Git & GitHub

pt2

Andrea Fornaia, Ph.D.

Department of Mathematics and Computer Science

University of Catania

Viale A.Doria, 6 - 95125 Catania Italy

fornaia@dmi.unict.it

http://www.cs.unict.it/~fornaia/

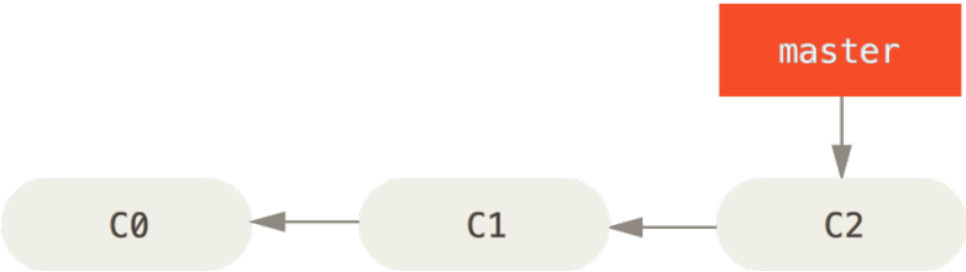
Branching Workflow

Branching workflows: basics

- **Long-Running Branch**
  - always open (never deleted)
  - used for different stages of development cycle with various levels of stability
  - when a more stable level is reached they're merged into the branch above them
    - master: only stable code
    - develop: code under development
- **Topic Branch**
  - short-lived branch (deleted after merge)
  - created and used for a single particular feature or work
    - New feature
    - Issue fix
    - Hot fix
  - generally too expensive in older VCS
    - create and merge branches was costly

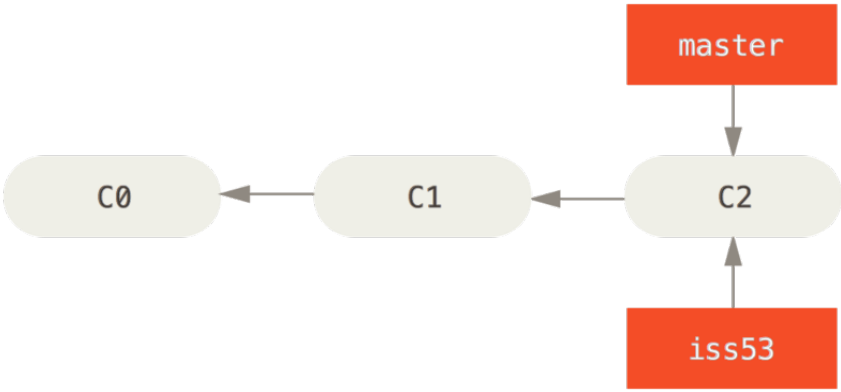
Branching and Merging Scenario

You're working on your project and have a couple of commits already



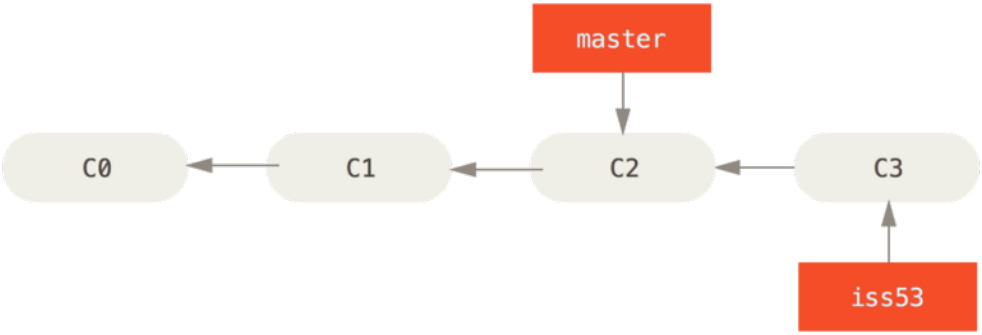
Work on issue #53: create a branch and switch to it at the same time:

```
$ git checkout -b iss53
Switched to a new branch "iss53"
```



Branching and Merging Scenario

```
$ vim index.html
$ git commit -a -m 'added a new footer [issue 53]'
```



OOOPS!! A critical bug need to be fixed in production code!!! We need to switch to master. We will not lose any commits of current feature branch

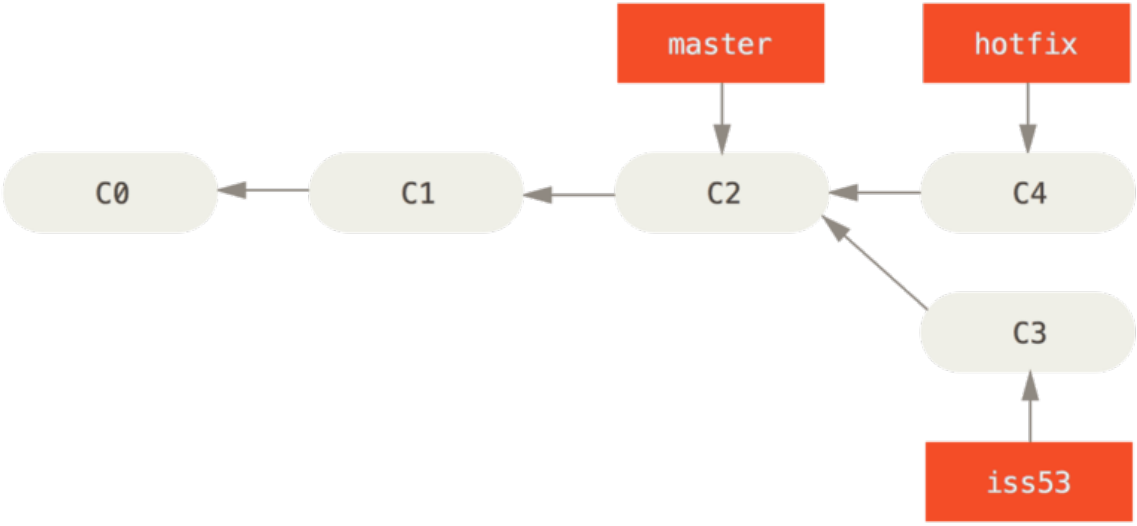
```
$ git checkout master
Switched to branch 'master'
```

Git resets your working directory to look like it did the last time you committed on that branch

Branching and Merging Scenario

Let's create a **hotfix branch** on which to work until it's completed:

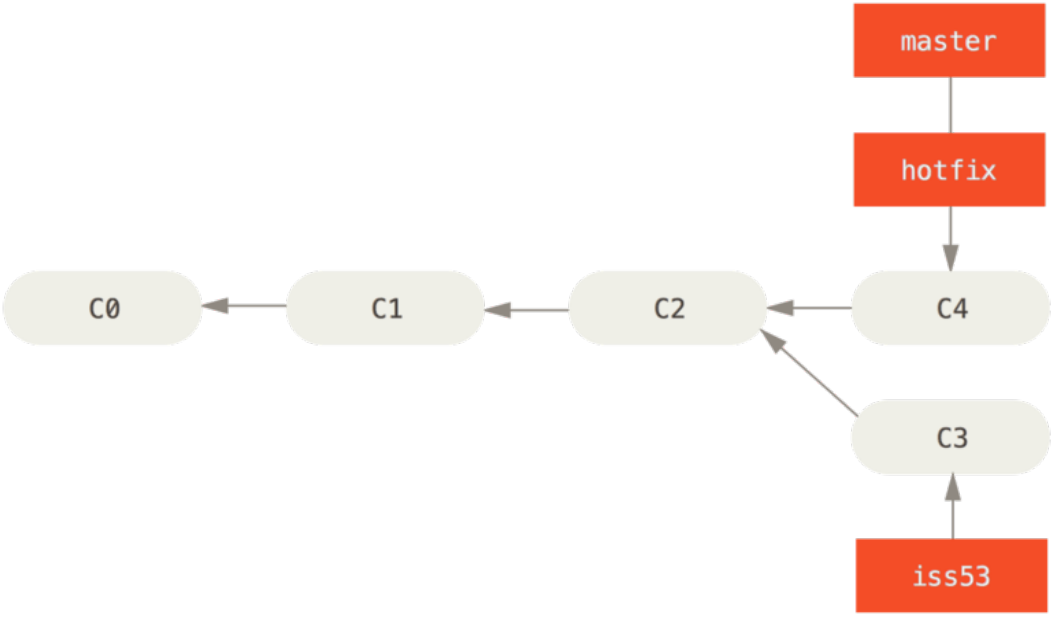
```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ vim index.html
$ git commit -a -m 'fixed the broken email address'
[hotfix 1fb7853] fixed the broken email address
 1 file changed, 2 insertions(+)
```



Branching and Merging Scenario

Merge it back into your master branch to get it into production

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward                # to avoid ff use --no-ff
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```



Branching and Merging Scenario

Delete the hotfix branch, because the master branch points at the same place

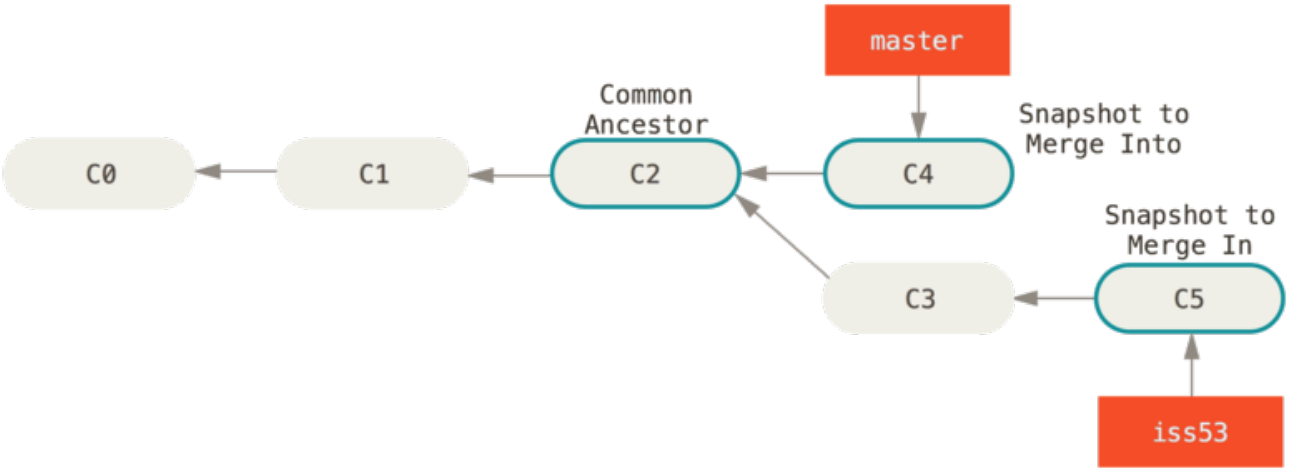
```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).
```

After your super-important fix is deployed, you’re ready to switch back to the work you were doing before you were interrupted

```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'finished the new footer [issue 53]'
[iss53 ad82d7a] finished the new footer [issue 53]
1 file changed, 1 insertion(+)
```

Branching and Merging Scenario

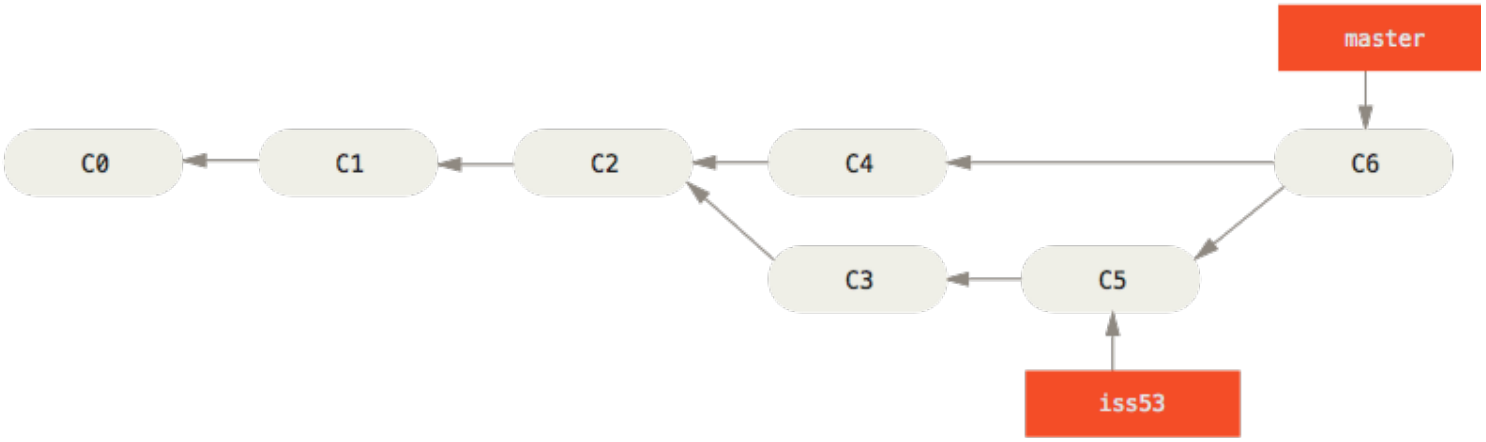
- We are ready to merge issue #53 work into master branch



- In this case, your development **history has diverged** from some older point
- Git cannot just move branch pointer forward
- Git creates a new snapshot that results from this **three-way merge**
- Automatically creates a new commit that points to it
  - referred as **merge commit** (C6, see next slide)
  - it's special: has more than one parent

Branching and Merging Scenario

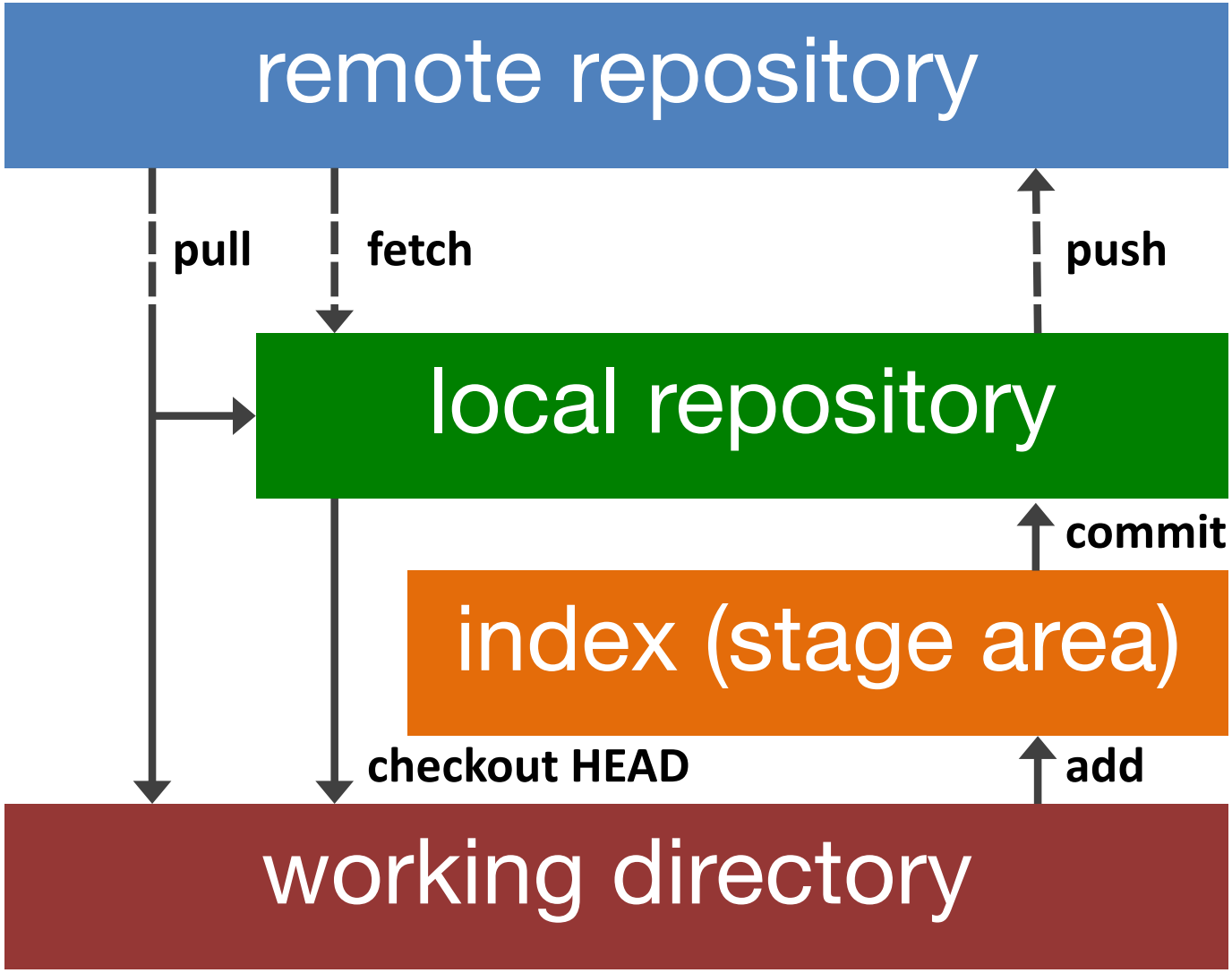
```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```




```
$ git branch -d iss53
```

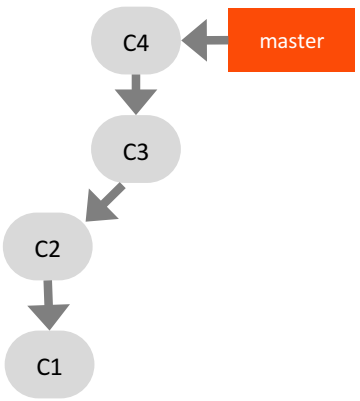
Go Remote

Working with Remotes



Local And Remote

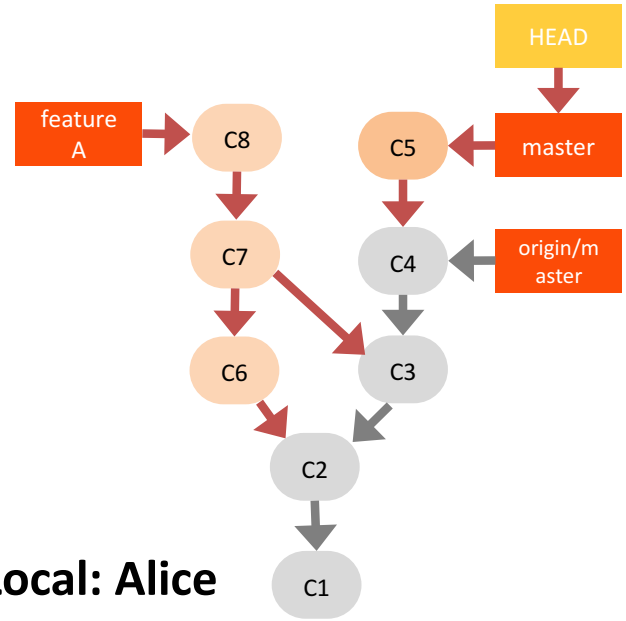
REMOTE
(origin)



GitHub

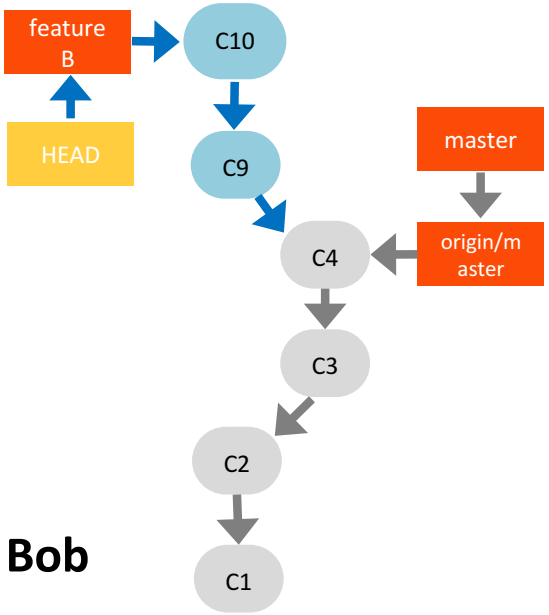
GitLab

Bitbucket



Local: Alice

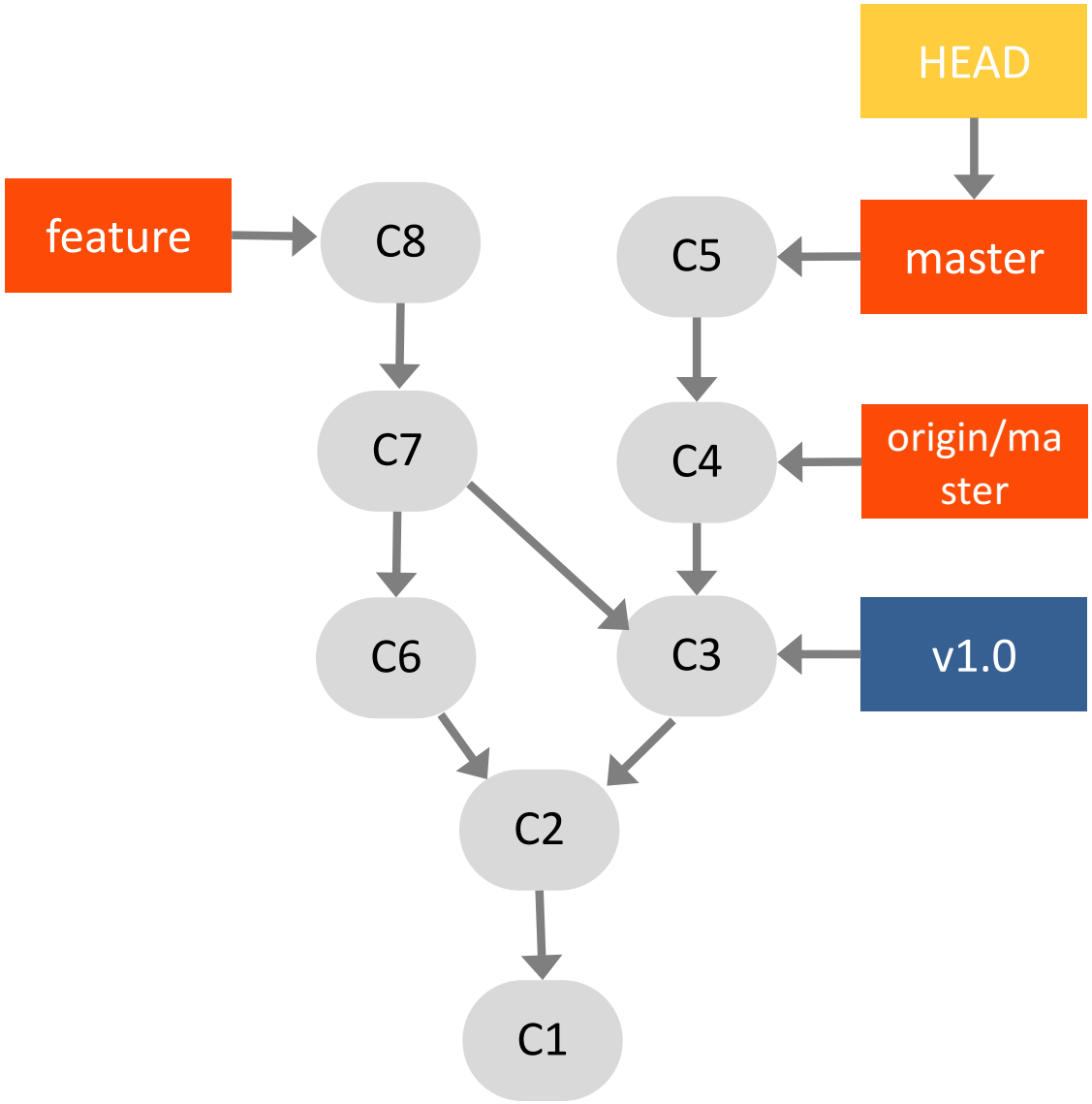
git



Local: Bob

git

A complete Commit Graph



Labels are **just pointers to commits**

There are three types of **pointers** to access commits:

- Branch (local and remote)
- Tag (a branch that can't be moved)
- Special (HEAD: "current branch")

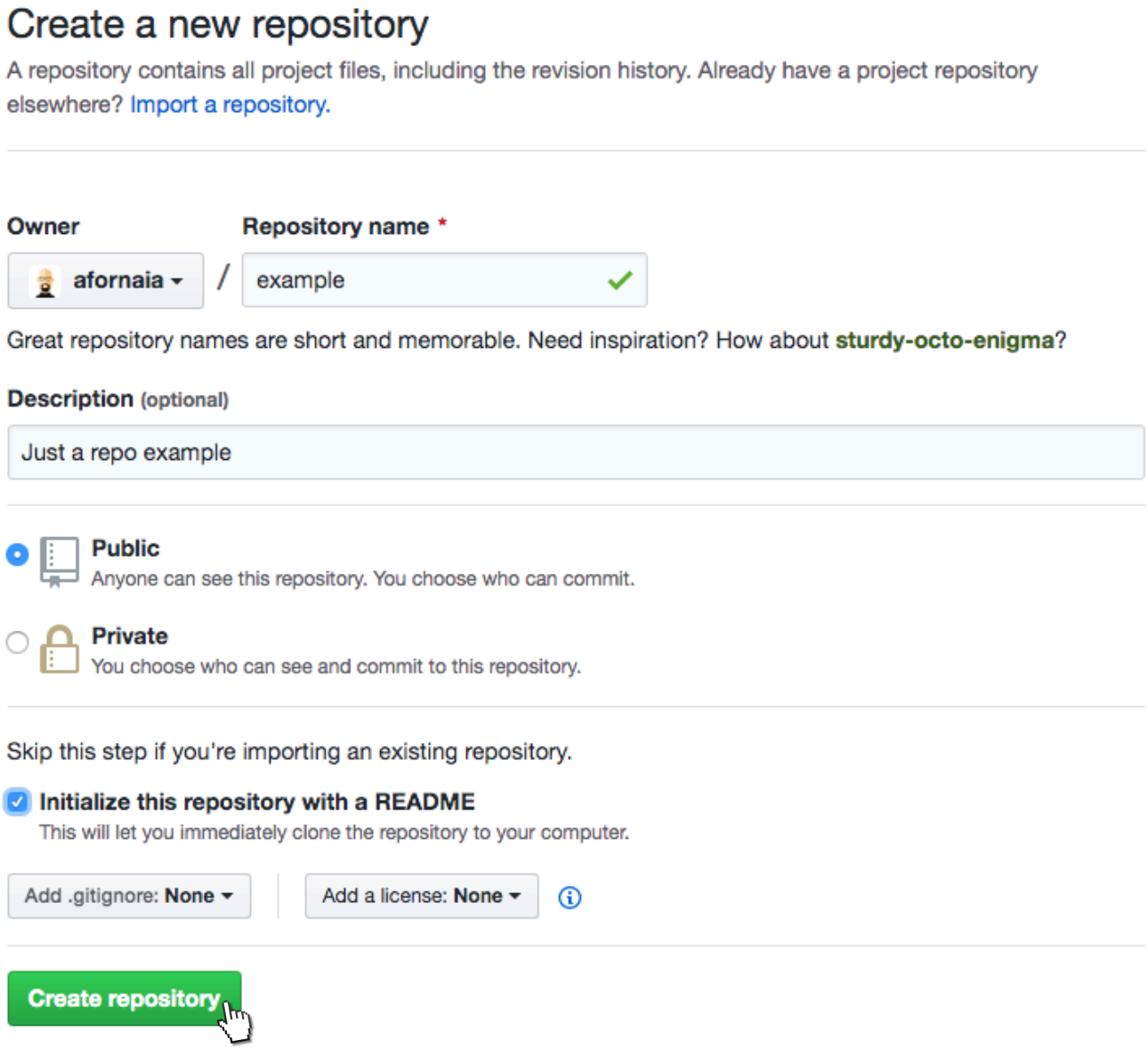
A **remote branch** tell us the position of a branch in a remote repository (we need to periodically fetch this information to keep it in sync)

Note: we may have multiple remote! E.g.:

- origin: our remote repository on GitHub
- upstream: the repository we forked

Create a private repository on GitHub

- Register
- Click on “+”
- > New Repository



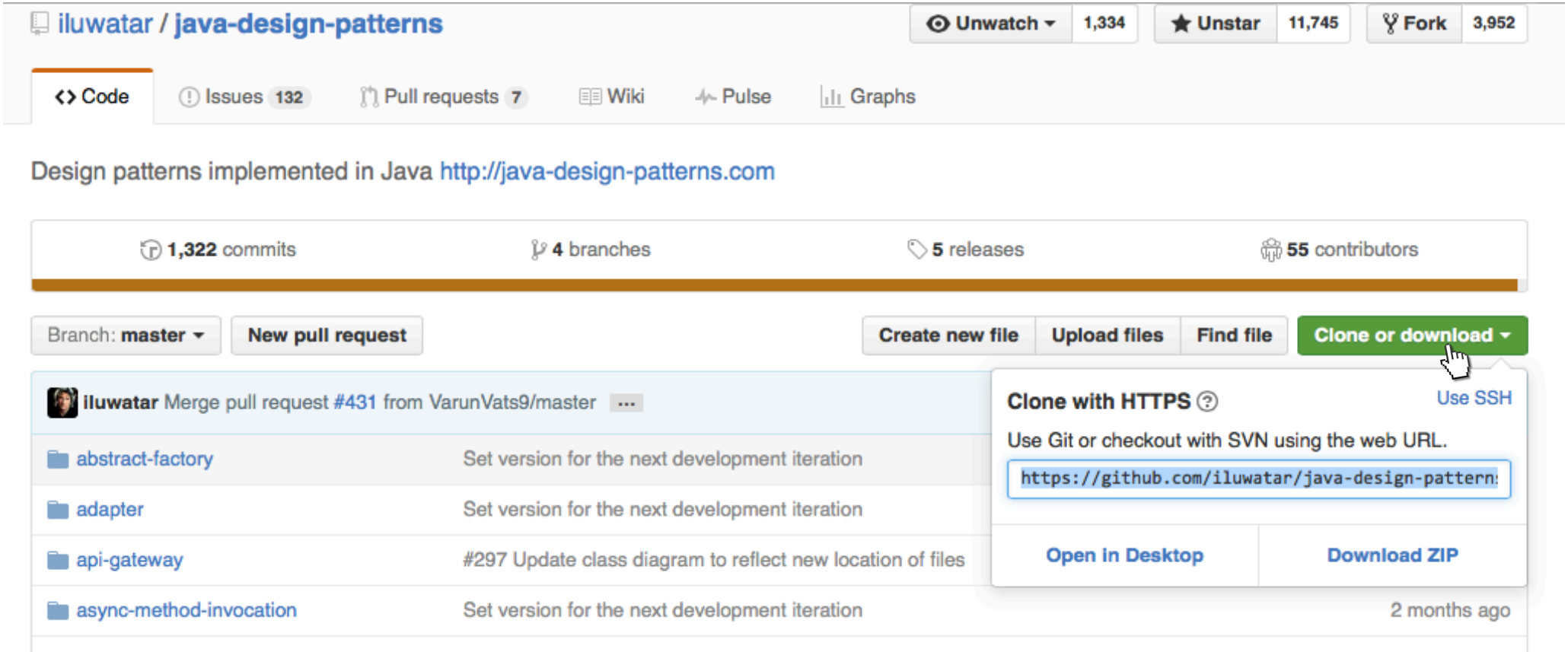
Cloning an **Existing** Repository

- If you want to get a copy of an existing Git repository, for example, a project you'd like to contribute to, type:

```
$ git clone https://github.com/project_name
```

- What does it do?
  - Creates a directory named "project_name"
  - Initializes a **.git** directory inside it
  - Pulls down all the data for that repository
  - Checks out a working copy of the latest version (HEAD)
- Default branch is **master**
- Supports different transfer protocols
  - https://github.com/project_name (https)
  - user@server:path/to/repo.git (ssh)

Cloning Existing Project



Remotes

- Collaborating with others involves managing remote repositories
- **origin** is the server you cloned from
- You can have different remotes

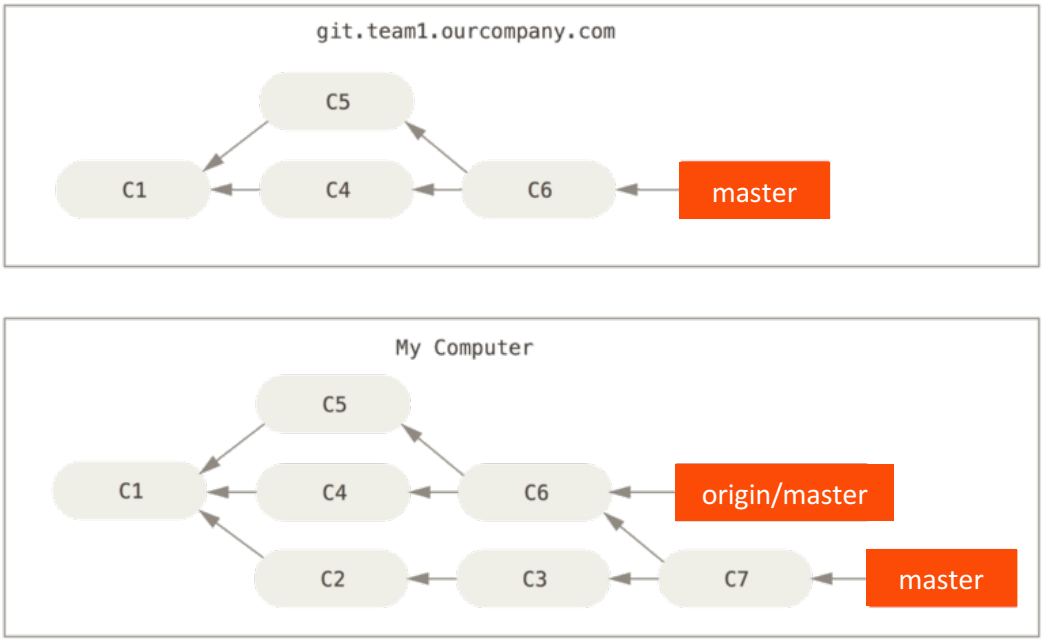
```
# list remotes (-v for verbose output, include urls)
$ git remote -v
origin https://github.com/afornaia/example.git
```

- Connect a **local initialised repository** to remote server

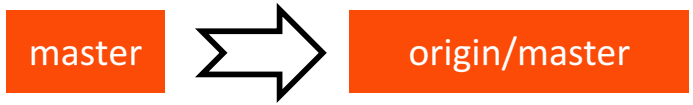
```
$ git init
$ git remote add origin https://github.com/afornaia/example.git
```

- Remote repository must be **empty**! Two repositories must have at least the **same root commit**!

branch-to-upstream mapping



branch-to-upstream mapping



master **tracks**
the remote branch
origin/master

```
$ git branch -v                          # list local branches
* master a366c90 C7
$ git branch -v --all                    # list all branches
* master a366c90 C7
  remotes/origin/master b786a41 C6
$ git branch --v                         # branch-to-upstream mapping
* master a366c90 [origin/master] C7      # (remote-tracked branch)
```

Commands

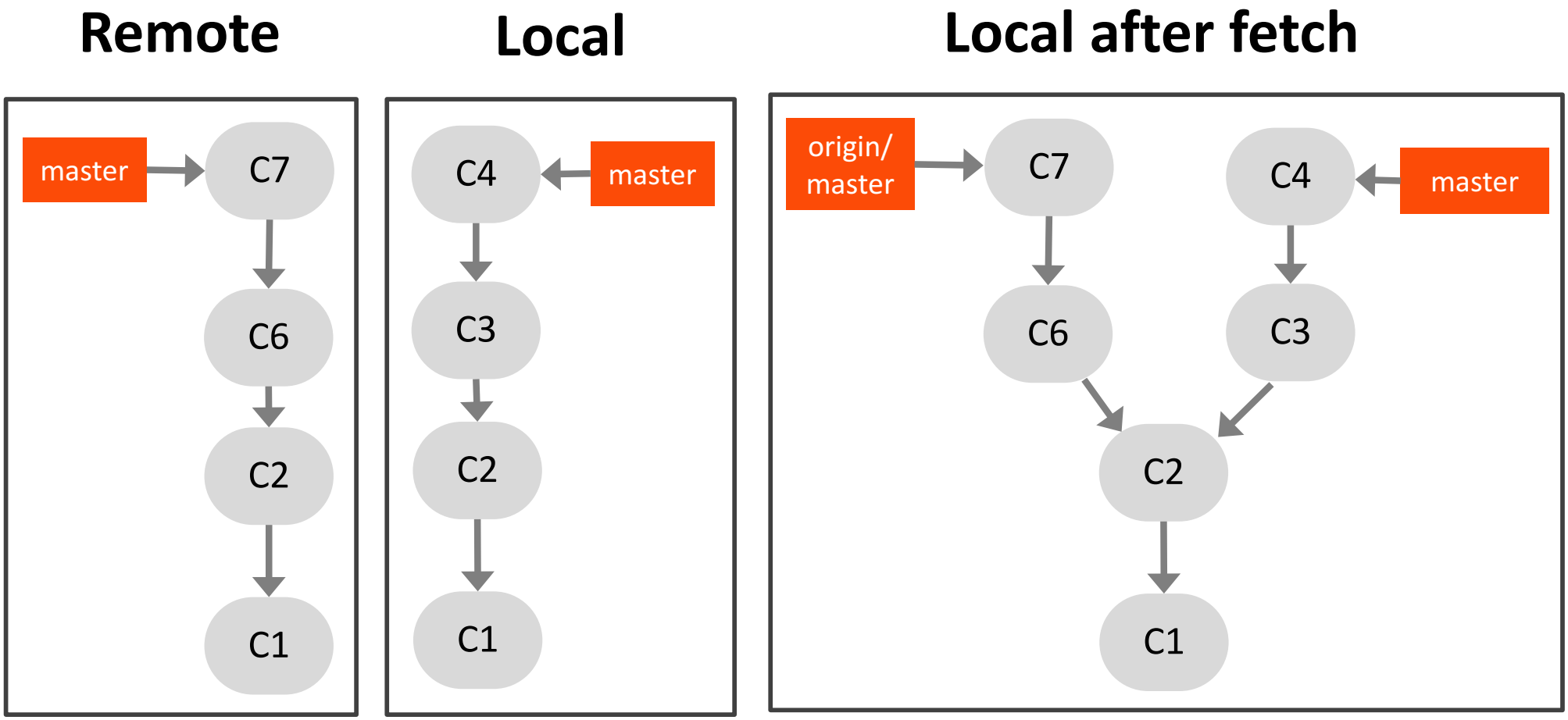
- **Fetch:** download <u>all</u> remote branches without updating (merging into) local ones
- **Pull:** update local branch with remote changes
  - Merge local changes with remote ones
- **Push:** update remote branch with local changes
  - Local branch must be up to date with remote one (need to pull before push)

Git fetch

- Downloads commits, files, and refs (branches) from a remote repository into your local repo
- See everybody else (pushed) work
- Doesn’t force you to merge the changes into your local branches
- Remember: git stores all commits, local and remote

```
$ git fetch <remote>
```

Git fetch


Remote
master
C7
C6
C2
C1
Local
C4
master
C3
C2
C1
Local after fetch
origin/
master
C7
C4
master
C6
C3
C2
C1

Git pull

- Update a local branch with remote changes
- Fetch content from a remote repository and automatically **merge** changes only into working branch:
- **Merges a remote branch into a local branch**

```
$ git pull <remote> <branch>
```

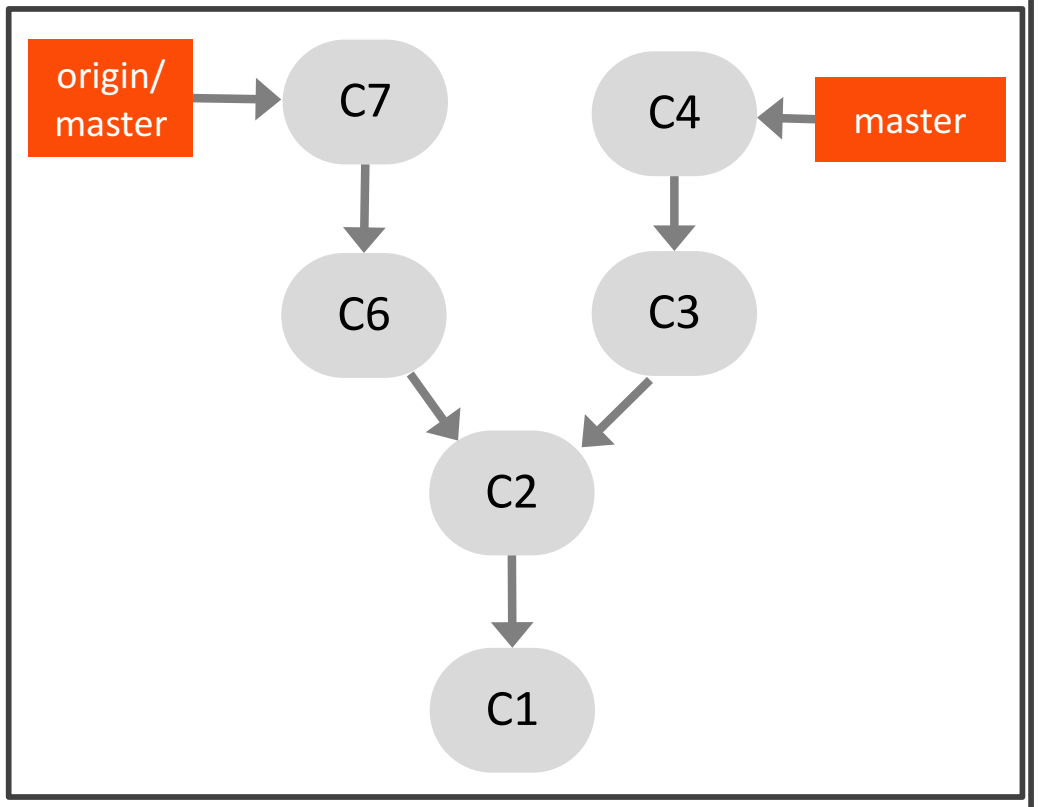
- Same as a (clearer and safer) fetch & merge

```
$ git fetch <remote>
$ git merge <remote>/<branch>
```

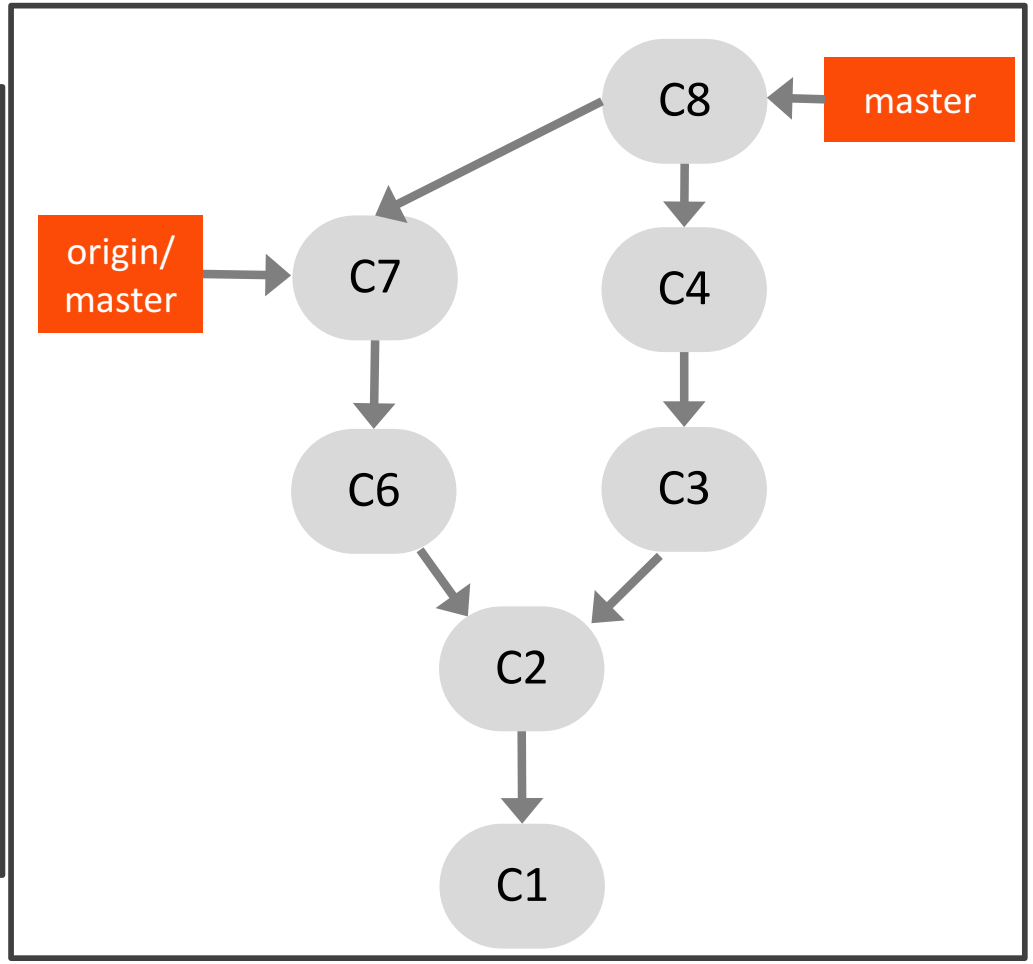
- Try to use fetch & merge instead of pull
- In case of problem it's often difficult to work out the reason

Git pull = fetch + merge

Local after fetch



Local after merge



Git push

- Update remote branch with local changes

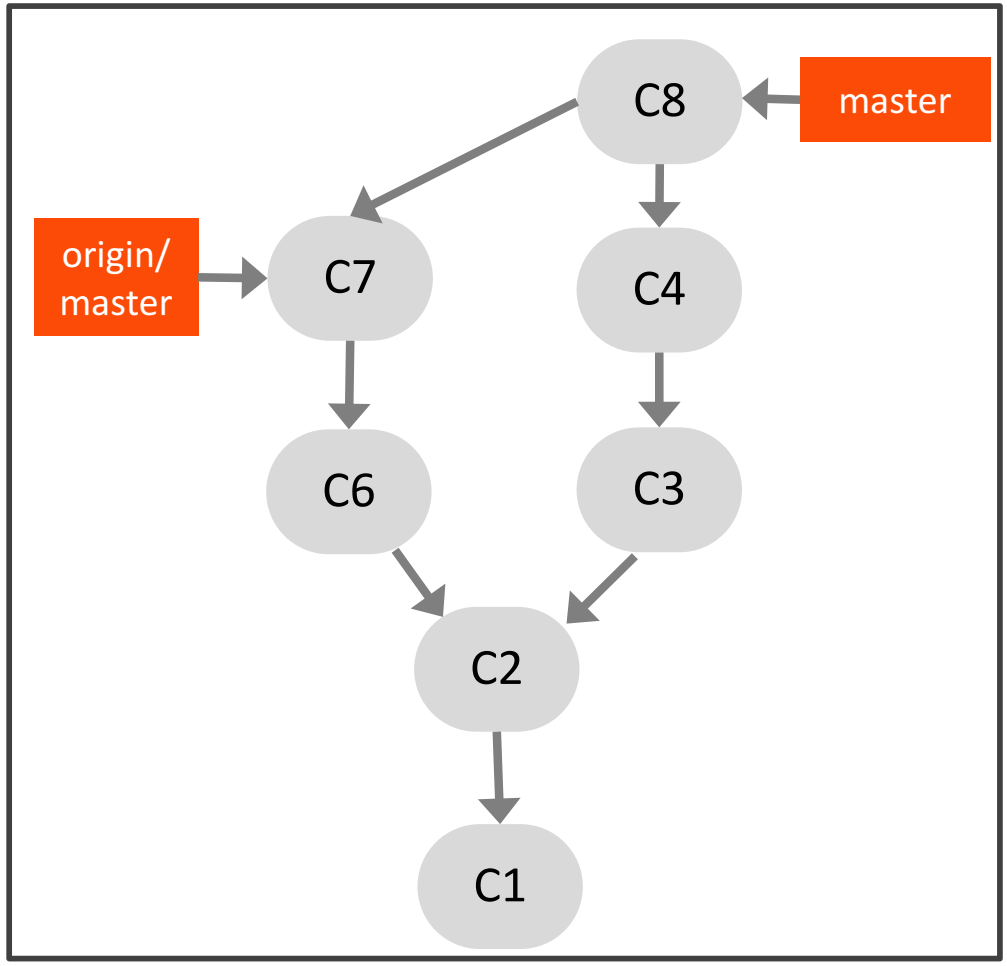
```
$ git push <remote> <branch>
```

- This creates or update the branch in the remote repository
- **<u>Merge a local branch into a remote branch</u>**

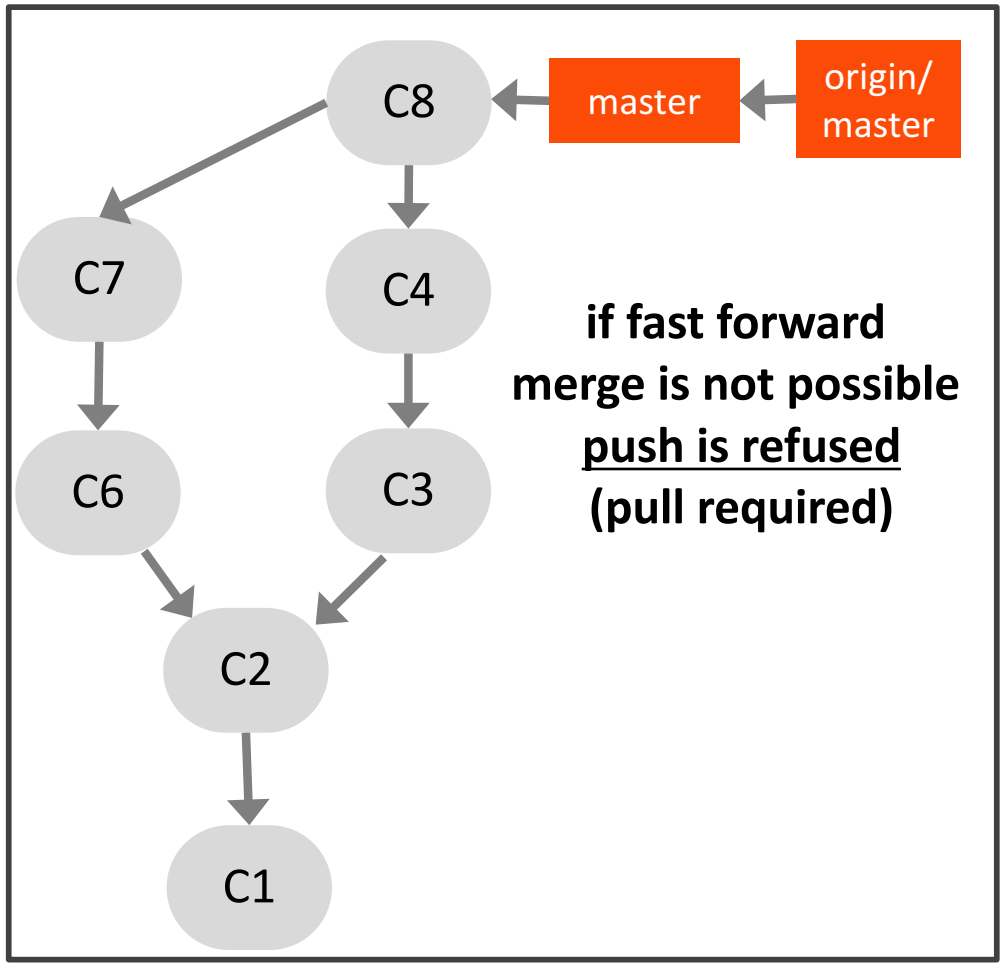
- Git won't let you push when it results in a non-fast-forward merge in the destination repository
  - The local branch is not up to date
  - **Prevents you from overwriting commits**
  - To solve the problem, **update the branch** content with either pull or fetch & merge

Git push = remote ff merge

Local after pull



Local <u>and</u> Remote after push



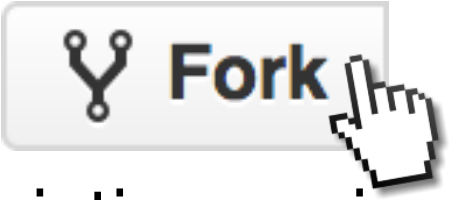
GitHub

- A home for free public git repositories
- Interface for exploring git repositories
- Real open source
  – immediate, easy access to the code
- Fork it, try it, learn it
- Social Coding

Why use GitHub

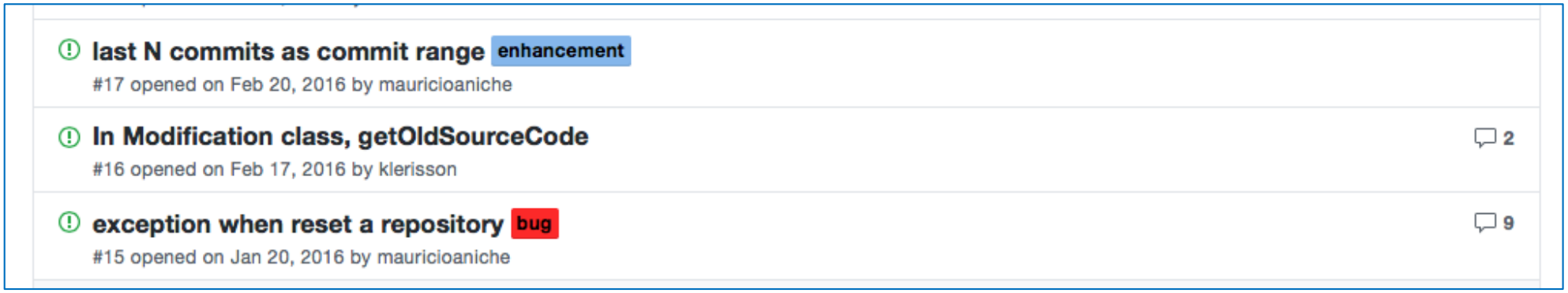
- It takes care of the **server aspects of git**
- And **much more**:
  - Exploring code and its history
  - Tracking issues
  - Pull Requests
  - Much more…
- Facilitates:
  - Learning from others
  - Seeing what people are up to
  - Contributing to others' code
- Lowers the barrier to collaboration
  - "There's a typo in your documentation" vs.
  - "Here's a correction for your documentation”
- **Improve your CV** contributing to open-source projects

Forking Projects



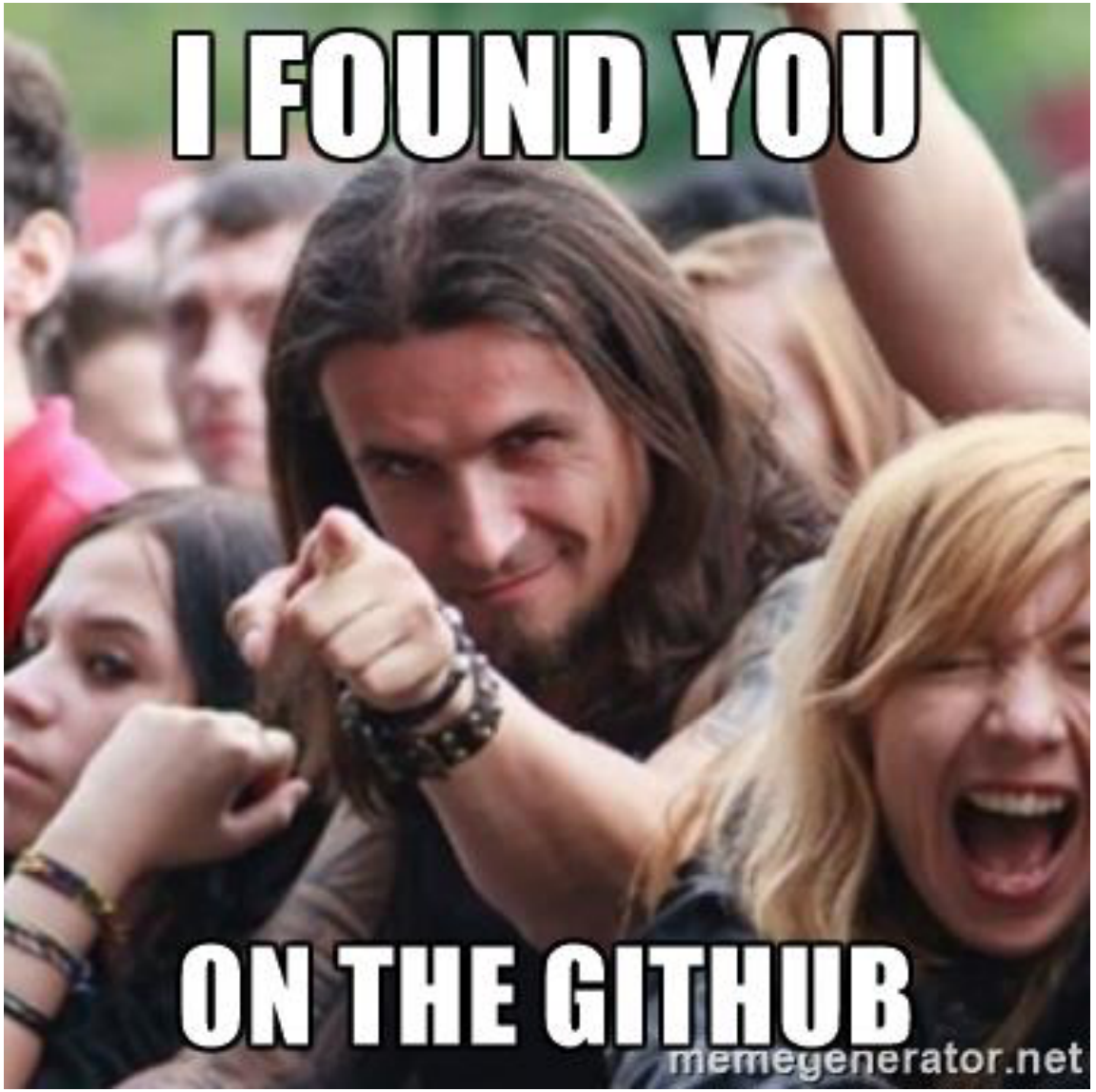
- Contribute to an existing project to which you **don't have push access**
- GitHub will make a **copy of the project that is entirely yours**
- No need to add users to projects as collaborators to give them push access
- People can push their changes back to the original repository by **Pull Requests (PR)**

Issues



- **Requests** for contribution:
  - Bugfix
  - Enhancement
  - New Feature
- It opens an **issue discussion**
- A **good way to start** with GitHub open-source projects:
  - fork the project
  - work on one of the issues
  - propose your solution with a Pull Request

[https://github.com/mauricioaniche/repodriller]

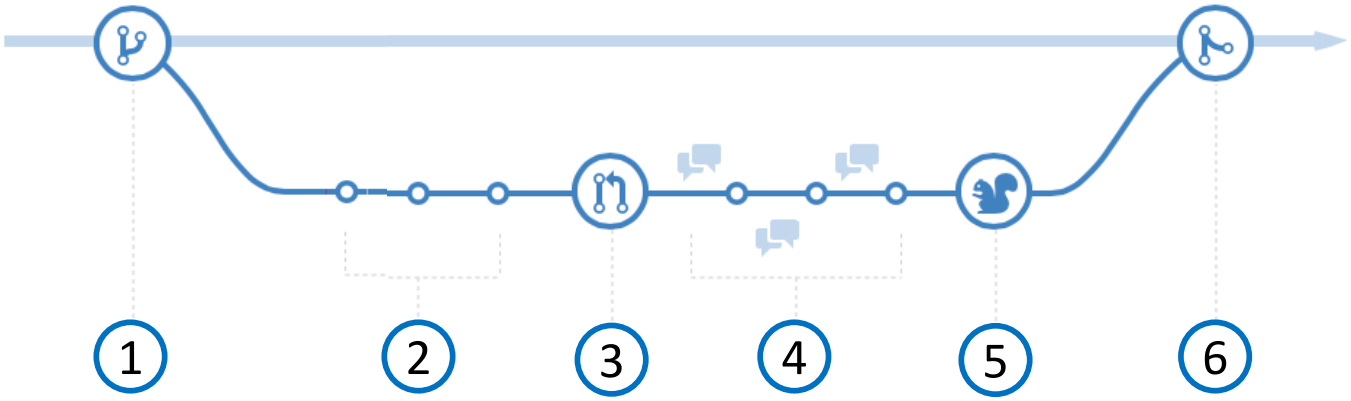

I FOUND YOU
ON THE GITHUB
memegenerator.net

Due modi per contribuire su GitHub

- **Merge request** (push abilitato):
  - Lo sviluppatore deve avere **accesso in scrittura** (push abilitato), facendo parte dei **contributors** del progetto (origin)
  - Consiste nell richiedere il **merge di due branch nello stesso repository**
  - Tipicamente tra un topic branch e il master in origin
  - Tipicamente il **master è protetto**, e solo il maintainer può approvarne i commit

- **Pull request** (push disabilitato):
  - Lo sviluppatore **ha accesso solo in lettura**, ma ha fatto un **fork** del progetto nel suo namespace
  - Upstream è il riferimento (remote) al repository da cui è stato fatto il fork
  - Le modifiche vengono aggiunte ad un topic branch sul respository privato (origin)
  - Viene avviata una pull request per fare il **merge di due branch in repository diversi**
  - Tipicamente tra un topic branch in origin e il master in upstream

GitHub Flow



1. Creare un **topic branch** dal master
2. Fare dei commit per **migliorare** il progetto
3. Aprire una **Pull Request**
4. Gli sviluppatori **discutono** le modifiche, e potenzialmente aggiungono altri commit
5. Si fa il **pull** e il test delle modifiche
6. Si fa il **merge** del topic branch nel master

Tutorial

LibroGame

1.

Devi far presto, perché qualcosa ti dice che non è prudente indugiare presso le rovine fumanti del monastero. I Kraan, i mostri dalle nere ali, potrebbero tornare da un momento all'altro. Devi raggiungere Holmgard, la capitale di Sommerlund, e portare al Re la terribile notizia: tutti i cavalieri Ramas, salvo te, sono stati massacrati. Senza i Ramas alla testa del suo esercito, Sommerlund sarà alla mercè del suo antico nemico, i Signori delle Tenebre.

Trattenendo le lacrime, dai un ultimo saluto ai tuoi compagni uccisi. Dentro di te giuri che la loro morte sarà vendicata. Volti le spalle alle rovine e scendi con circospezione il ripido sentiero.

Ai piedi della collina il sentiero si biforca, ma entrambe le piste portano nel folto della foresta.

Se scegli il sentiero di destra, vai all'**85.**
Se scegli quello di sinistra, vai al **275.**
Se vuoi utilizzare l'Arte del Sesto Senso, vai al **141.**

2.

Mentre corri nel bosco sempre più fitto le grida dei Giak si sentono sempre meno. Ormai li hai quasi seminati, ma improvvisamente inciampi in un ramo basso e finisci dentro un cespuglio.

Scegli un numero della Tabella del Destino.

Se esce un numero fra 0 e 4, vai al **343.**
Se esce un numero da 5 a 9, vai al **276.**

Tratto da: "I signori delle tenebre" di Dever – Chalk
Traduzione Moss – Lughi. Edizioni E. Elle (1985)

RepoGame Tutorial

dmi-tutorato-ids / **esercitazione-repogame**

Watch 0 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Wiki | Security 0 | Insights | Settings

Branch: master | esercitazione-repogame / **README.md** | Find file | Copy path

afornaia Aggiunti dettagli sullo svolgimento dell'esercizio `5a03047` 2 minutes ago

1 contributor

43 lines (31 sloc) | 4.09 KB | Raw | Blame | History

Scriviamo un RepoGame: esercitazione sull'uso di GitHub

Avete mai giocato ad un gioco di ruolo cartaceo, in cui un master presenta agli altri giocatori al tavolo delle sfide da affrontare con i loro personaggi? Ecco: in un librogame il giocatore è uno solo, il lettore, e il master è il libro stesso!

I librogame sono libri-gioco nati negli anni ottanta. Invece di essere letti dall'inizio alla fine come dei normali libri, alla fine di ogni capitolo il lettore viene comunemente posto davanti ad un *bivio*, una scelta da compiere per determinare il prossimo capitolo da leggere e il destino del protagonista. Un esempio di bivio è il seguente: "se decidi di inoltrarti nella foresta, vai al capitolo 56; se dicidi invece di rimanere sulla strada principale, val al capitolo 38".

In questo esercizio costruiremo un **repogame**, un librogame scritto collaborativamente utilizzando un repository git. Ogni capitolo sarà costituito da una cartella numerata da 01 a 99, e il contenuto sarà solo un README.md contenente il titolo del capitolo, il testo e i collegamnti ai capitoli successivi. Ogni capitolo può avere uno o più possibili capitoli successivi, o anche nessuno, determinando così la fine dell'avventura dell'eroe con una gloriosa vittoria o una terribile disfatta.

Obiettivo

- Sfruttare la scrittura collaborativa di un libro, inevce che di un programma, per imparare i meccanismi fondamentali dello sviluppo con GitHub, ovvero:
  - gestire le issue per individuare i task da svolgere
  - crere un feature branch per soddisfare una issue
  - creare una pull request per far revisionare il codice/testo scritto
  - fare il merge di una pull request, unendo le modifiche al master
  - chiudere il feature branch
  - creare nuove issue

- Vedremo il caso di una merge request (due branch nello stesso repsitory) ma in maniera analga si può procedere con una pull request da un fork.

Passi da svolgere

1. Controllare le issue aperte con le richieste dei capitoli da scrivere, scegliendone uno. Supponiamo il capitolo *99*;
2. Creare un nuovo branch per la scrittura del capitolo, es. *capitolo-99*;
3. Creare una cartella nominata con il numero del capitolo e contenente solo un README.md: es. */99/README.md*;
4. Scrivere il contenuto del capitolo nel README.md prendendo spunto dai capitoli precedenti;
5. A meno che la storia non sia conclusa, aggiungere dei link a dei capitoli successivi (possibilmente non ancora esistenti);
6. Creare uno (o più) commit sul branch creato, completando il capitolo;
7. creare una *merge request*, ovvero una pull request per richiedere il merge del branch nel master;
8. Attendere una review da parte di un revisore;
9. Se richiesto, aggiungere eventuali altri *follow-up commit* sul branch per soddisfare le richieste del revisore;
10. Soddisfatte le richieste del revisore, la pull request verrà chiusa con un merge. Il nuovo capitolo è ora visibile sul master;
11. Chiudere la issue per il capitolo appena scritto (issue risolta);
12. Creare una issue per ognuno dei capitoli non ancora scritti ma indicati come successivi, chiedendo così ad altri di continuare la storia.

- Nota tutti questi passi possono essere eseguiti anche usando esclusivamente gli strumenti di GitHub, senza fare il clone. Ad ogni modo, è ovviamente possibile eseguire le operazioni di branch, commit e push partendo da un repositori locale.

Controllare una issue aperta

Creare capitolo 01 #1

Open **afornaia** opened this issue now · 0 comments

afornaia commented now Member

L'inizio del viaggio nel nostro avventuriero

Write Preview

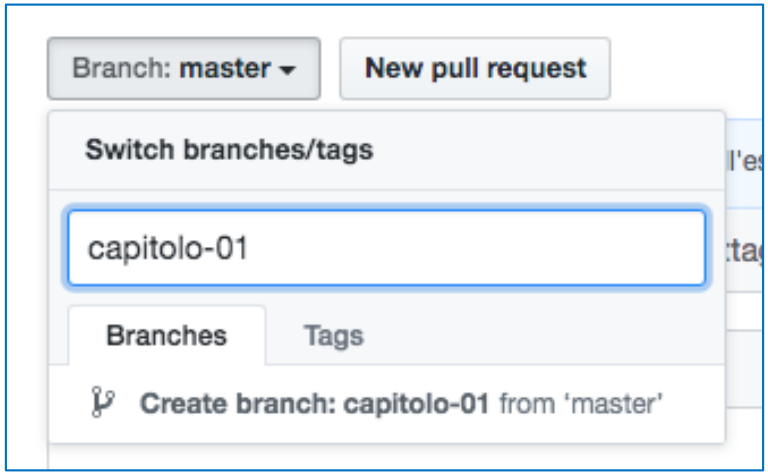
Leave a comment

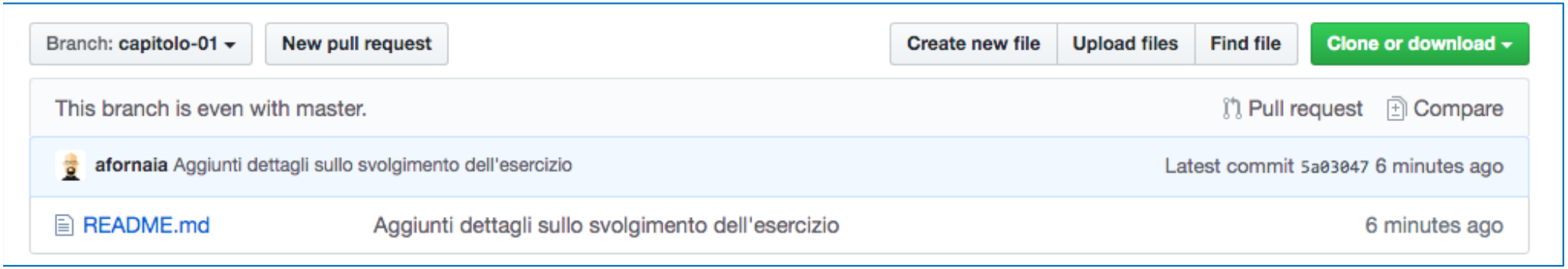
Attach files by dragging & dropping, selecting or pasting them.

Close issue Comment

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Creare un topic branch





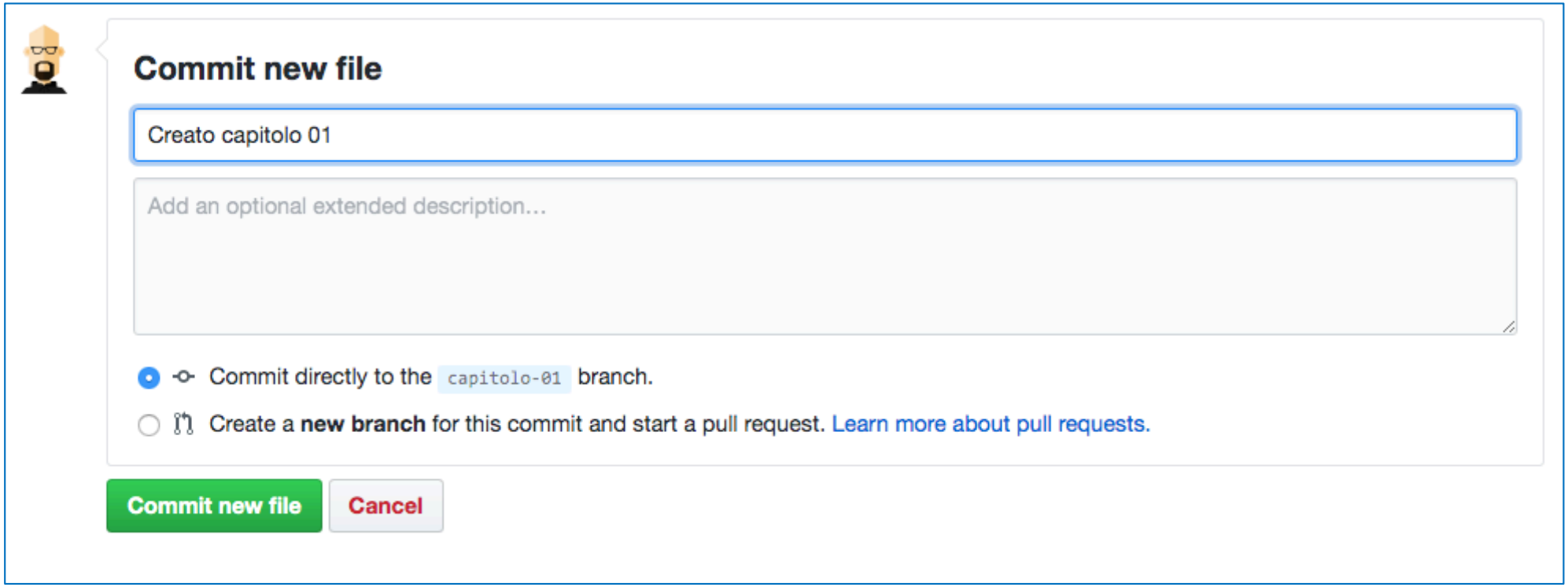
Aggiungere il file del capitolo

esercitazione-repogame / 01 / README.md Cancel

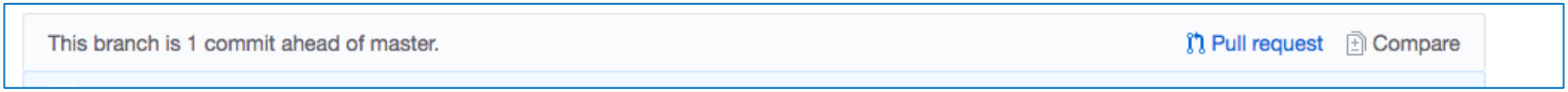
Edit new file | Preview | Spaces | 2 | No wrap

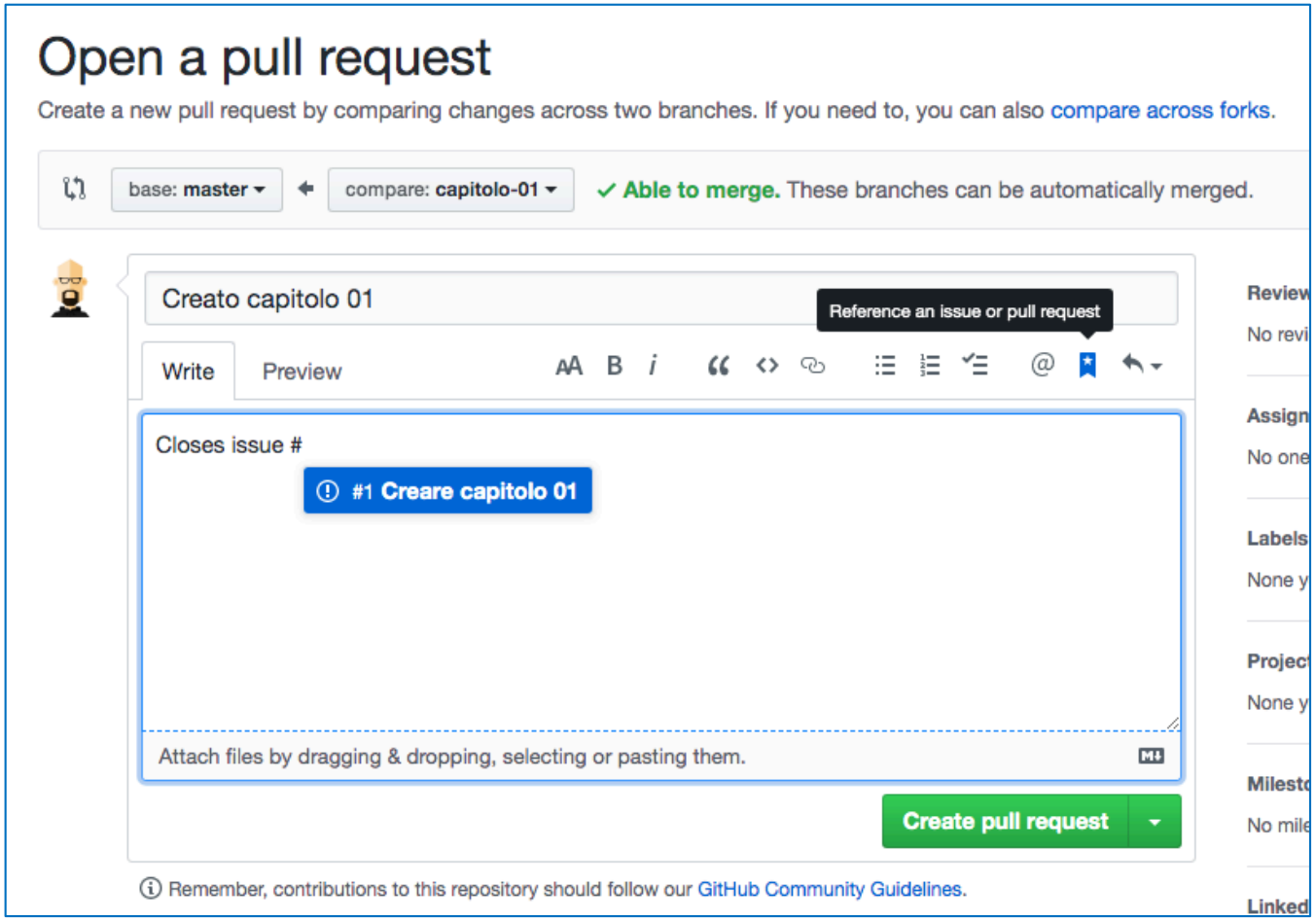
```
# 01

Sei a cavallo da ore ormai sulla strada che porta ad Erudun.
Come ti aspettavi c'è poca gente in giro. La guerra tra i due regni dura ormai da un anno,
e l'esercito pattuglia spesso le strade. Sono due giorni che non ricevi una paga decente, e
hai deciso di raggiungere la città più vicina sperando di trovare un contratto di qualsiasi tipo.

Decidi di fermarti ad una fonte per far bere e riposare il tuo cavallo, quando noti un piccolo elfo,
rannicchiato di spalle poco distante. Sembra molto giovane. Sta piangendo.
Forse si è smarrito? Magari qualcuno lo sta cercando disperatamente!
Potrebbe essere una buona occasione per guadagnarsi una ricompensa facile!

- Se decidi di avvicinarti al piccolo elfo, vai al capitolo [**09**](/09/README.md);
- Se decidi invece di proseguire per la tua strada, vai al capitolo [**22**](/10/README.md).
```

Fare un commit sul topic branch



Avviare una pull request (merge)

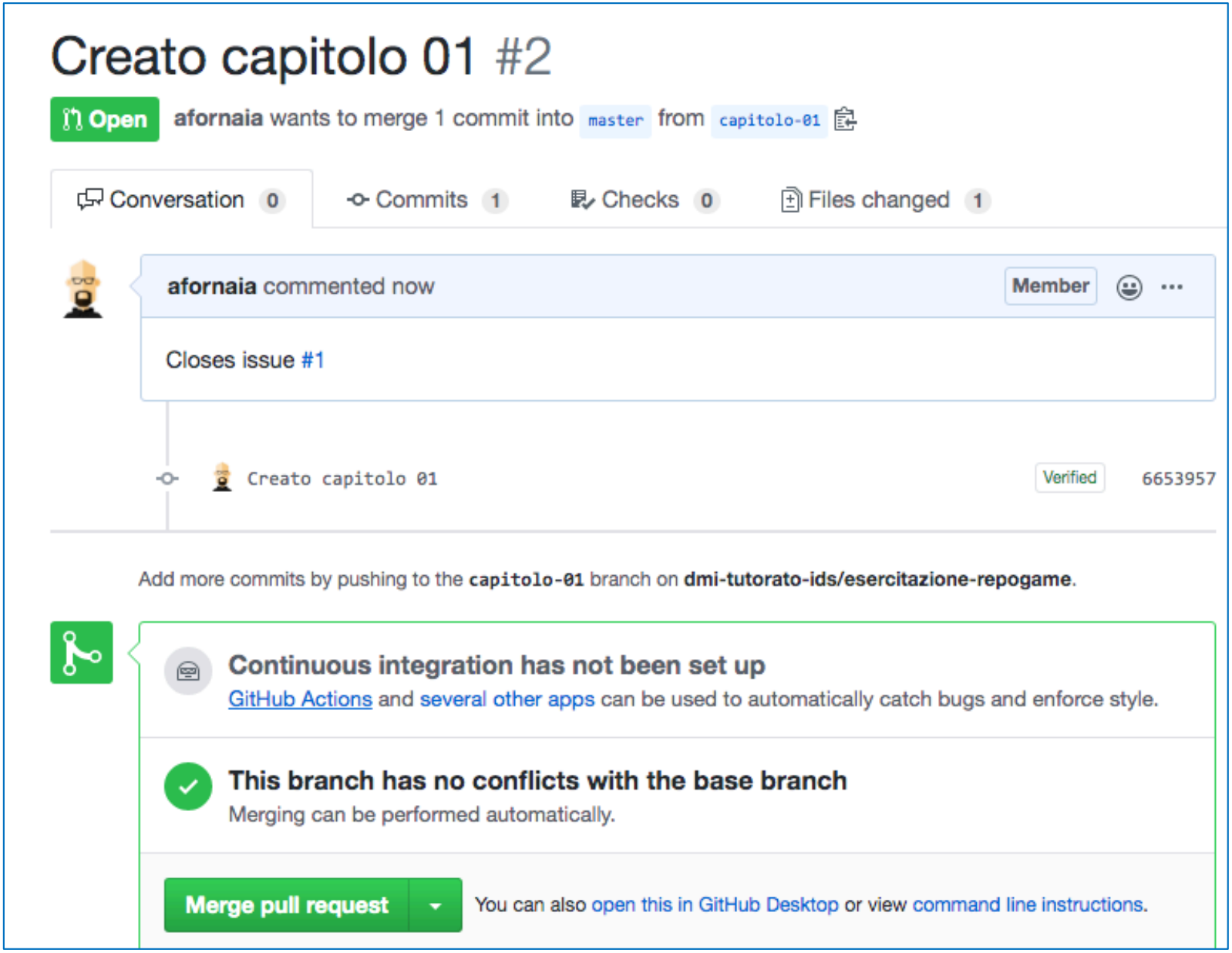




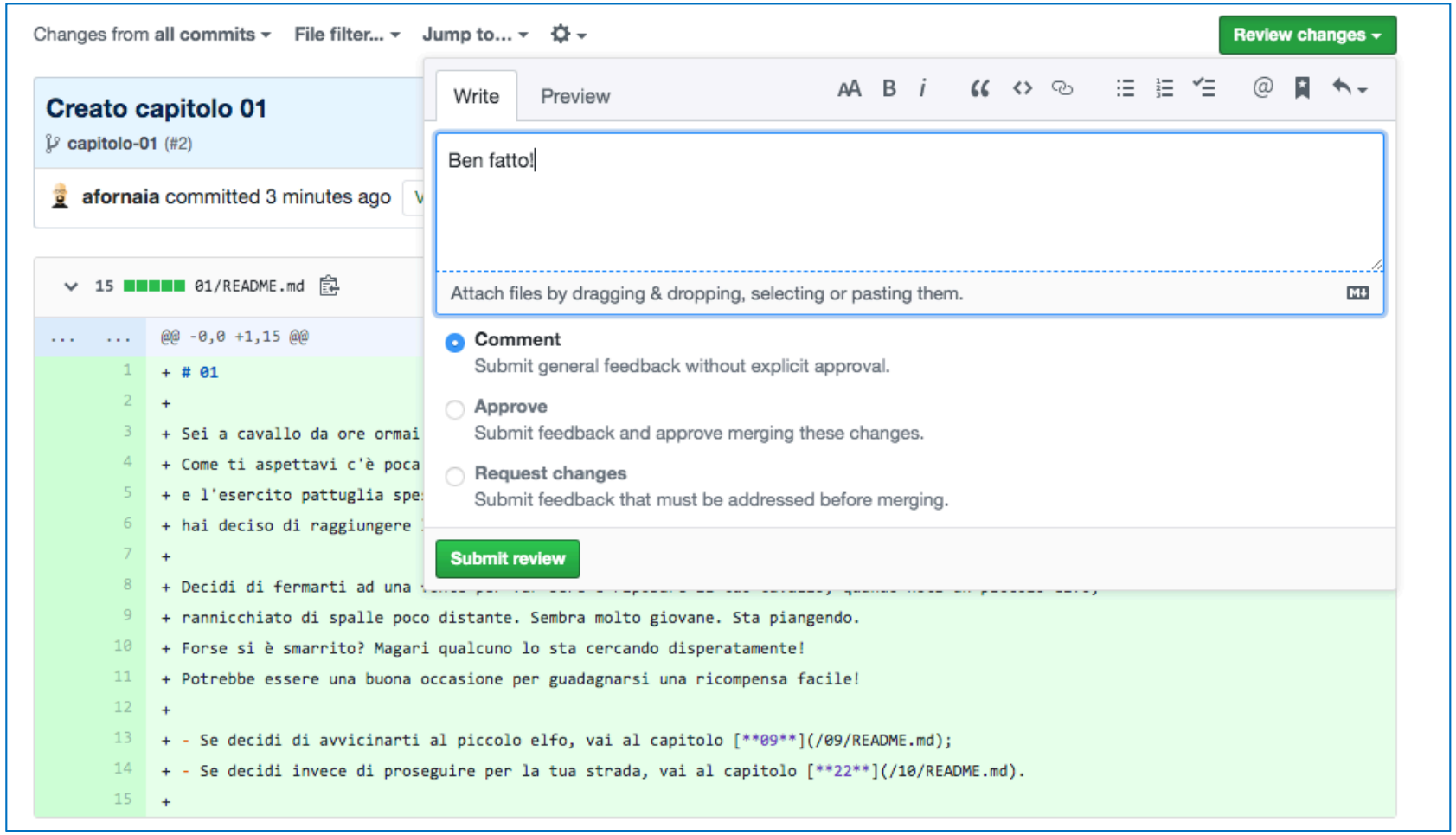
Consigliabile aggiungere un riferimento alla issue che il topic branch risolve (es. #1).

Apparirà nella chat della issue come riferimento

Attendere una review



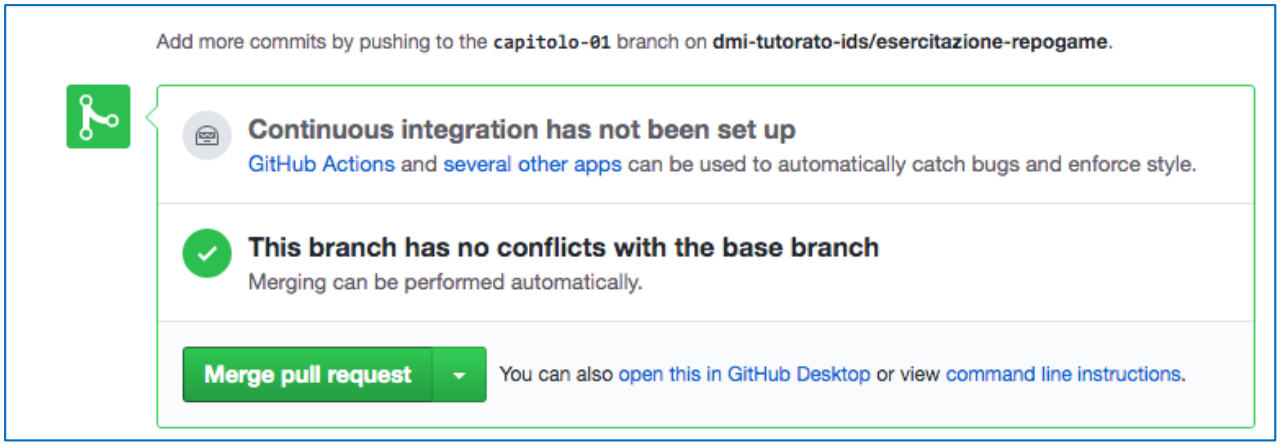
Fare una review



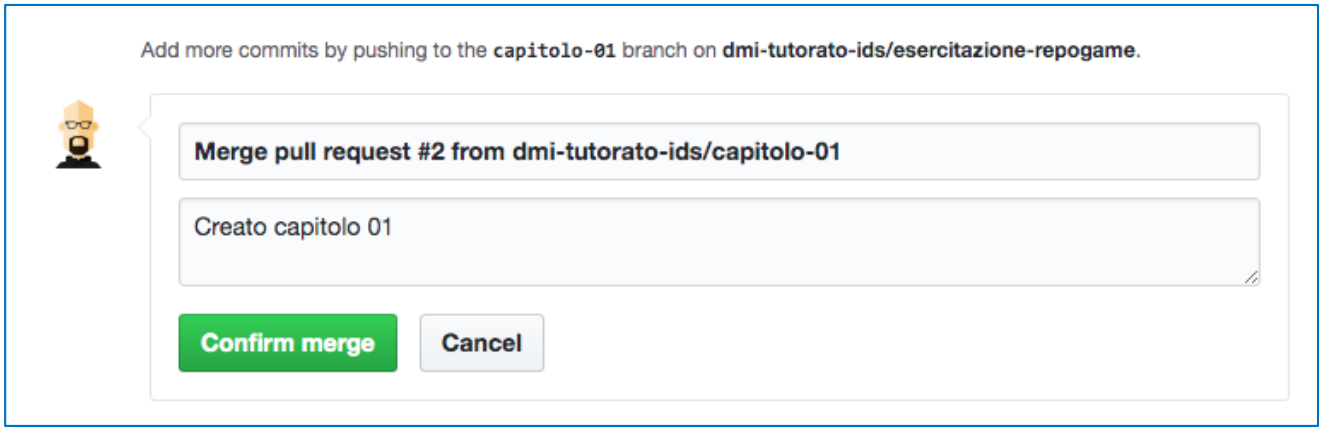
Comment: solo un commento di revisione
Approve: approva il merge dellemodifiche apportate (tipicamente solo il maintainer)
Request chages: richiedi modifiche (follow-up commit) prima di fare il merge

Merge della pull request

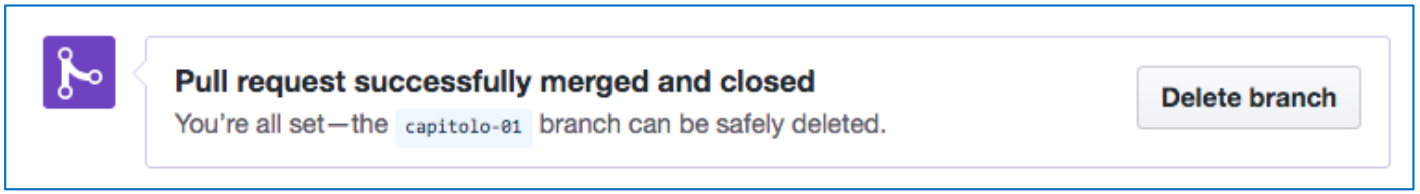
1



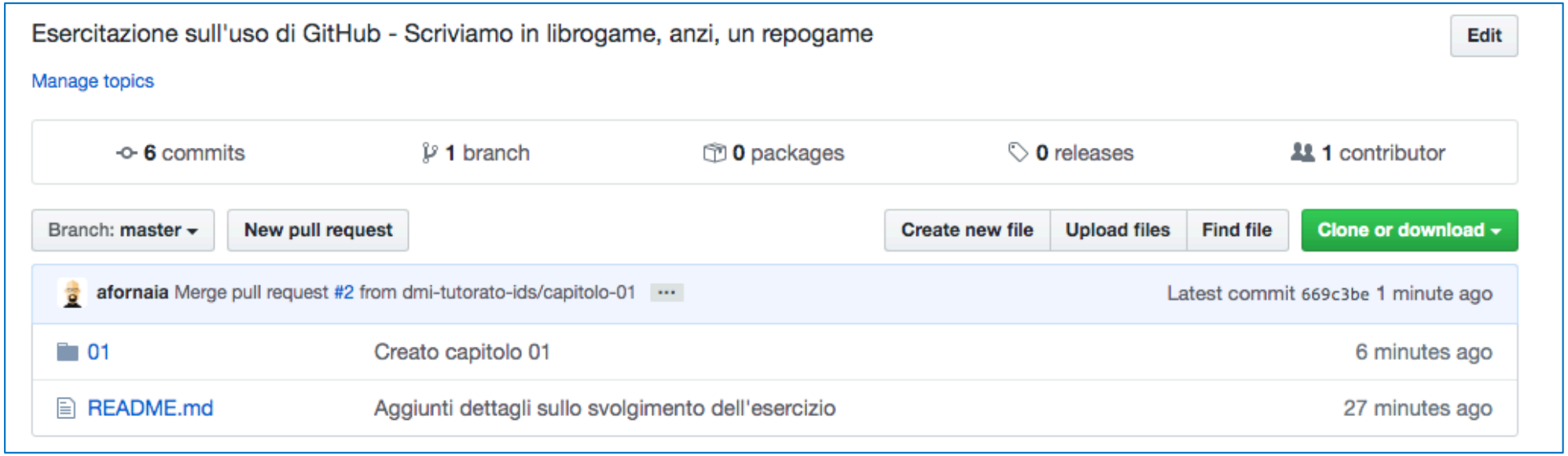
2



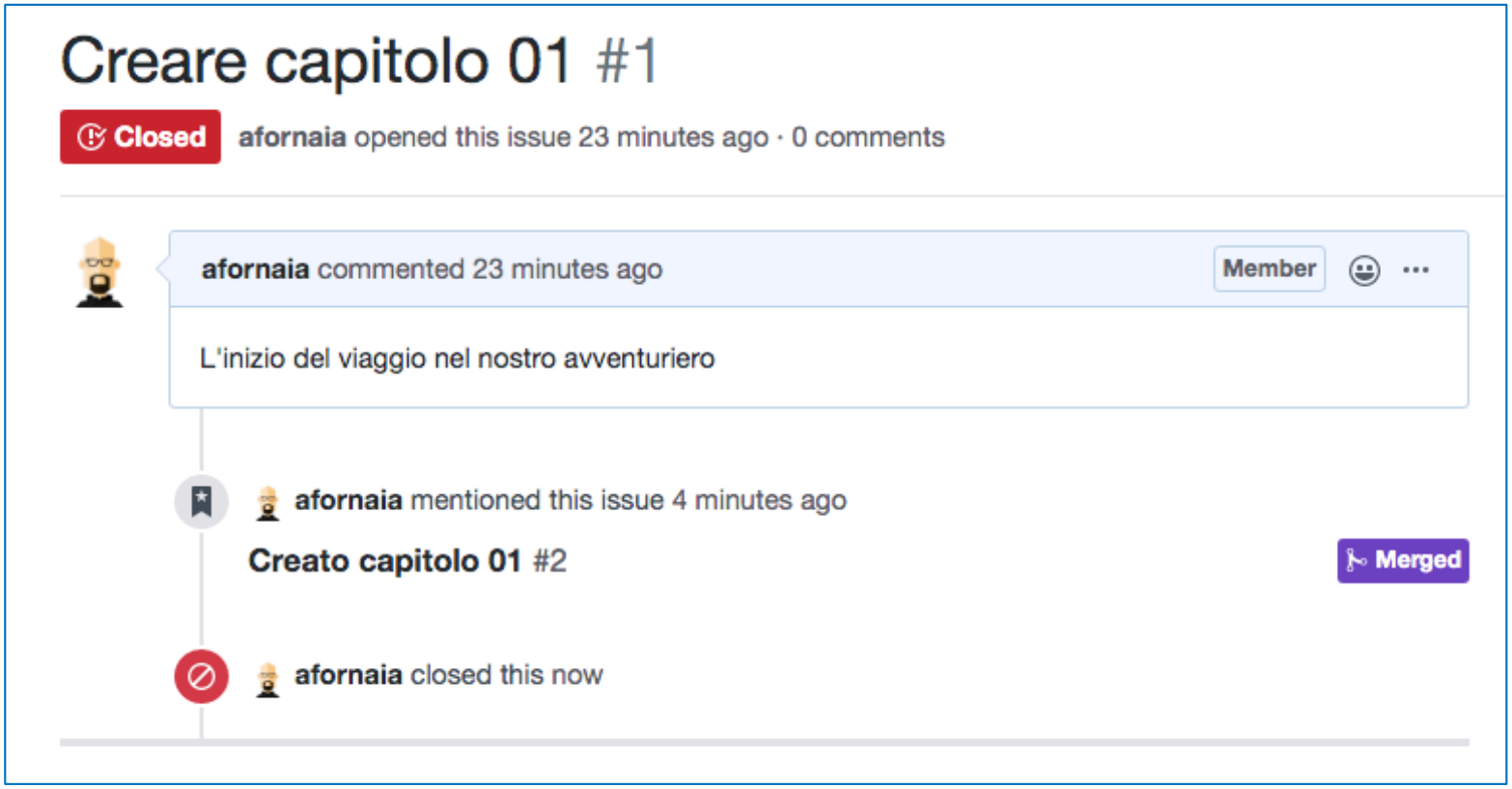
3



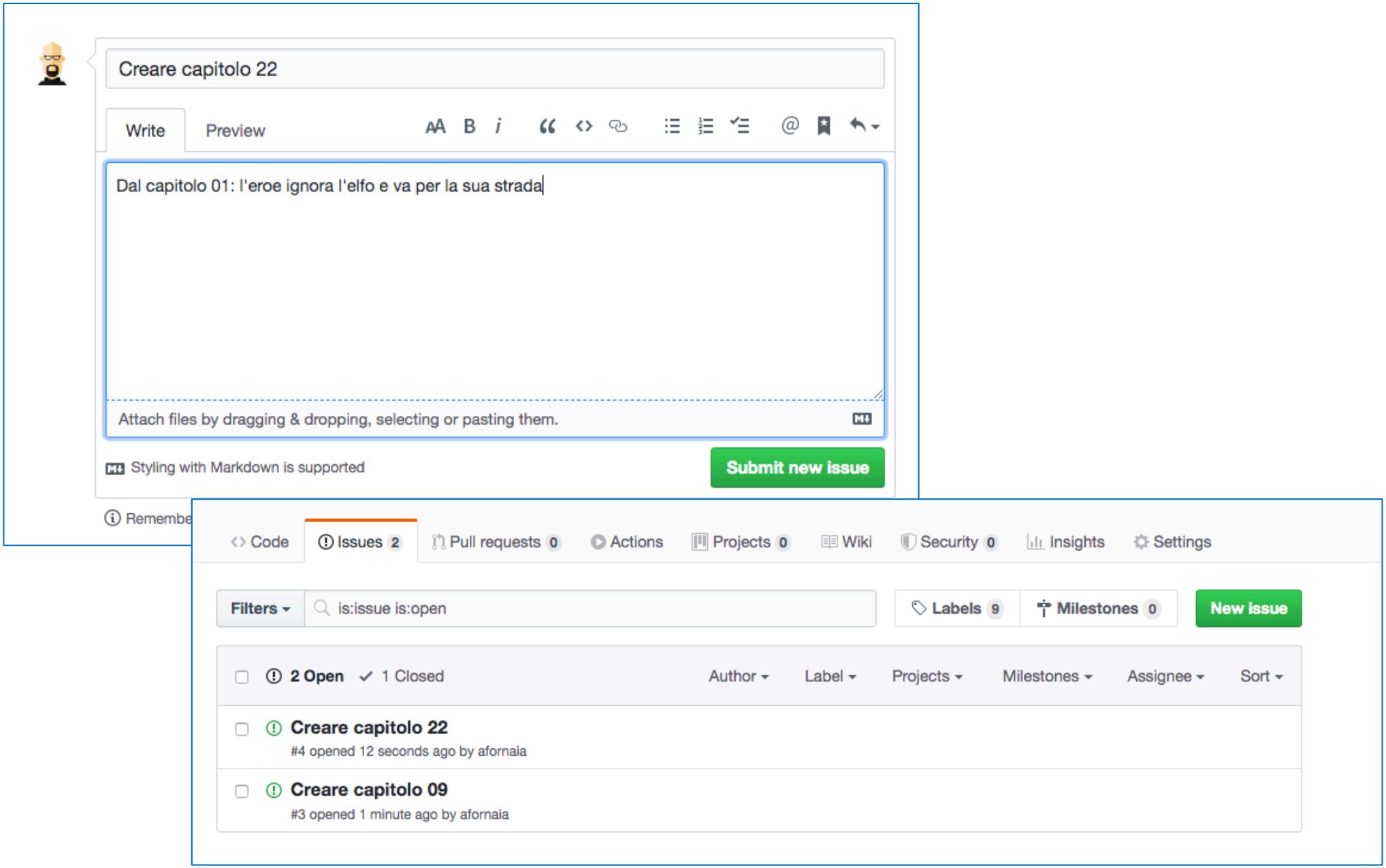
Le modifiche sono ora sul master



Chiudere la issue risolta



Aprire nuove issue



That’s all! (for now)