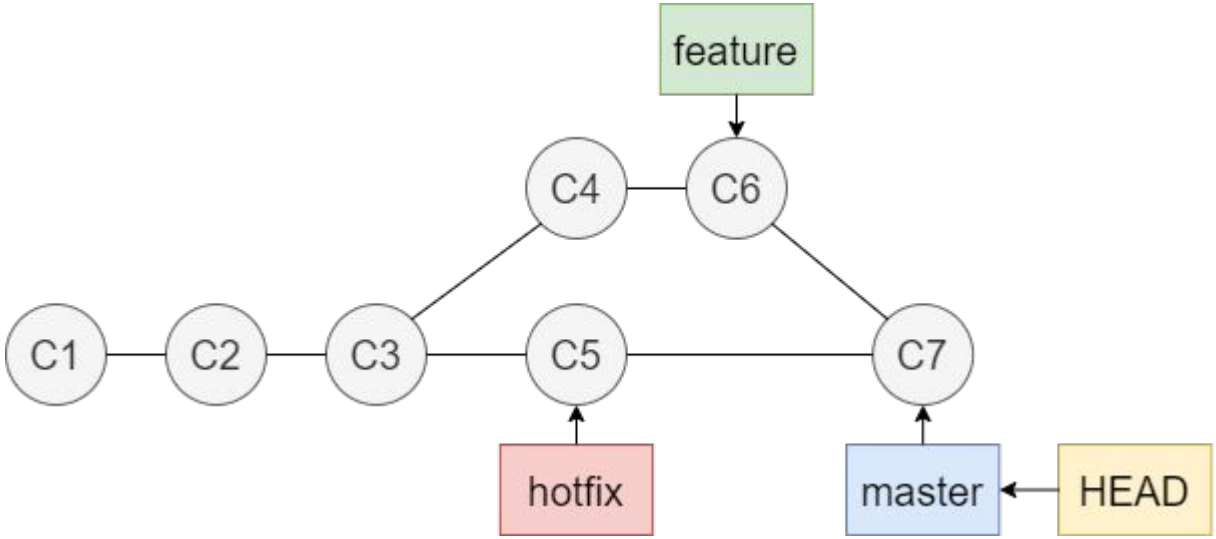
Branch - Merge to feature

- git merge master



Branch - Merge Conflicts

- If you change the same part of the same file differently in two branches, you will get a merge conflict



Branch - Merge Conflicts

- There is a conflict in index.html

```
> git merge feature
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Branch - Merge Conflicts

- There is a conflict in index.html

```
> git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   index.html
```

Branch - Merge Conflicts

- There is a conflict in index.html

```
<<<<<<< HEAD (Current Change)
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
    please contact us at support@github.com
</div>
>>>>>>> feature (Incoming Change)
```

Branch - Merge Conflicts

- Resolve the conflicts manually
- Suppose we want to keep the part of **feature branch**

```
<div id="footer">
    please contact us at support@github.com
</div>
```

Branch - Merge Conflicts

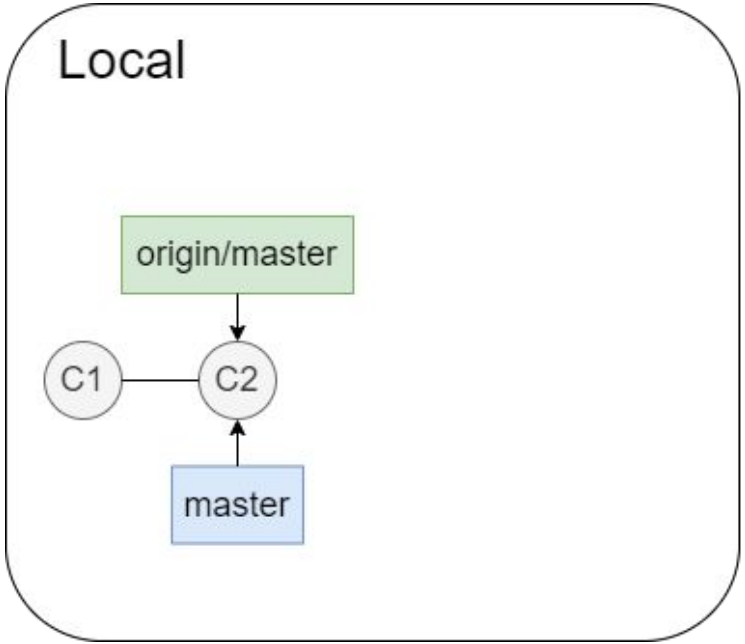
- Use **git add** & **git commit** after resolving all conflicts

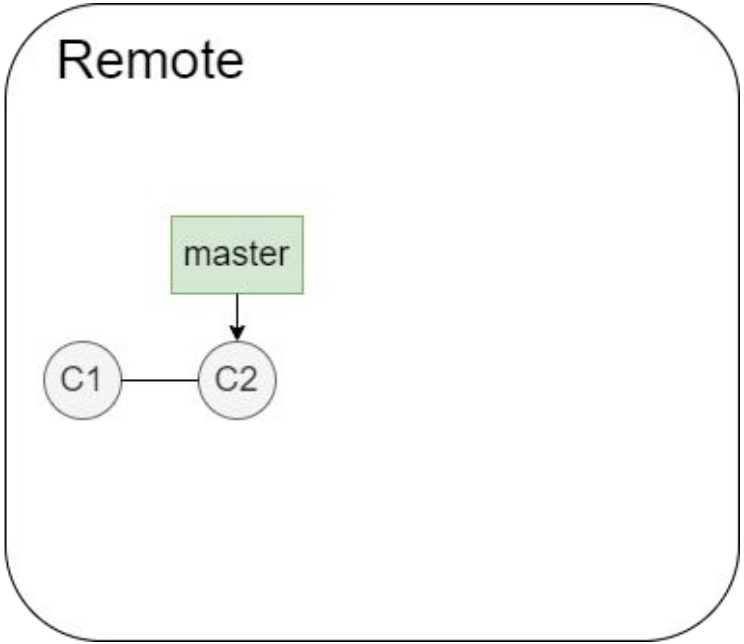
```
> git add index.html
> git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
        modified:   index.html

> git commit -m "Merge master and feature branch"
```

Remote Branch

Remote Branch

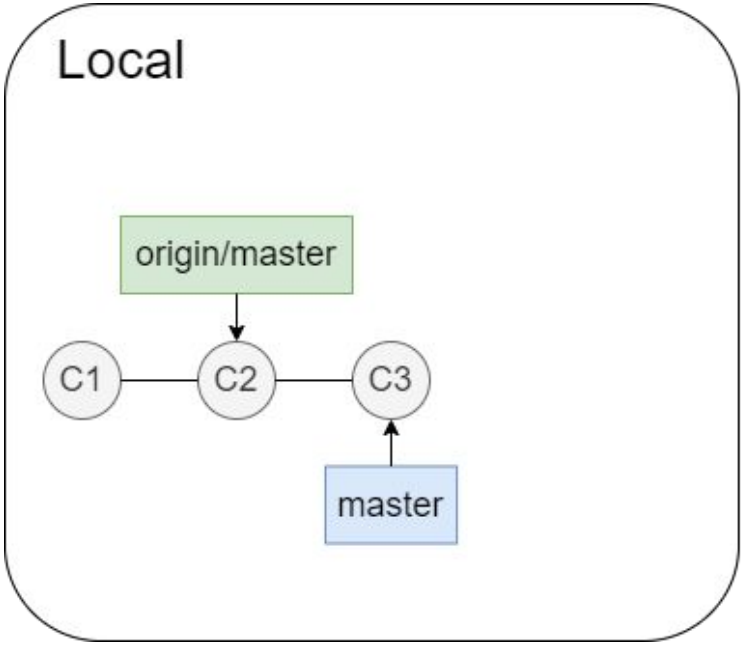
- References to the state of remote branches

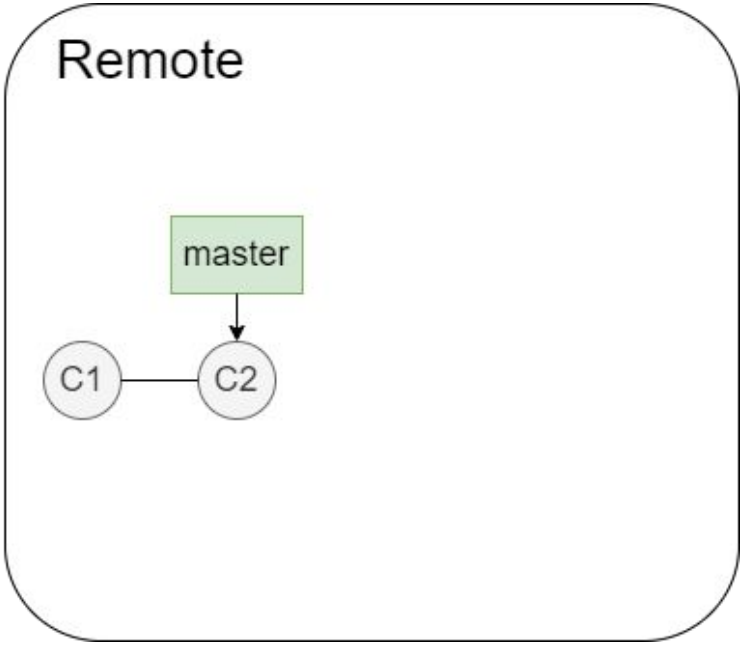




Push to Remotes

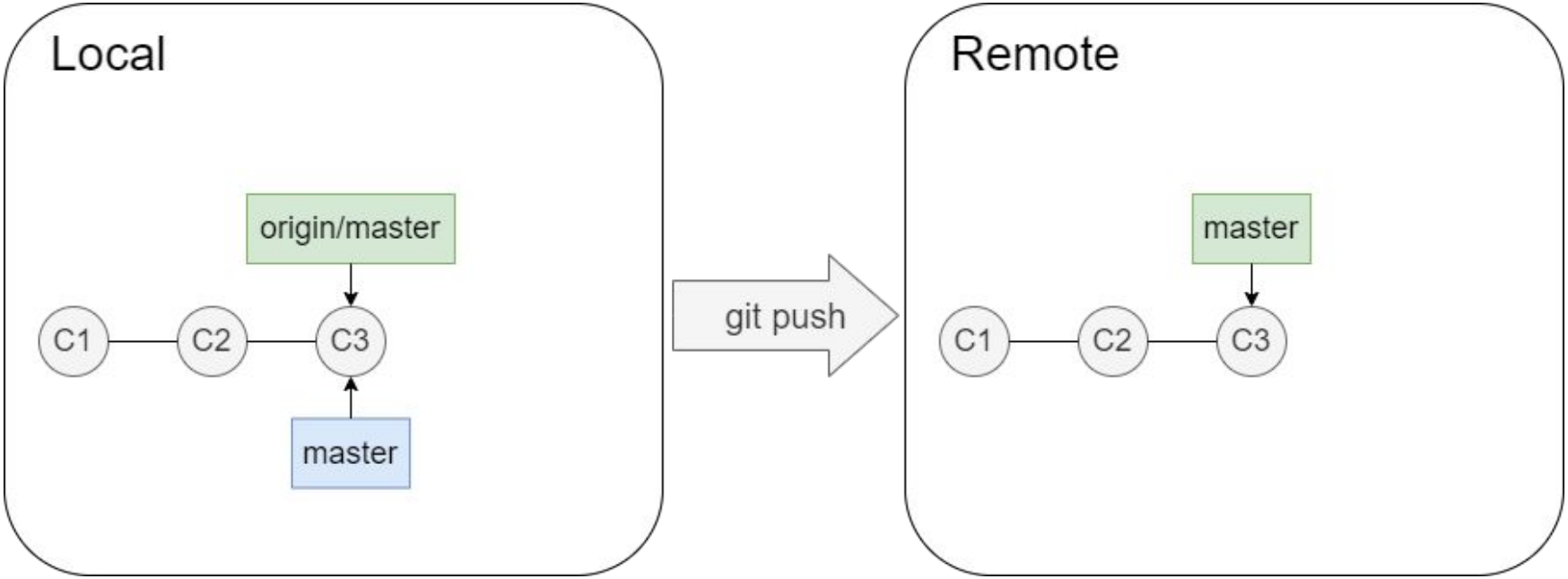
- Create a commit in local repo





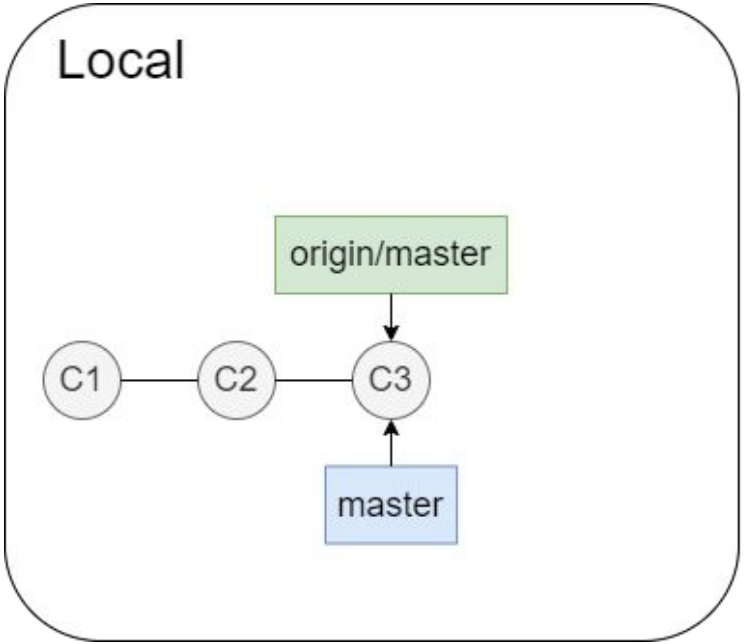
Push to Remotes

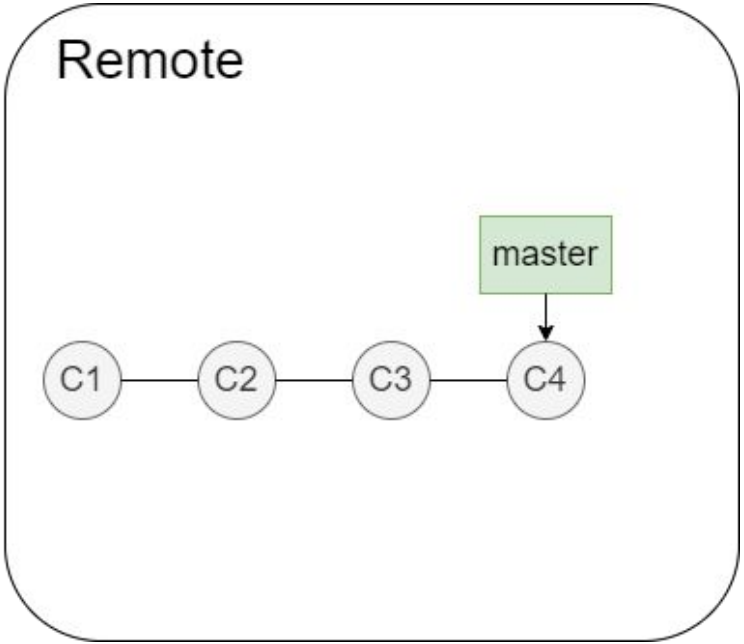
- **git push**



Pull from Remotes

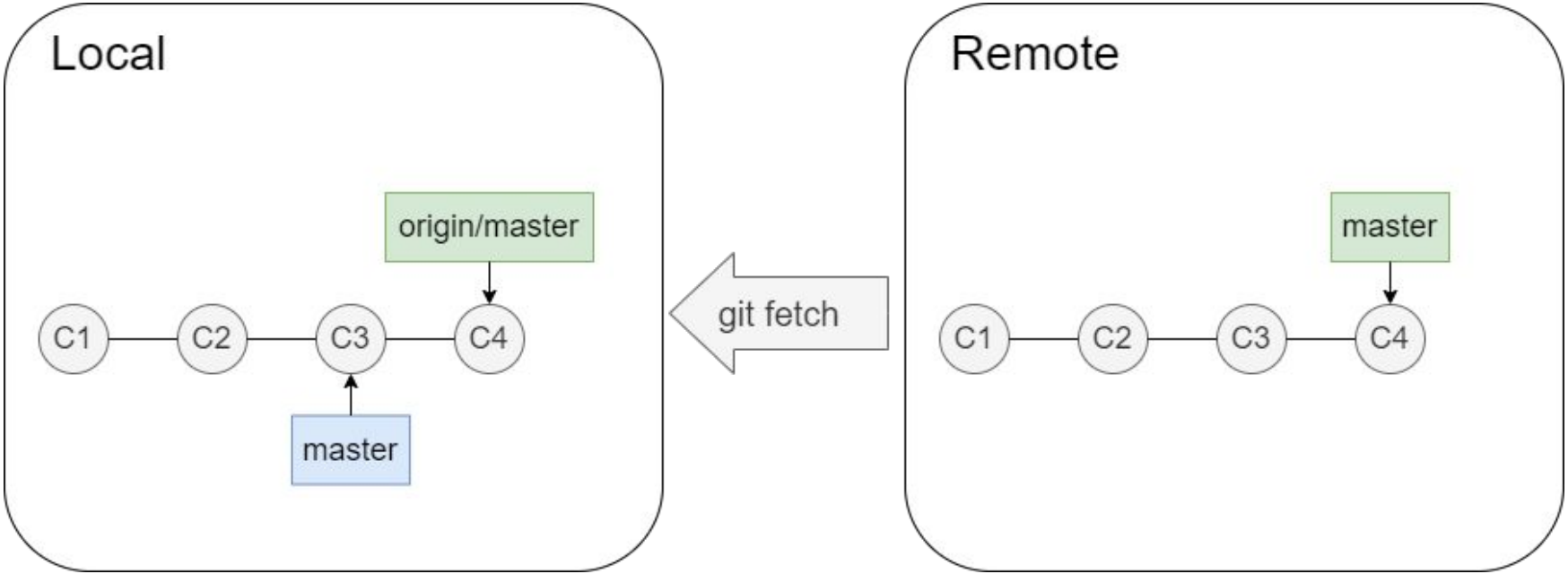
- There is a new commit in remote repo





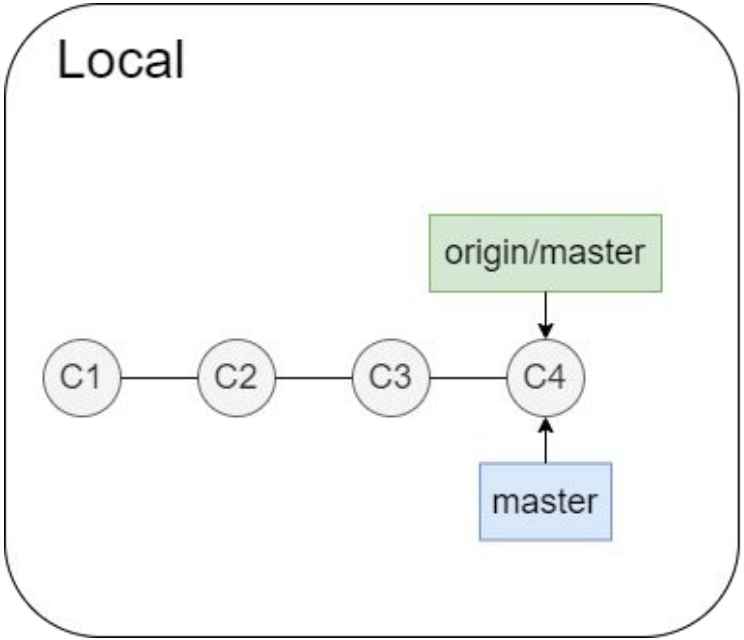
Pull from Remotes

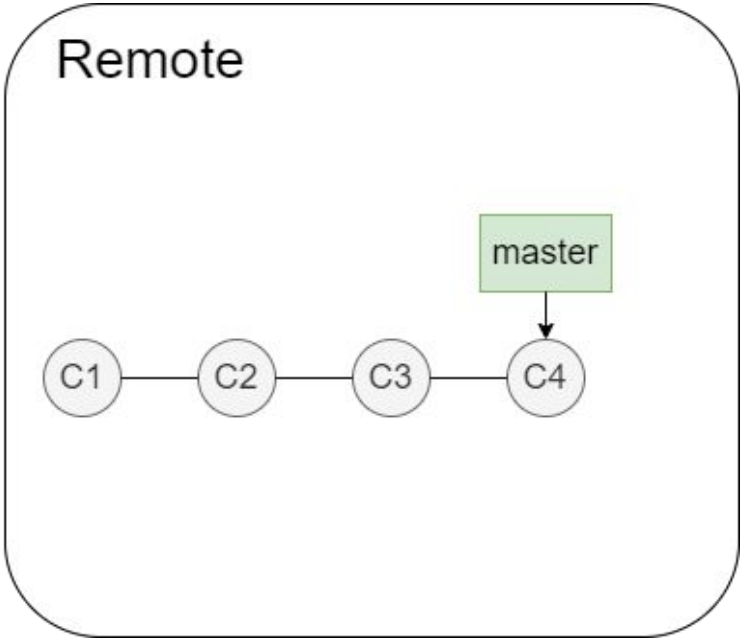
- git pull = **git fetch** + git merge



Pull from Remotes

- git pull = git fetch + **git merge**





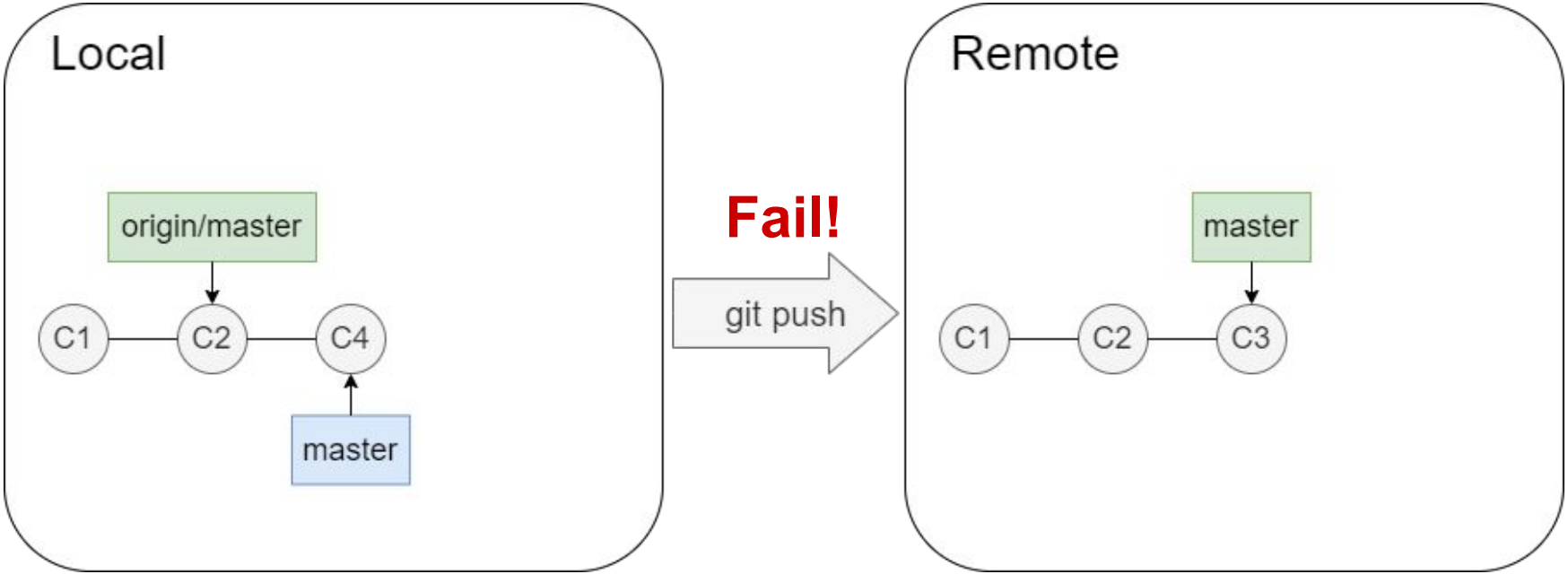
Why Fail to Push?

Why Fail to Push?

```
> git push origin master
To https://github.com/kaocy/git-test.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/kaocy/git-test.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

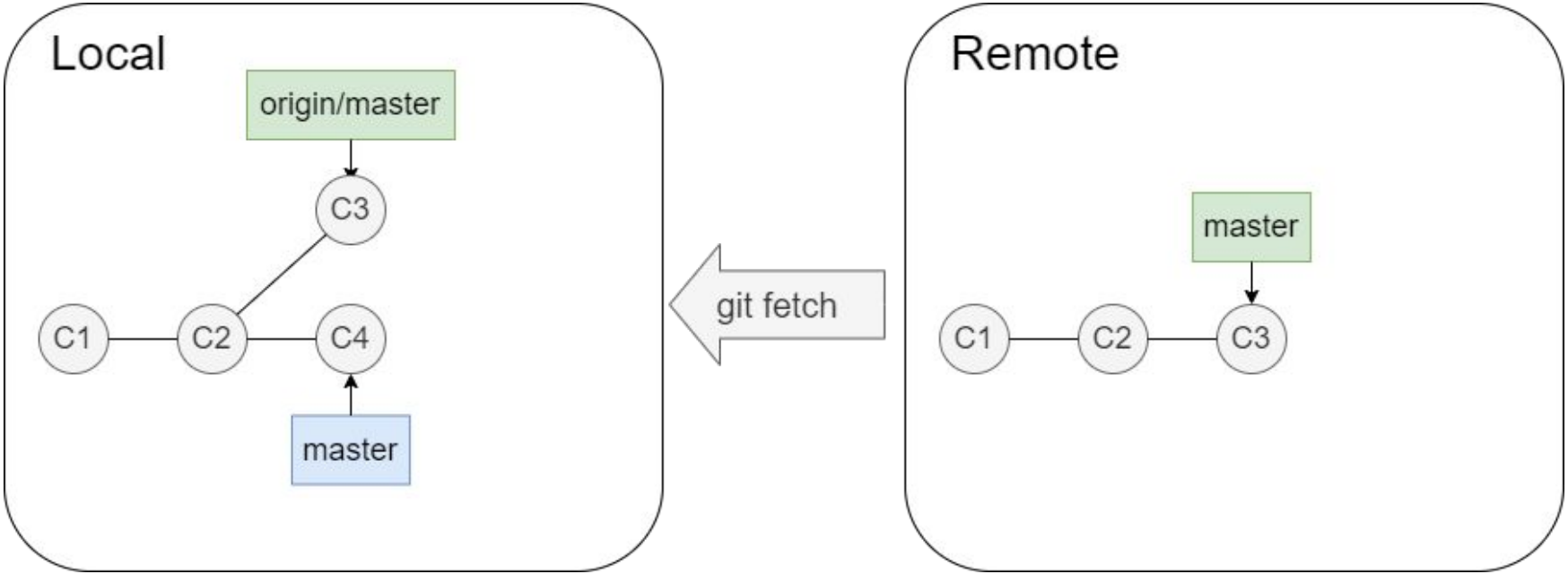
Why Fail to Push?

- Someone pushed a new commit before you did



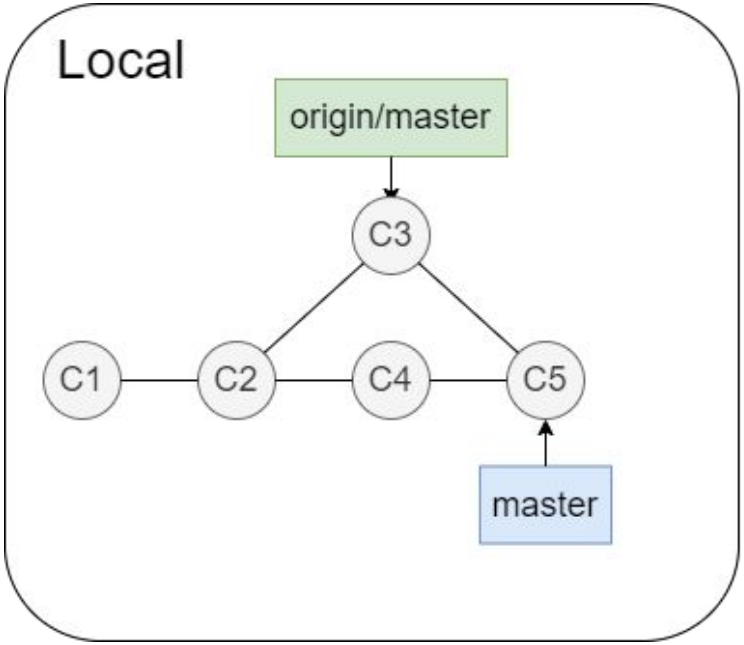
Solution: Pull before Push

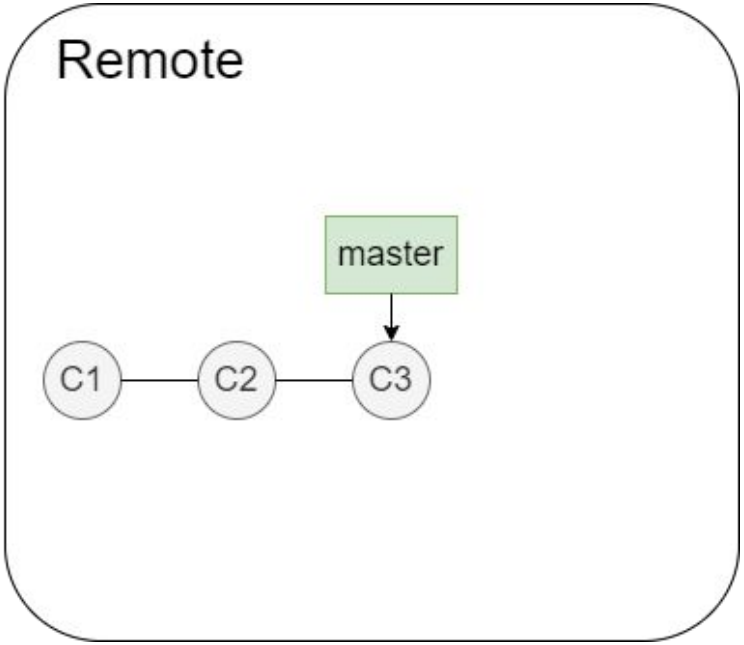
- git pull = **git fetch** + git merge



Solution: Pull before Push

- git pull = git fetch + **git merge**





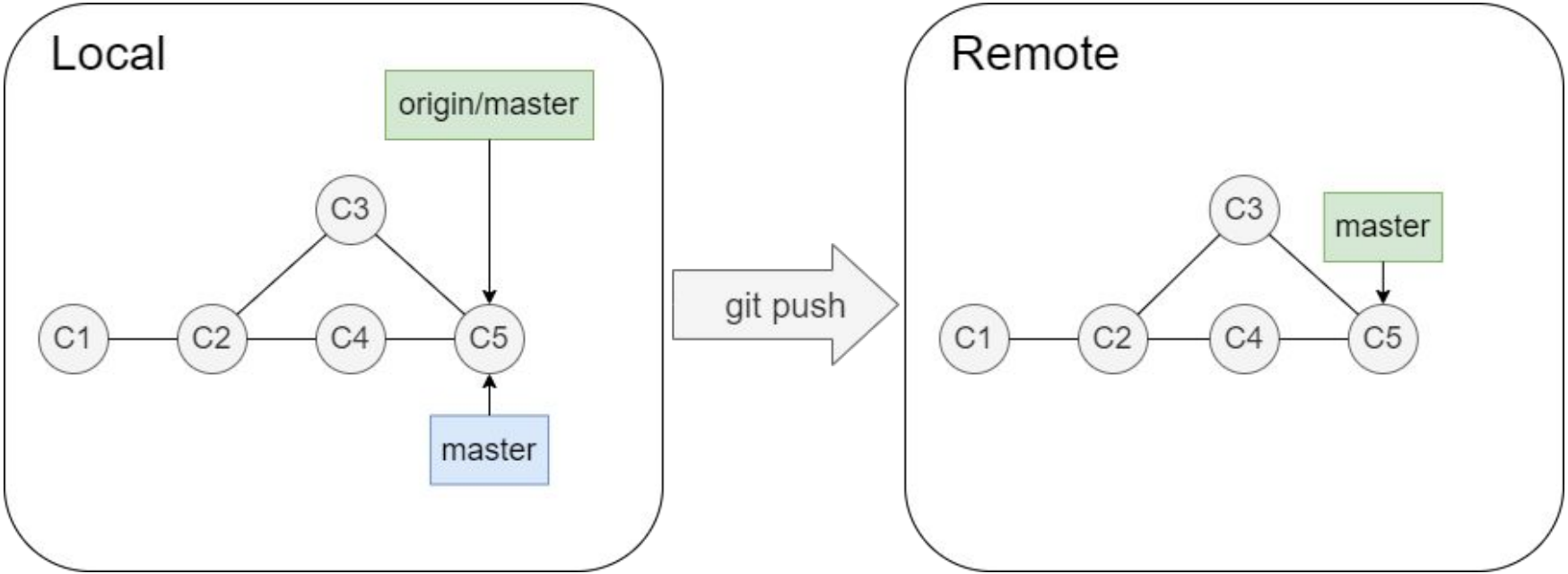
Solution: Pull before Push

- **git pull = git fetch + git merge**
- That is why a merge commit pops up when you run **git pull**

```
Merge branch 'master' of github.com:kaocy/git-test
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

Solution: Pull before Push

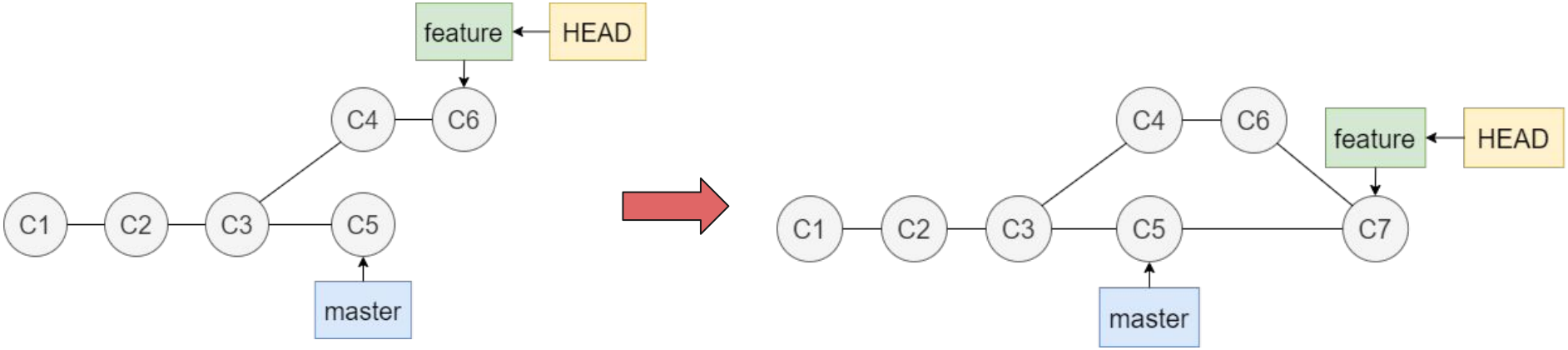
- **git push**



Merge vs Rebase

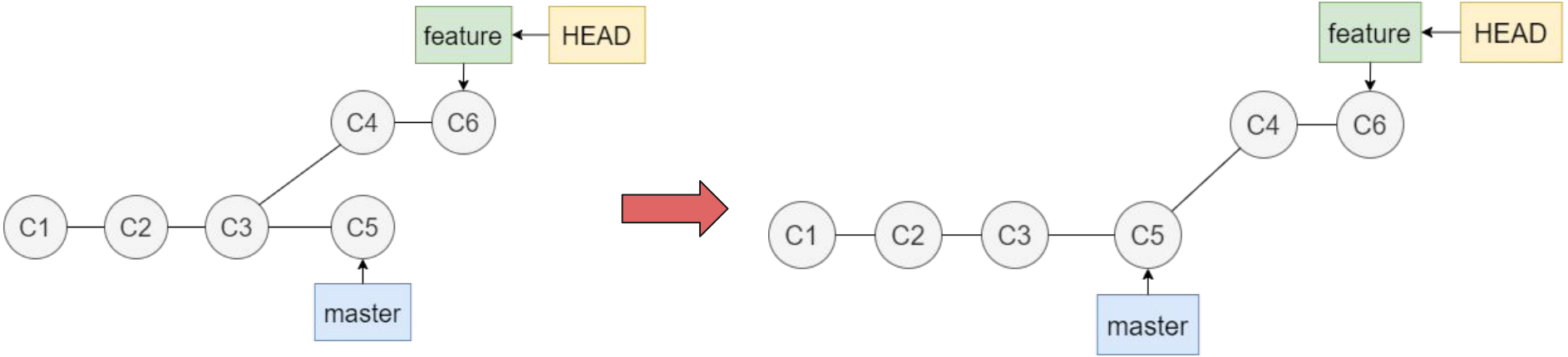
Merge vs Rebase

- git merge master



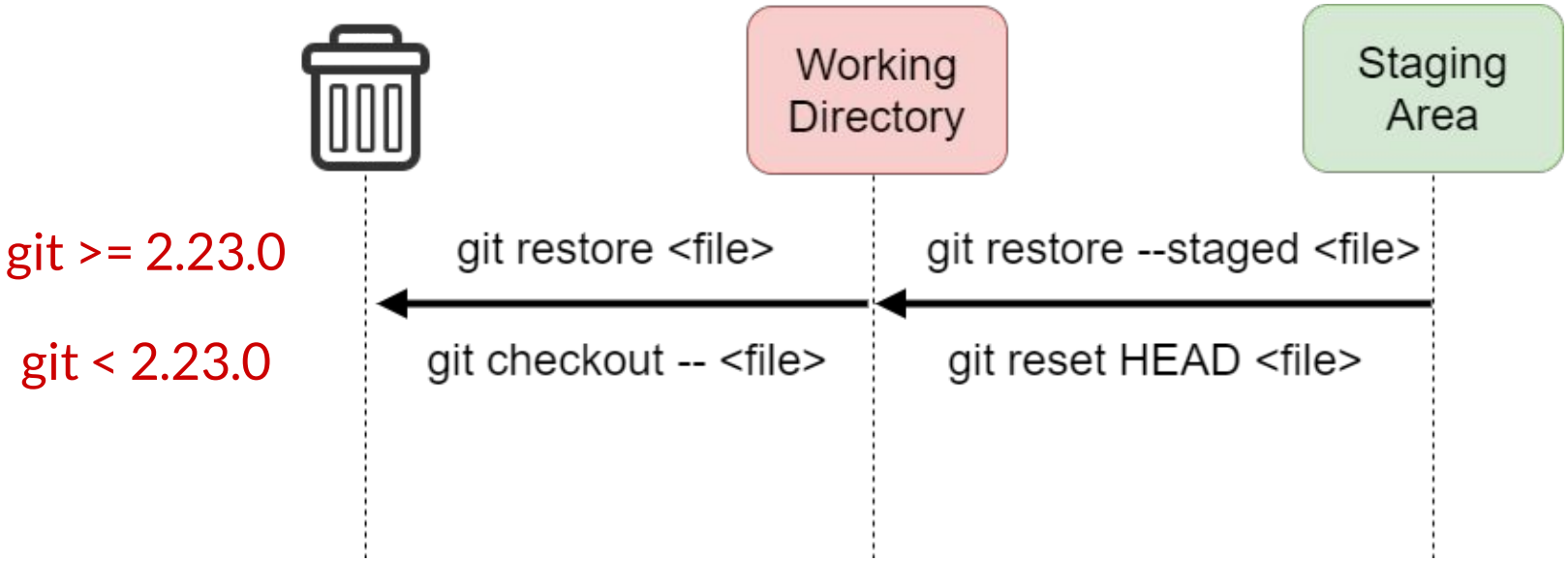
Merge vs Rebase

- git rebase master



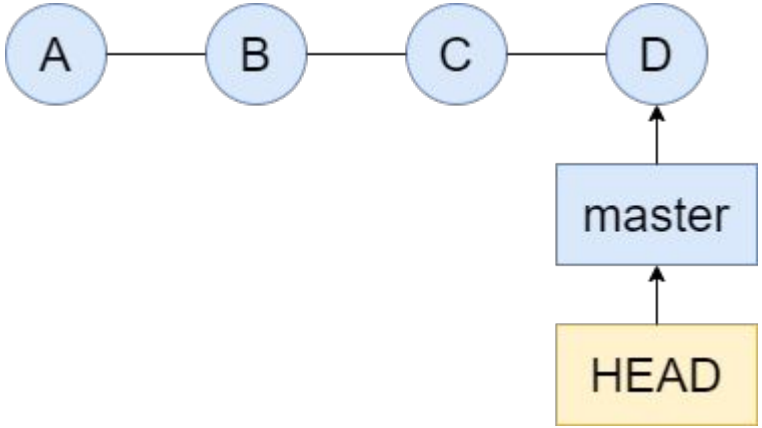
Undoing Changes

Undo Files


Working Directory
Staging Area
git >= 2.23.0
git restore <file>
git restore --staged <file>
git < 2.23.0
git checkout -- <file>
git reset HEAD <file>

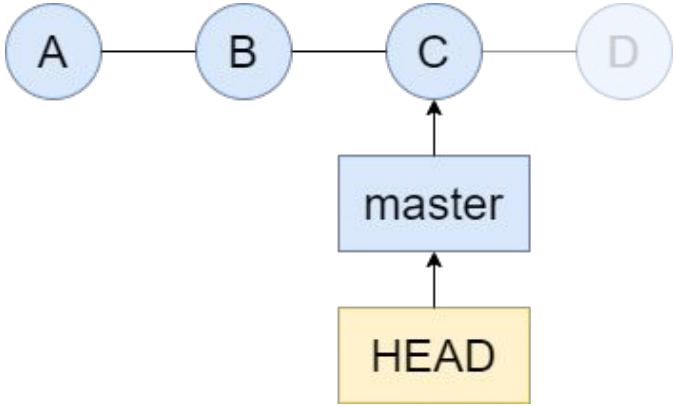
Undo Commits

- Suppose we want to undo commit D

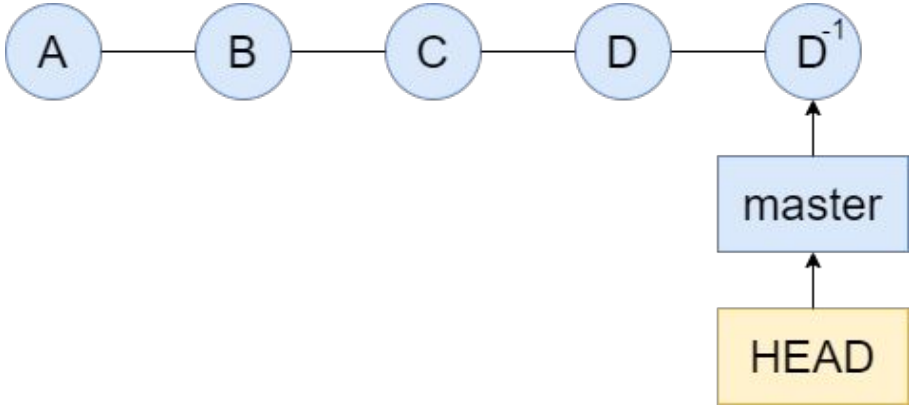


Undo Commits

- Two methods to undo commit D



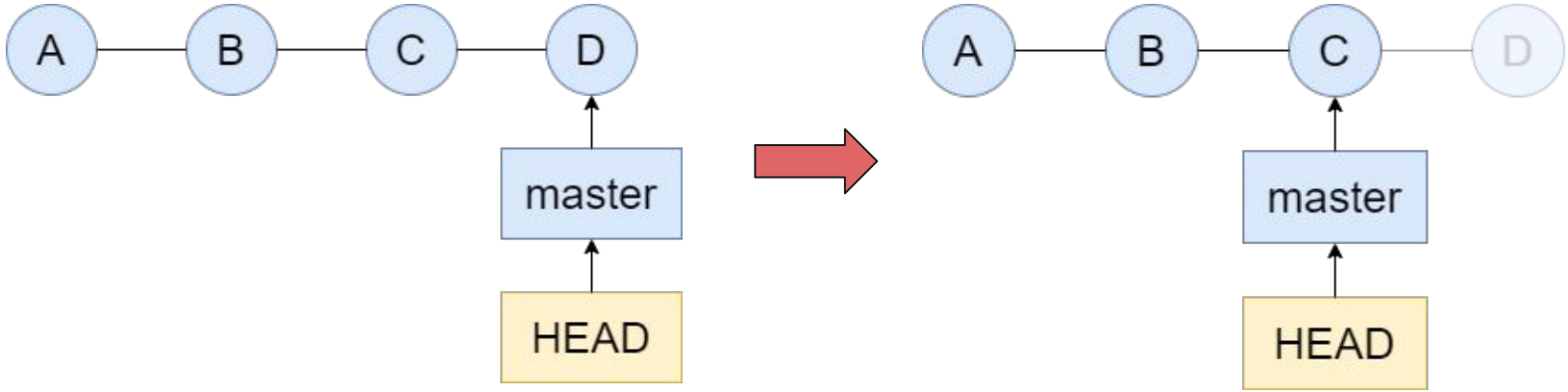
git reset



git revert

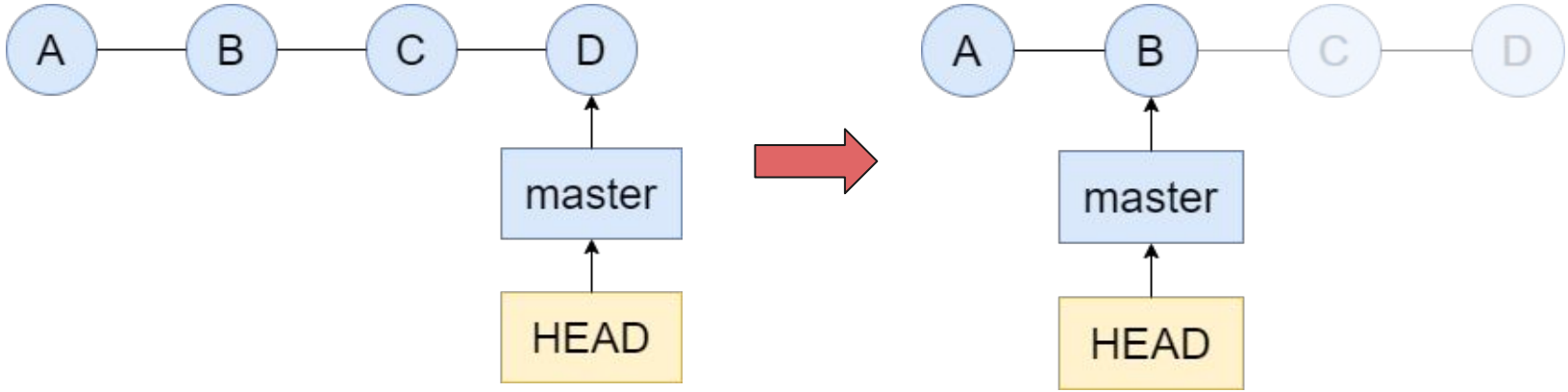
Undo Commits - git reset

- Reset current HEAD to the specified state
- Only for **local commit**
- Command: **git reset HEAD~1**

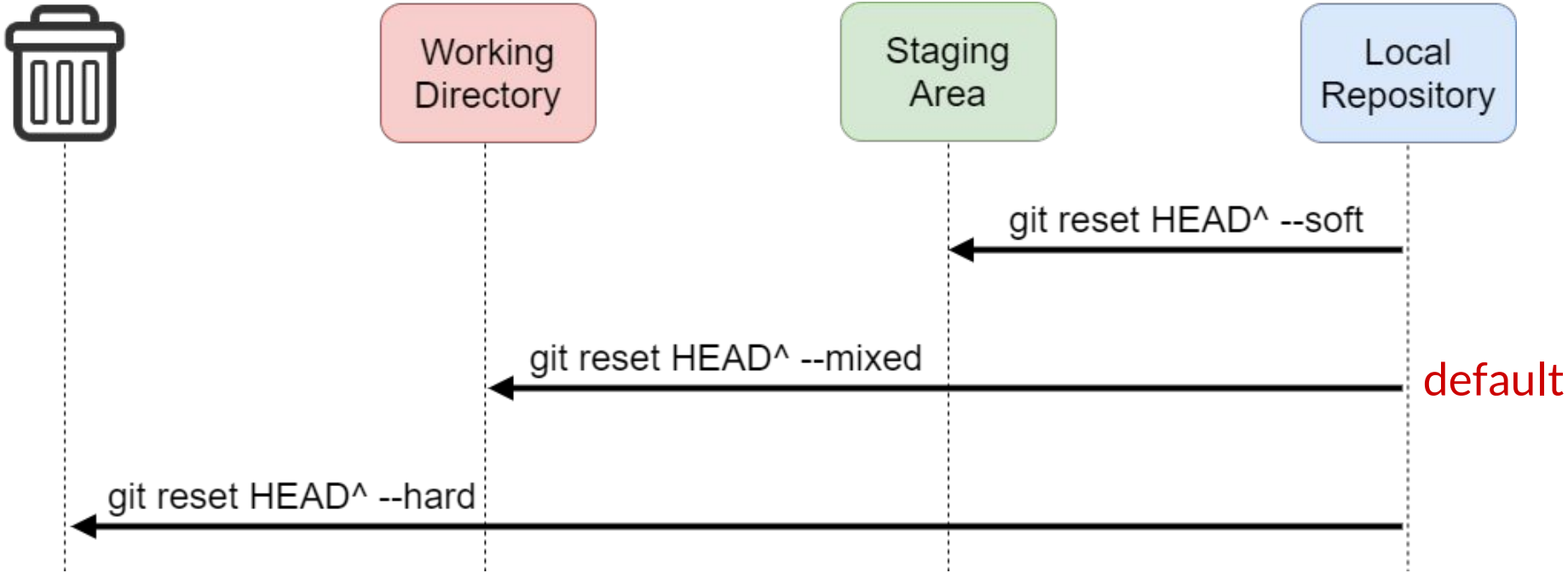


Undo Commits - git reset

- Reset current HEAD to the specified state
- Only for **local commit**
- Command: **git reset HEAD~2**

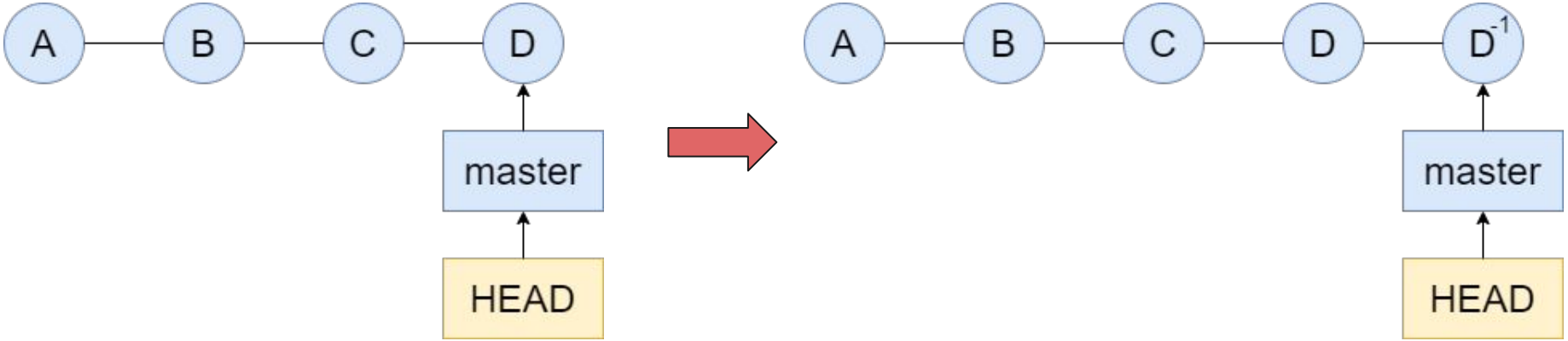


Undo Commits - git reset



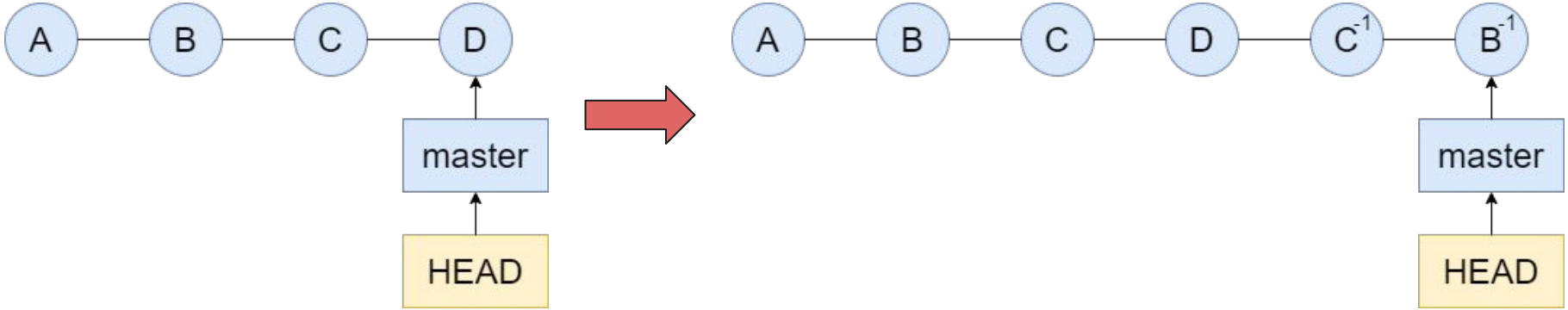
Undo Commits - git revert

- Revert some existing commits
- Usually for **public commit**
- Command: **git revert HEAD**



Undo Commits - git revert

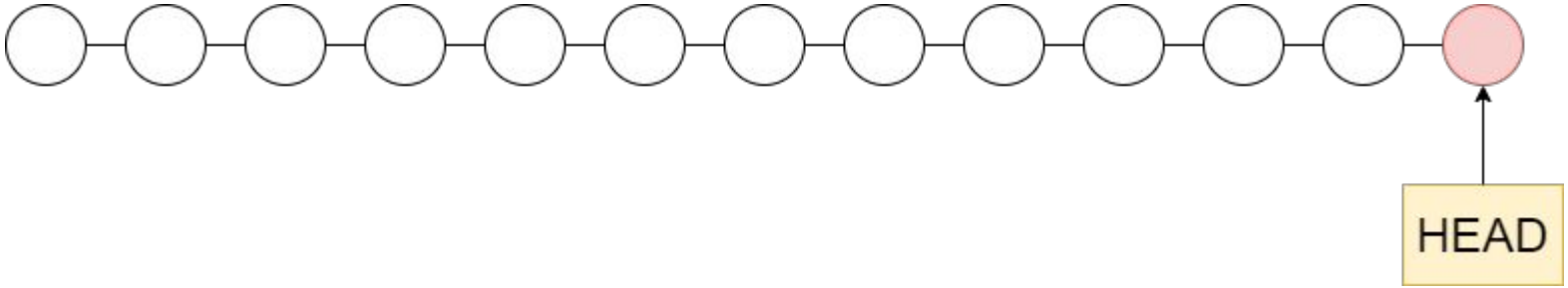
- Revert some existing commits
- Usually for **public commit**
- Command: **git revert HEAD~3..HEAD~1**



Debugging

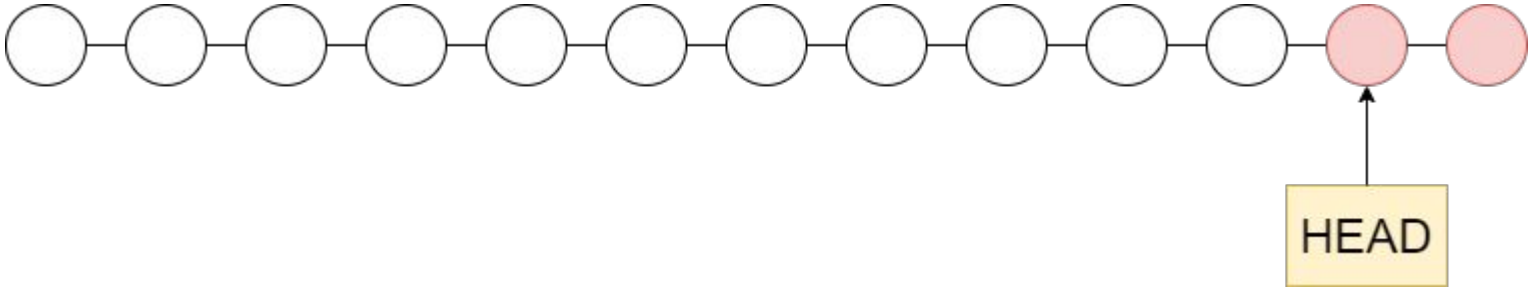
Solution 1: git checkout

- Testing result: Bad



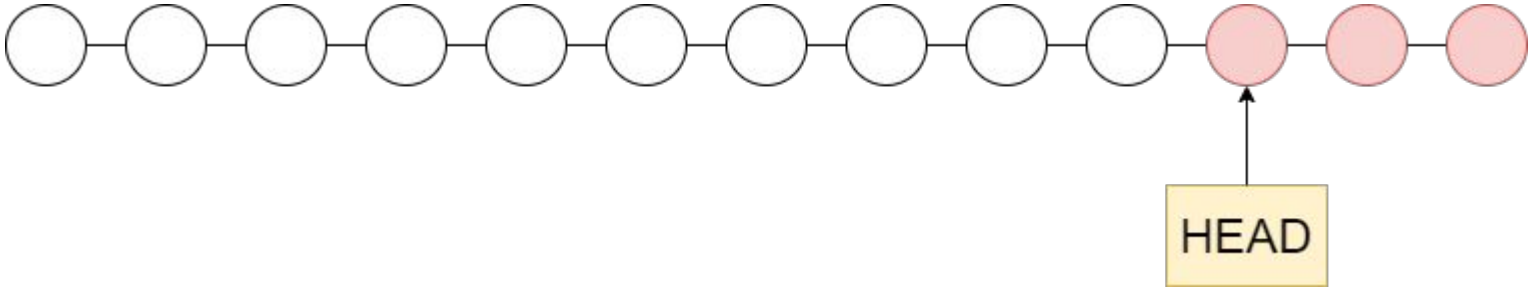
Solution 1: git checkout

- Testing result: Bad



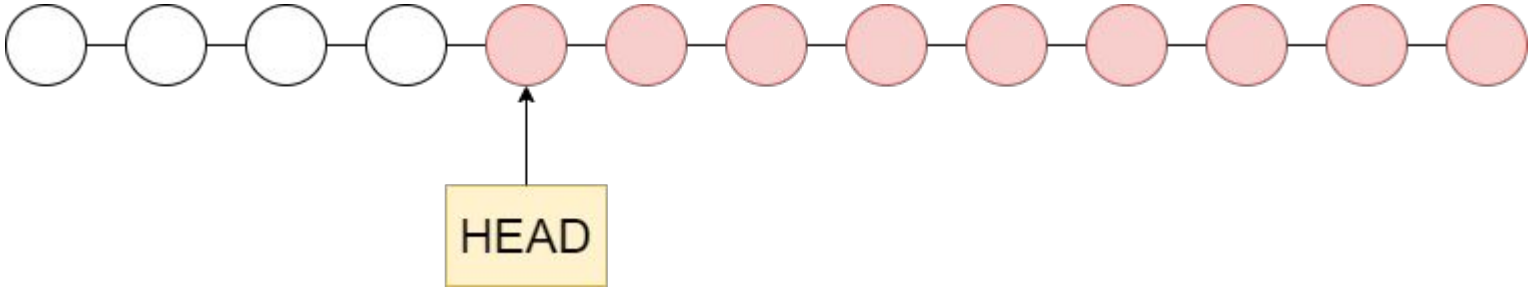
Solution 1: git checkout

- Testing result: Bad



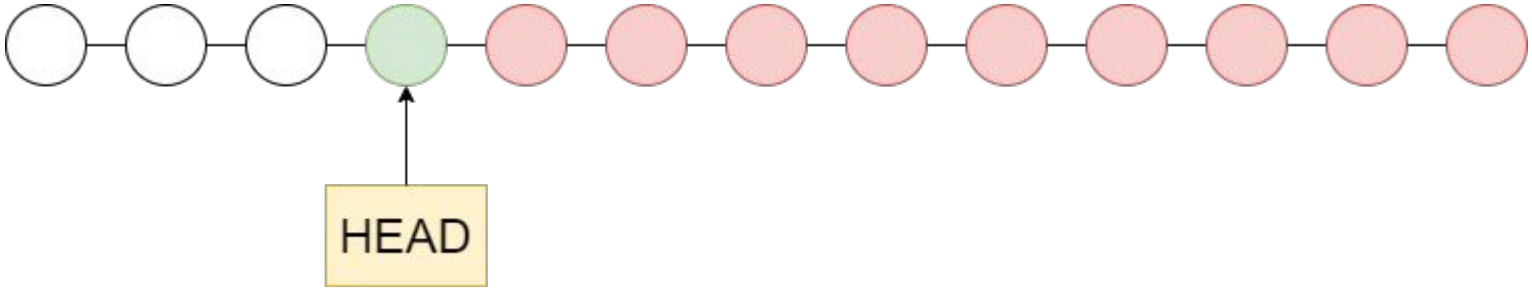
Solution 1: git checkout

- Testing result: Bad



Solution 1: git checkout

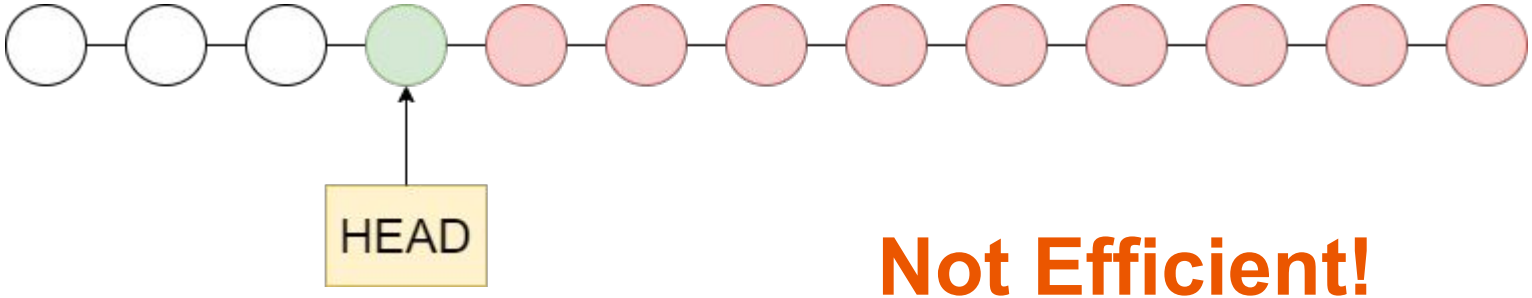
- Testing result: Good



Solution 1: git checkout

Good Bad

- Testing result: Good



Not Efficient!

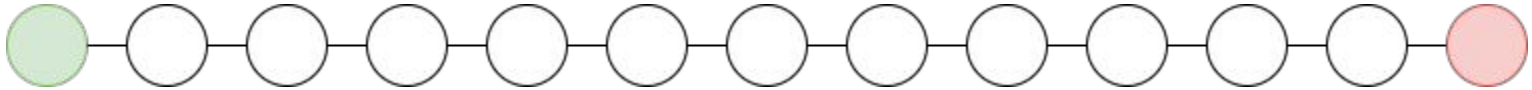
Solution 2: git bisect

- git bisect -
  Use binary search to find the commit that introduced a bug

```
> git bisect start
> git status
On branch master

You are currently bisecting, started from branch 'master'.
  (use "git bisect reset" to get back to the original branch)
```

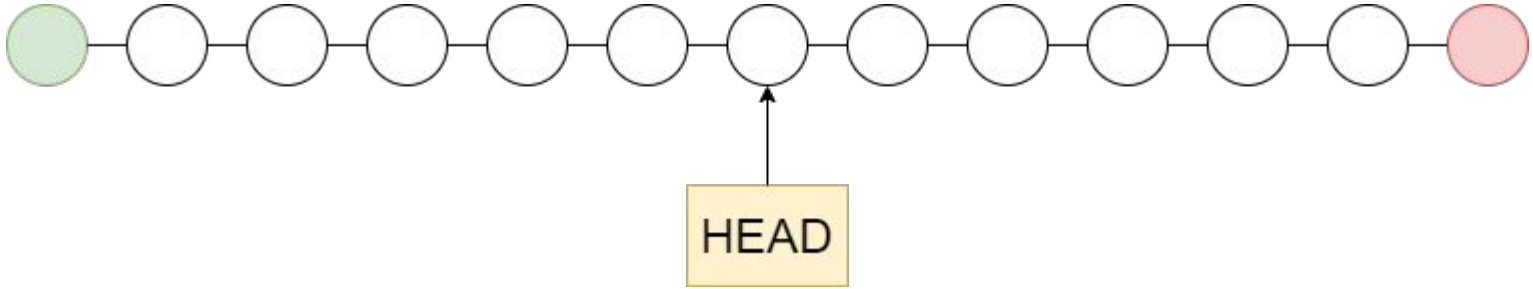
Solution 2: git bisect

- Command: **git bisect bad <COMMIT_ID>**
  git bisect good <COMMIT_ID>



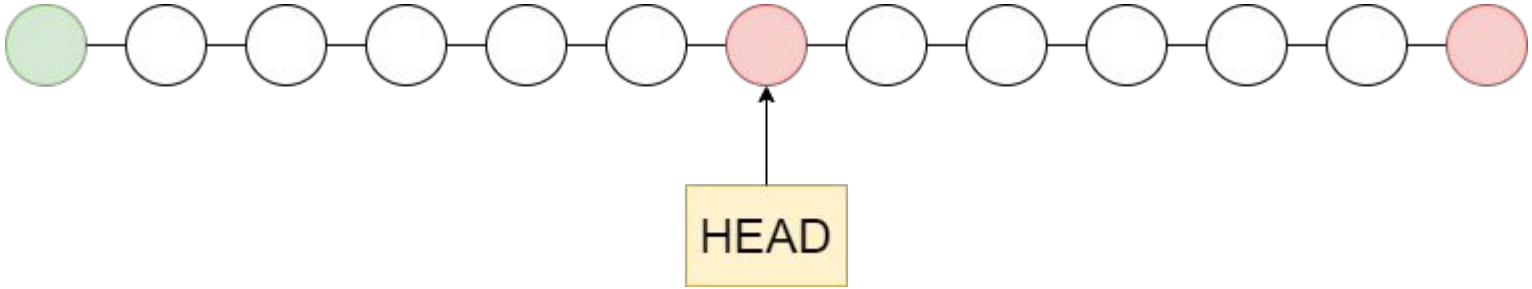
Solution 2: git bisect

- Checkout automatically



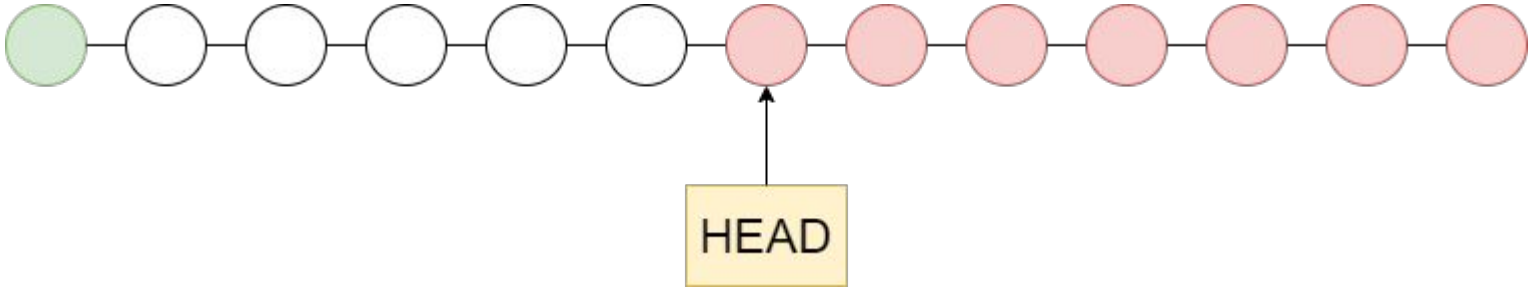
Solution 2: git bisect

- Testing result: Bad
- Command: **git bisect bad**



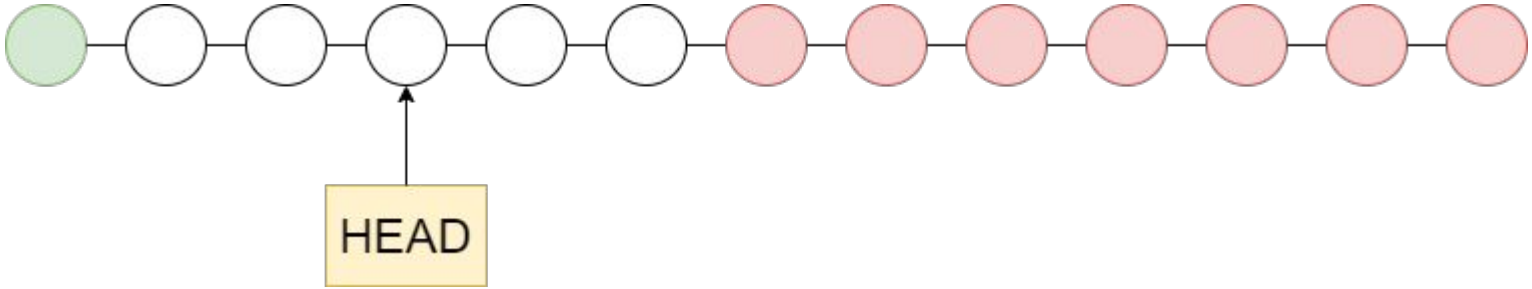
Solution 2: git bisect

- Commits after the current commit are Bad



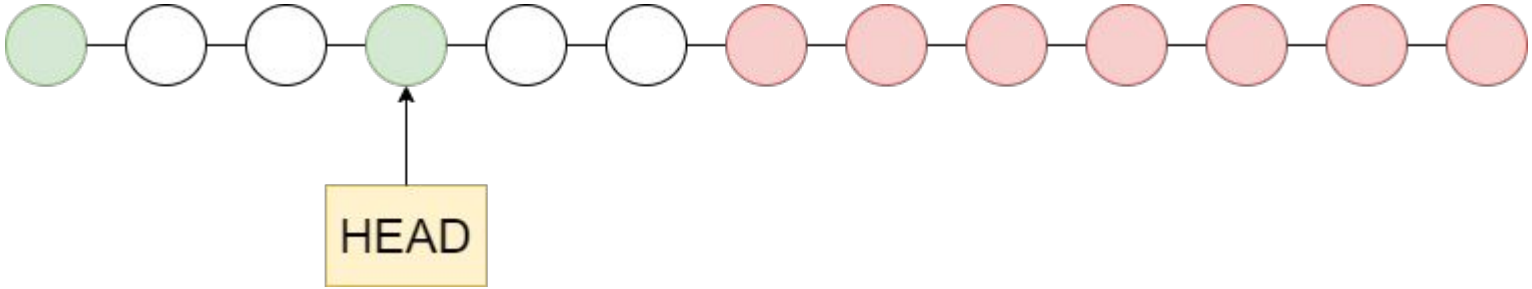
Solution 2: git bisect

- Checkout automatically



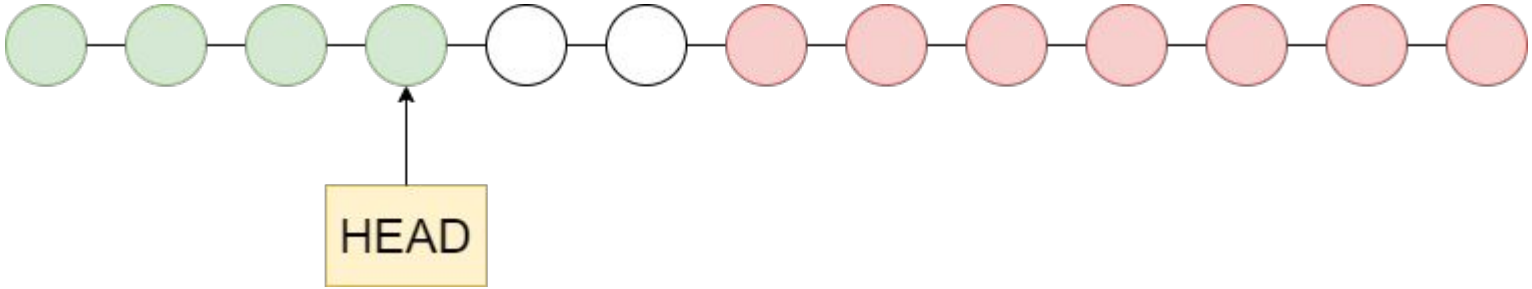
Solution 2: git bisect

- Testing result: Good
- Command: **git bisect good**



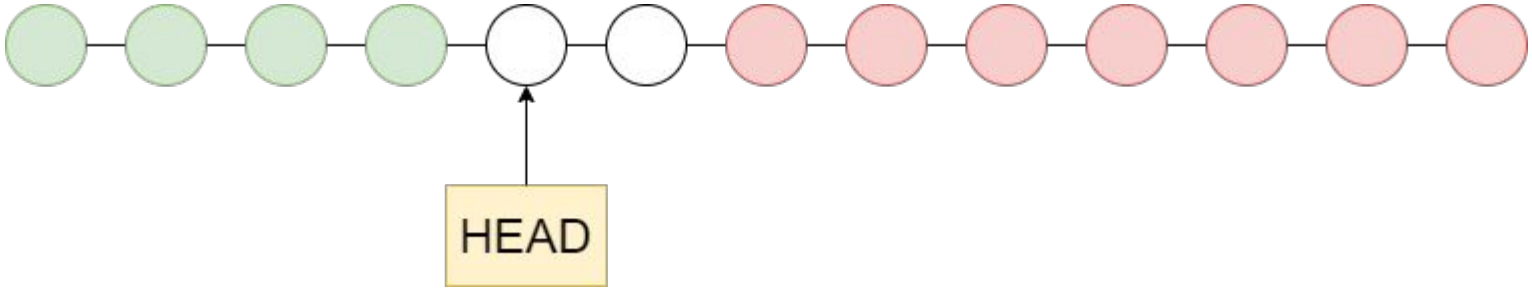
Solution 2: git bisect

- Commits before the current commit are Good



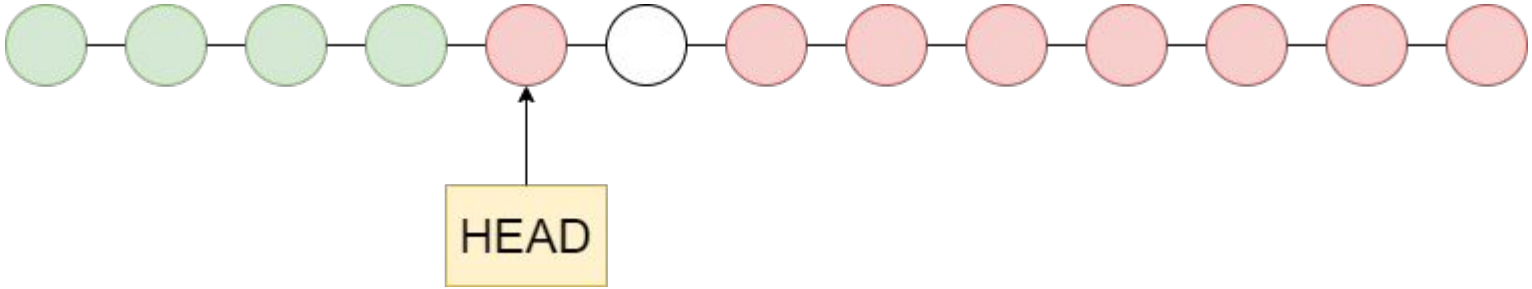
Solution 2: git bisect

- Checkout automatically



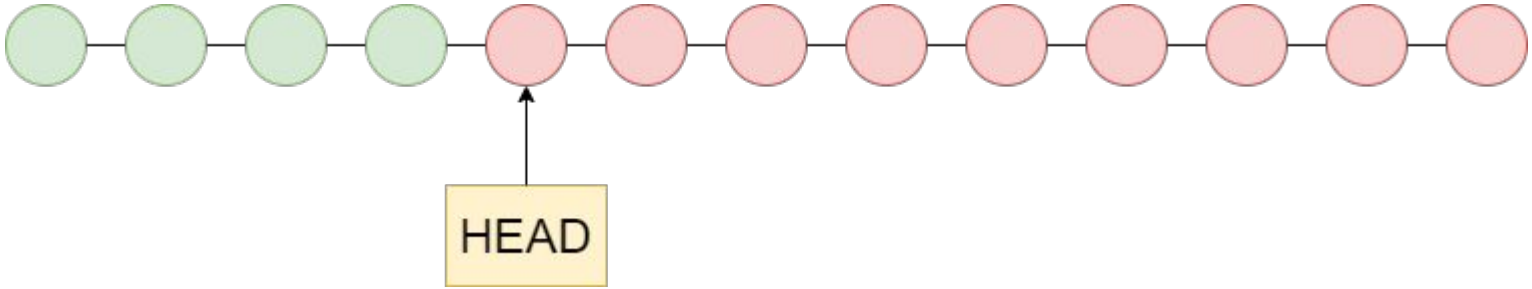
Solution 2: git bisect

- Testing result: Bad
- Command: **git bisect bad**



Solution 2: git bisect

- Commits after the current commit are Bad



Solution 2: git bisect

```
> git bisect bad
d2ea4298a23b449509aedb59959b28afd1d89718 is the first bad commit
commit d2ea4298a23b449509aedb59959b28afd1d89718
Author: Chih-Yu Kao <lcd010308@gmail.com>
Date:   Sun Mar 7 01:53:35 2021 +0800

    Second commit

 main.cpp | 1 +
 1 file changed, 1 insertion(+)
> git bisect reset
Previous HEAD position was d2ea429 Second commit
Switched to branch 'master'
```

Solution 2: git bisect

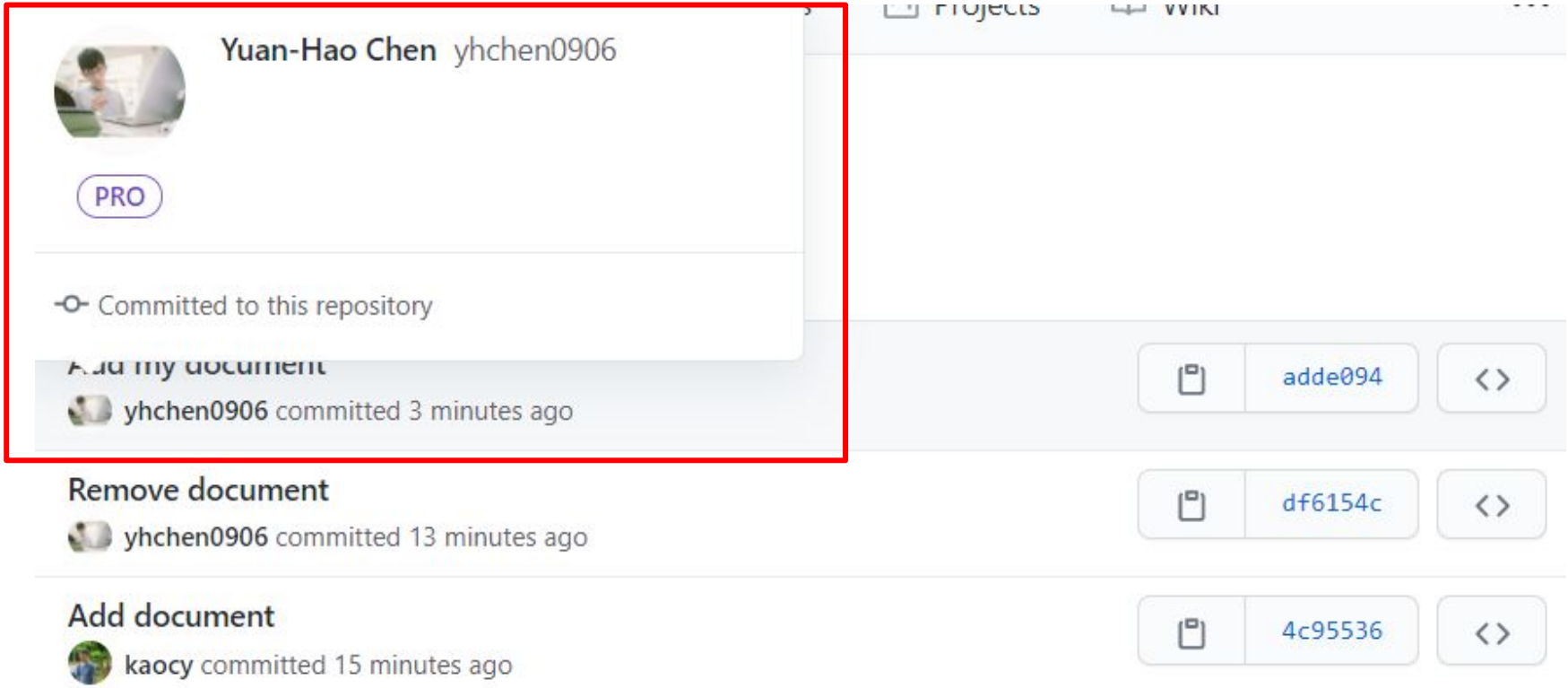
- Automatically bisect with script or command
- Use exit code to distinguish good/bad commits
  - exit code 0: Good
  - exit code 1~127 (except 125): Bad
  - exit code 125: skip

Solution 2: git bisect

```
> git bisect start
> git bisect bad 8618f96
> git bisect good 4232bdb
Bisecting: 1 revision left to test after this (roughly 1 step)
[0ca319e23ddd26e4f315b7746683548527f0e393] First commit
>
> git bisect run ./test.sh
```

Commit Spoofing

Who Made the Commit?



How Does GitHub Identify Commit Author?

- Use email set by the user in the beginning

```
> git config --global user.email "lcd010308@gmail.com"
> git config --global user.name "Chih-Yu Kao"
```

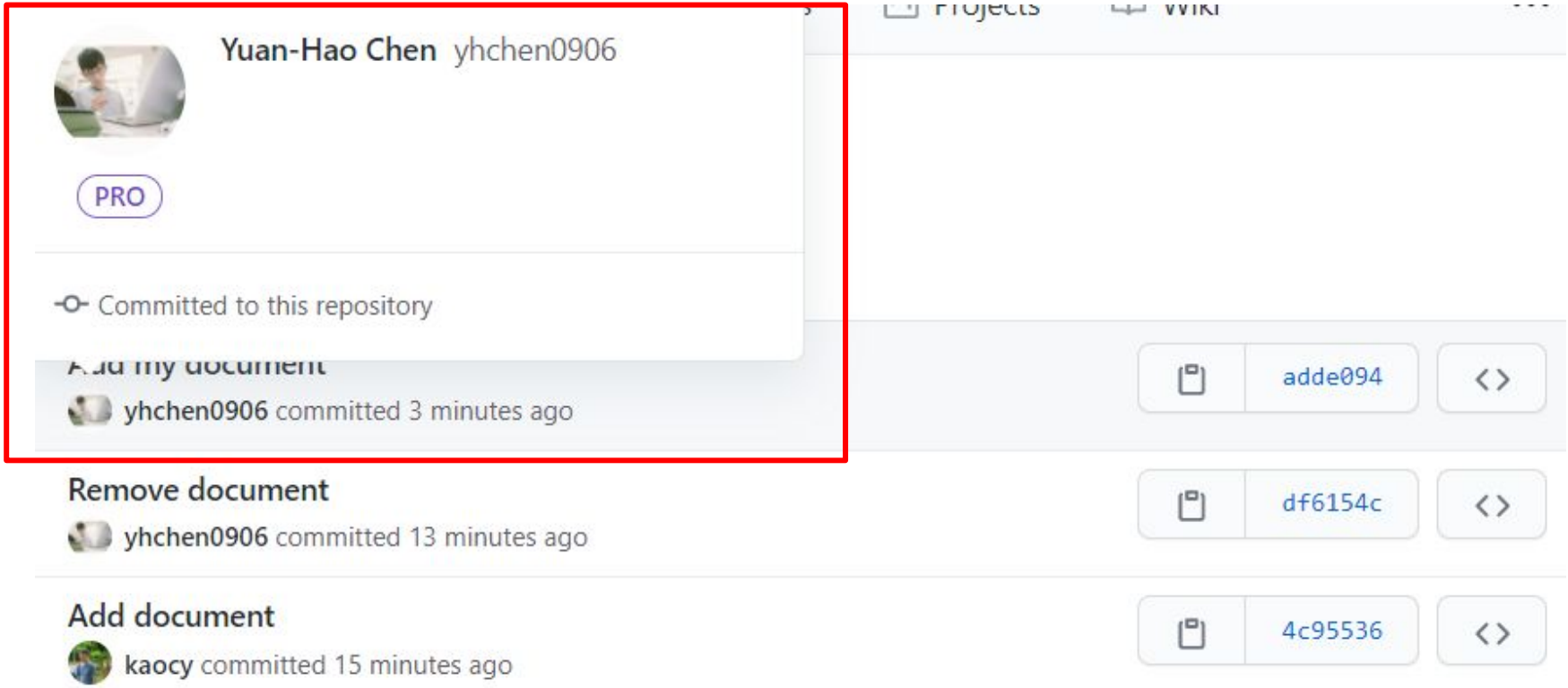
```
commit 4c955366462610865e729148209f8fa597721699
Author: Chih-Yu Kao <lcd010308@gmail.com>
Date:   Fri Mar 5 03:07:15 2021 +0800

    Add document
```

Spoof a Commit as Someone Else

- Change your email before committing

```
> git config user.email "yhchen0906@gmail.com"
> git config user.name "Yuan-Hao Chen"
>
> git commit -m "Add my document"
[master ff24ce0] Add my document
 1 file changed, 1 insertion(+)
> git push
```

Spoof a Commit as Someone Else



Commit Signing

Generate a GPG Key

```
> gpg --generate-key
gpg (GnuPG) 2.2.19; Copyright (C) 2019 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Chih-Yu Kao
Email address: lcd010308@gmail.com
You selected this USER-ID:
    "Chih-Yu Kao <lcd010308@gmail.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? O
```

Get GPG Key ID

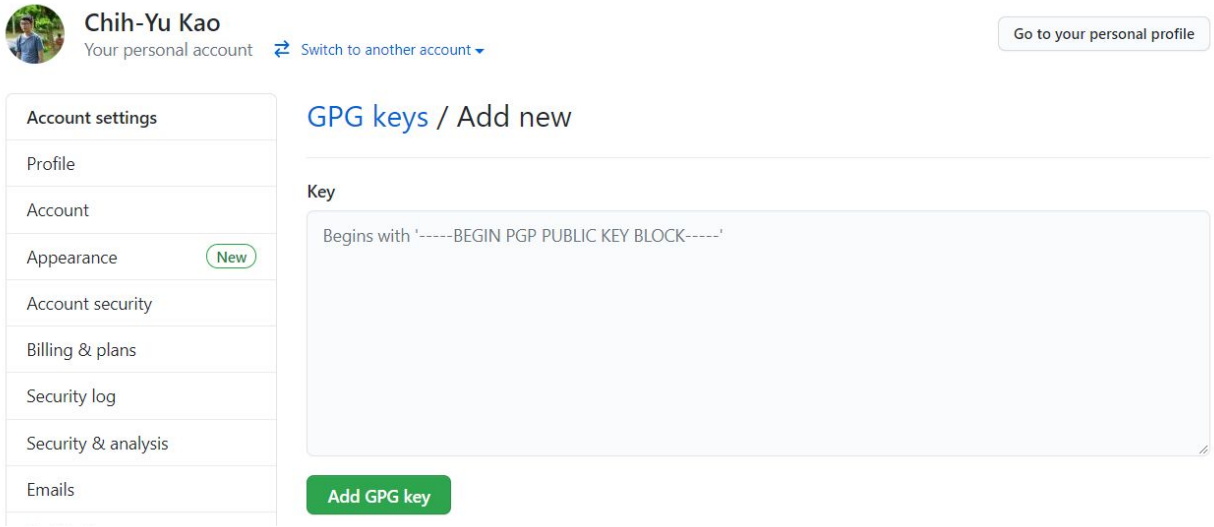
```
> gpg --list-keys
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model:
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n,
gpg: next trustdb check due at 2023-03-04
/home/kaocy/.gnupg/pubring.kbx
-------------------------------
pub   rsa3072 2021-03-04 [SC] [expires: 2023-03-04]
      C4B2F471F00B4A3E0F1A2EE2EA827E920B1A71DD   key ID
uid           [ultimate] Chih-Yu Kao <lcd010308@gmail.com>
sub   rsa3072 2021-03-04 [E] [expires: 2023-03-04]
```

Add Public Key to GitHub Account

- Get public key

```
> gpg --armor --export <KEY_ID>
```

- Settings > SSH and GPG keys > New GPG key



Set Git Config

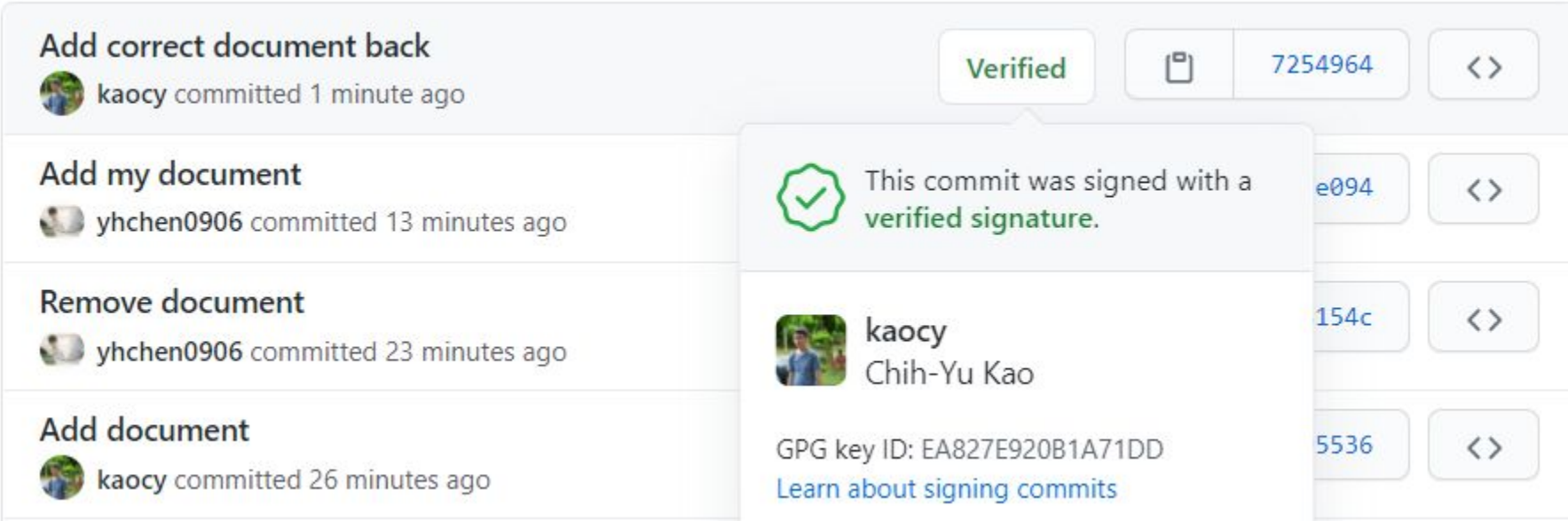
- Set your GPG signing key in Git

```
> git config --global user.signingkey <KEY_ID>
```

- Sign all commits by default

```
> git config --global commit.gpgsign true
```

Commit with Verified Signature



Appendix

Why Fail to Checkout?

Why Fail to Checkout?

- Uncommitted changes would be overwritten

```
> git status
On branch test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.cpp

no changes added to commit (use "git add" and/or "git commit -a")
> git checkout master
error: Your local changes to the following files would be overwritten by checkout:
        main.cpp
Please commit your changes or stash them before you switch branches.
Aborting
```

Solution 1: git stash

- git stash push - Save your local changes to a new stash entry

```
> git stash push -m "add testing function"
Saved working directory and index state On test: add testing function
>
> git status
On branch test
nothing to commit, working tree clean
>
> git checkout master
Switched to branch 'master'
```

Solution 1: git stash

- git stash list - List the stash entries that you currently have

```
> git stash list
stash@{0}: On refactor: refactor main function
stash@{1}: On test: add testing function
```

stash name branch name description

Solution 1: git stash

- git stash pop [<stash>] - Remove and apply the stash back

```
> git checkout test
Switched to branch 'test'
>
> git stash pop stash@{1}
On branch test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.cpp

no changes added to commit (use "git add" and/or "git commit -a")
Dropped stash@{1} (3d03a82f6fdb89d6dce1df1971976669c39e8c00)
```

Solution 2: git commit & reset

- Create a temporary commit

```
> git add .
> git commit -m "not finish yet"
[test 117a4e2] not finish yet
 1 file changed, 1 insertion(+)
>
> git checkout master
Switched to branch 'master'
```

Solution 2: git commit & reset

- Use **git reset** to undo the temporary commit

```
> git checkout test
Switched to branch 'test'
>
> git reset HEAD^
Unstaged changes after reset:
M       main.cpp
>
> git status
On branch test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.cpp
```

Undo Changes - File

```
> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.cpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Makefile
```

Staged

Unstaged

Untracked