Eserciziario - Sistemi di versionamento: Git

Daril Marra

Sommario

Disclaimer

ATTENZIONE: dato che word non è un editor di testo e modifica i caratteri scritti (ad esempio gli apici di word “ ” ‘ ‘ sono caratteri diversi dai normali apici…) i comandi git o il codice scritto in files word come questo **NON VANNO MAI COPIA-INCOLLATI**, ma sempre **RISCRITTI A MANO** nella bash o nell’editor di codice.

Installazione

Scarichiamo git al seguente link: https://git-scm.com/downloads ed installiamolo

Repository

Creiamo una nuova cartella e apriamo la bash di git in essa: in windows, clicchiamo all’interno della cartella in uno spazio vuoto col tasto destro e selezionamo la voce “Git Bash Here”.

Inizializziamo la repository git col comando **git init**

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0
$ git init
Initialized empty Git repository in C:/workspaces/academy/lesson-0/.git/
```

Ora colleghiamo la nostra repo locale con una remota creata da me per l’occasione, accessibile al seguente link: https://github.com/daril-marra/gft-academy-0.git

git remote add origin https://github.com/daril-marra/gft-academy-0.git

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (master)
$ git remote add origin https://github.com/daril-marra/gft-academy-0.git
```

Commit e Stato dei files

Creiamo qualche file all’interno della cartella e proviamo a lanciare **git status** per verificare il loro stato:

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        home/

nothing added to commit but untracked files present (use "git add" to track)
```

Vediamo che sono *untracked*, ovvero non ancora tracciati da git, aggiungiamoli entrambi quindi: **git add .**

Ora **git status** restituisce:

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   home/index.html
```

I files sono quindi nella *staging area*, pronti per essere committati, proviamo però a modificare un file

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   home/index.html

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:   file1.txt
```

git status ci mostra i files nella staging area, ma ci avverte anche che le modifiche fatte a file1.txt dopo aver lanciato il comando add non sono ancora state aggiunte alla *staging area* e sono quindi ancora nel *working tree*. Lanciamo un altro **git add .** per aggiungere anche le ultime modifiche al commit che stiamo preparando

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   home/index.html
```

Ora lanciamo il comando **git commit -m "primo commit"**

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (master)
$ git commit -m "primo commit"
[master (root-commit) f35f38e] primo commit
 2 files changed, 16 insertions(+)
 create mode 100644 file1.txt
 create mode 100644 home/index.html
```

Branch e Checkout

Al momento dell'init git ci ha messi sul branch di default master, creiamo ora un nostro branch di sviluppo utilizzando il nostro nome come identificativo, col comando **git branch *daril-marra***

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (master)
$ git branch daril-marra

a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (master)
$
```

Come vediamo però dalla scritta in azzurro nonostante abbiamo creato il nuovo branch, il branch corrente è ancora master, verifichiamo con **git branch --all**

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (master)
$ git branch --all
  daril-marra
* master
```

Infatti vediamo il nuovo branch nella lista ma siamo ancora su master, il branch corrente è evidenziato in verde dalla bash di git.

Facciamo allora un checkout del nuovo branch **git checkout *daril-marra***

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (master)
$ git checkout daril-marra
Switched to branch 'daril-marra'

a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$
```

Ora verifichiamo i commit presenti nei branch con **git log**

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git log
commit f35f38ed0b0fbfe2fc39debeac531b9471916f7f (HEAD -> daril-marra, master)
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 09:08:53 2022 +0100

    primo commit
```

Vediamo che il commit creato su master è stato riportato nel branch corrente *daril-marra*, questo perché il comando branch lo crea basandosi sul branch corrente al momento del lancio del comando, ha quindi copiato il contenuto di master nel branch creato *daril-marra*.

Possiamo quindi cancellare il branch master senza perdere dati: **git branch -D *master***

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git branch -D master
Deleted branch master (was f35f38e).
```

Fetch

Finora abbiamo lavorato solo sulla repository locale e non abbiamo ancora scaricato i branch remoti: facciamolo utilizzando il comando **git fetch**

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git fetch
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 248 bytes | 13.00 KiB/s, done.
From https://github.com/daril-marra/gft-academy-0
 * [new branch]      main       -> origin/main
```

Abbiamo scaricato tutto il contenuto della repository remota: in questo caso c'era un solo branch chiamato *main* che è il branch di default.

Gurdando i branch ora presenti nella nostra repository vediamo che al branch locale *daril-marra* si è aggiunto il branch *main*, essendo remoto è scritto in rosso e prefisso dal nome del remote *origin*:

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git branch --all
* daril-marra
  remotes/origin/main
```

Push

Proviamo ora ad eseguire l'upload del nostro branch utilizzando il comando **git push -u origin *daril-marra***

Il comando di push richiede la creazione di un account github e l'abilitazione alla cartella remota per l'account.

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git push --set-upstream origin daril-marra
git: 'credential-manager' is not a git command. See 'git --help'.

The most similar command is
        credential-manager-core
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 737 bytes | 737.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'daril-marra' on GitHub by visiting:
remote:      https://github.com/daril-marra/gft-academy-0/pull/new/daril-marra
remote:
To https://github.com/daril-marra/gft-academy-0.git
 * [new branch]      daril-marra -> daril-marra
branch 'daril-marra' set up to track 'origin/daril-marra'.
```

Col comando **git branch --all** possiamo vedere che è stato creato il branch remoto daril-marra e col comando **git branch -vv** possiamo vedere che il nostro branch locale è collegato a quello remoto (in blu scuro)

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git branch --all
* daril-marra
  remotes/origin/daril-marra
  remotes/origin/main

a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git branch -vv
* daril-marra f35f38e [origin/daril-marra] primo commit
```

Merge

A partire dal nostro branch di sviluppo *daril-marra* creiamo un nuovo branch *daril-marra-fork* e spostiamoci su di esso: **git branch *daril-marra-fork* + git checkout *daril-marra-fork***.

Modifichiamo qualche file e prepariamo un secondo commit sul branch *daril-marra-fork*: **git add . + git commit -m "secondo commit"**.

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra-fork)
$ git log
commit 38282bd7cae8dfe940c1805d8f2cc48e8f6d6f1e (HEAD -> daril-marra-fork)
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 15:12:52 2022 +0100

    secondo commit

commit f35f38ed0b0fbfe2fc39debeac531b9471916f7f (origin/daril-marra, daril-marra)
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 09:08:53 2022 +0100

    primo commit
```

Con un **git log** vediamo che *daril-marra-fork* punta a questo secondo commit mentre invece il branch *daril-marra* non essendo stato modificato punta ancora al primo commit.

Per unire il branch locale *daril-marra-fork* al branch locale *daril-marra*, portando i commit di *-fork* al branch originale utilizziamo il comando merge. Prima ci portiamo sul branch di destinazione con **git checkout *daril-marra*** e poi riportiamo i commit del branch sorgente utilizzando **git merge *daril-marra-fork***

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git merge daril-marra-fork
Updating f35f38e..38282bd
Fast-forward
 file1.txt | 9 +++++++--
 1 file changed, 7 insertions(+), 2 deletions(-)
```

Ed ecco che ora **git branch -vv** ci dice che entrambi i branch puntano al secondo commit.

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git branch -vv
* daril-marra      38282bd [origin/daril-marra: ahead 1] secondo commit
  daril-marra-fork 38282bd secondo commit
```

Pull

Ora vogliamo incorporare nel nostro branch locale gli sviluppo di un branch remoto, creiamo quindi prima un branch remoto da cui eseguire un pull. Spostiamoci in *daril-marra-fork* **git checkout *daril-marra-fork*** , modifichiamo qualcosa e committiamolo **git add . + git commit -m "terzo commit"** e pushiamo il branch **git push -u origin *daril-marra-fork*** .

Infine per incorporare il terzo commit possiamo pullare il branch remoto *daril-marra-fork* posizionandoci prima nel branch di destinazione *daril-marra* **git checkout *daril-marra*** e poi pullando **git pull origin *daril-marra-fork*** (notare la mancanza di -u perché vogliamo tenere collegato il branch remoto *daril-marra* invece

di *daril-marra-fork*)

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git pull origin daril-marra-fork
From https://github.com/daril-marra/gft-academy-0
 * branch            daril-marra-fork -> FETCH_HEAD
Updating 38282bd..a3cc75d
Fast-forward
 home/index.html | 4 ++++
 1 file changed, 4 insertions(+)

a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git log
commit a3cc75d7fd1a29fcfa9b818bc86f87b663b65ad1 (HEAD -> daril-marra, origin/daril-marra-fo
rk, daril-marra-fork)
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 16:13:18 2022 +0100

    terzo commit

commit 38282bd7cae8dfe940c1805d8f2cc48e8f6d6f1e
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 15:12:52 2022 +0100

    secondo commit

commit f35f38ed0b0fbfe2fc39debeac531b9471916f7f (origin/daril-marra)
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 09:08:53 2022 +0100

    primo commit
```

Ecco che il nostro branch locale *daril-marra* è ora allineato al branch remoto *daril-marra-fork*. Questo procedimento è molto simile al merge (difatti il pull è composto da fetch + merge) la differenza sta nel fatto che il branch sorgente è remoto e non locale e vedremmo una differenza se i due *daril-marra-fork* remoto e locale fossero disallineati.

Conflitti

Proviamo a modificare la stessa riga di un file in modo diverso in entrambi i branch, creando un commit diverso per ogni branch.

Dopodichè proviamo a mergiarli uno dentro all'altro, abbiamo un conflitto!

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra-fork)
$ git merge daril-marra
Auto-merging home/index.html
CONFLICT (content): Merge conflict in home/index.html
Automatic merge failed; fix conflicts and then commit the result.

a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra-fork|MERGING)
$
```

Git ci dice anche che siamo ancora nel bel mezzo del merge e in questo stato ci è proibito eseguire alcune operazioni quali il checkout prima di risolvere i conflitti.

Aprendo il file vediamo che git ha scritto entrambe le righe in conflitto, delimitando il conflitto con i caratteri <<<<<<<, =======, >>>>>>>

```
7   <body>
8       <h1>Git rocks!</h1>
9       <p>Did you know git means idiot in british slang? <br />
10          When naming it, Torvalds said: "I'm an egotistical bastard, and I name all my projects after myself. First
11          'Linux', now 'git'."
12      </p>
    Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
13  <<<<<<< HEAD (Current Change)
14      <p>What a git!</p>
15  =======
16      <p>What a role model!</p>
17  >>>>>>> daril-marra (Incoming Change)
18  </body>    a38v [8 hours ago] • primo commit      You, 8 hours ago • primo commit
```

Possiamo modificare il file come vogliamo, scegliendo una delle due versioni oppure entrambe o anche modificando ulteriormente il file. In ogni caso dovremo cancellare tutti i delimitatori <<<<<<<, =======, >>>>>>> in modo da far capire a git che abbiamo risolto quel conflitto.

Fatto ciò, possiamo procedere a creare un commit “di merge” con i seguenti comandi **git add . + git commit -m “merged daril-marra in daril-marra-fork”**

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra-fork|MERGING)
$ git add .

a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra-fork|MERGING)
$ git commit -m "merged daril-marra in daril-marra-fork"
[daril-marra-fork 90b788f] merged daril-marra in daril-marra-fork
```

Ora che il merge è completato il branch contiene i commit di entrambi i branch sorgente e destinazione con in più il commit di merge

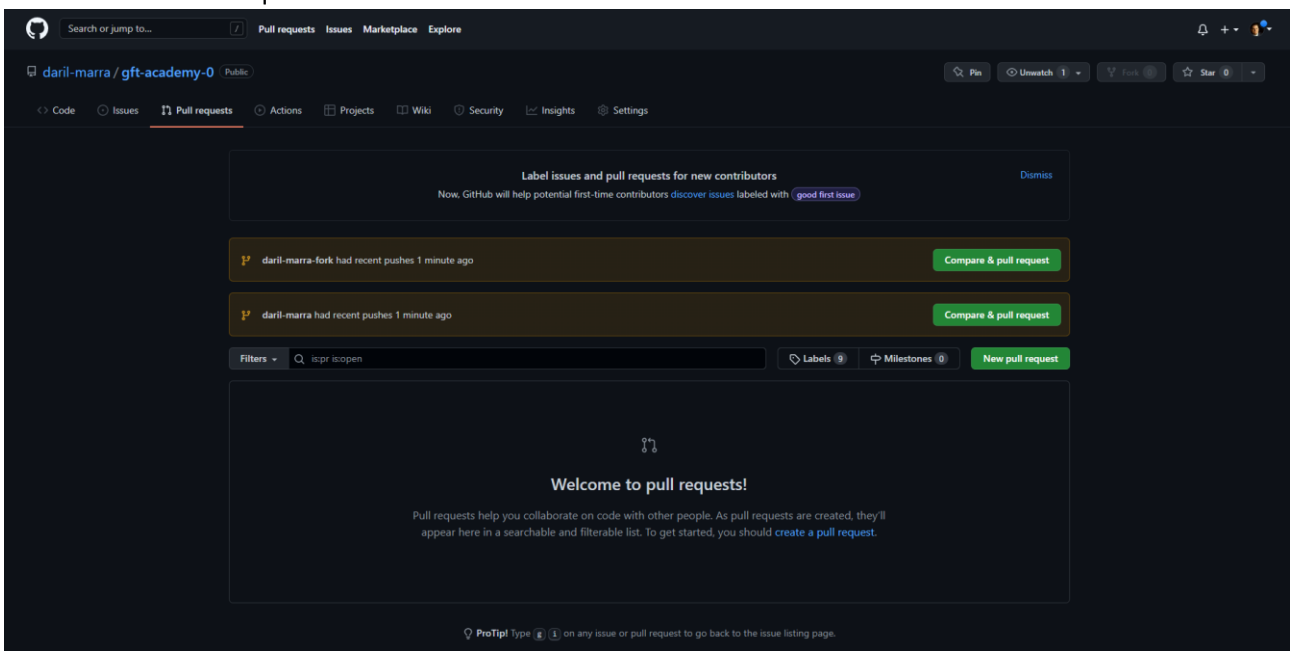
```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra-fork)
$ git log
commit 90b788f2a31309a899db341cf08513c324003227 (HEAD -> daril-marra-fork)
Merge: 47c1c1c a029cc0
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 17:15:14 2022 +0100

    merged daril-marra in daril-marra-fork

commit 47c1c1cc5934f5c18fead83971161b8b8a578de3
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 16:54:25 2022 +0100

    Linus is a git

commit a029cc05b067ef2ed85a86d5bb416767f27769ef (daril-marra)
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 16:53:31 2022 +0100

    Linus is a role model

commit a3cc75d7fd1a29fcfa9b818bc86f87b663b65ad1 (origin/daril-marra-fork)
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 16:13:18 2022 +0100

    terzo commit

commit 38282bd7cae8dfe940c1805d8f2cc48e8f6d6f1e
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 15:12:52 2022 +0100

    secondo commit

commit f35f38ed0b0fbfe2fc39debeac531b9471916f7f (origin/daril-marra)
Author: a38v <daril.marra@gft.com>
Date:   Wed Feb 9 09:08:53 2022 +0100

    primo commit
```

Merge Request/ Pull Request

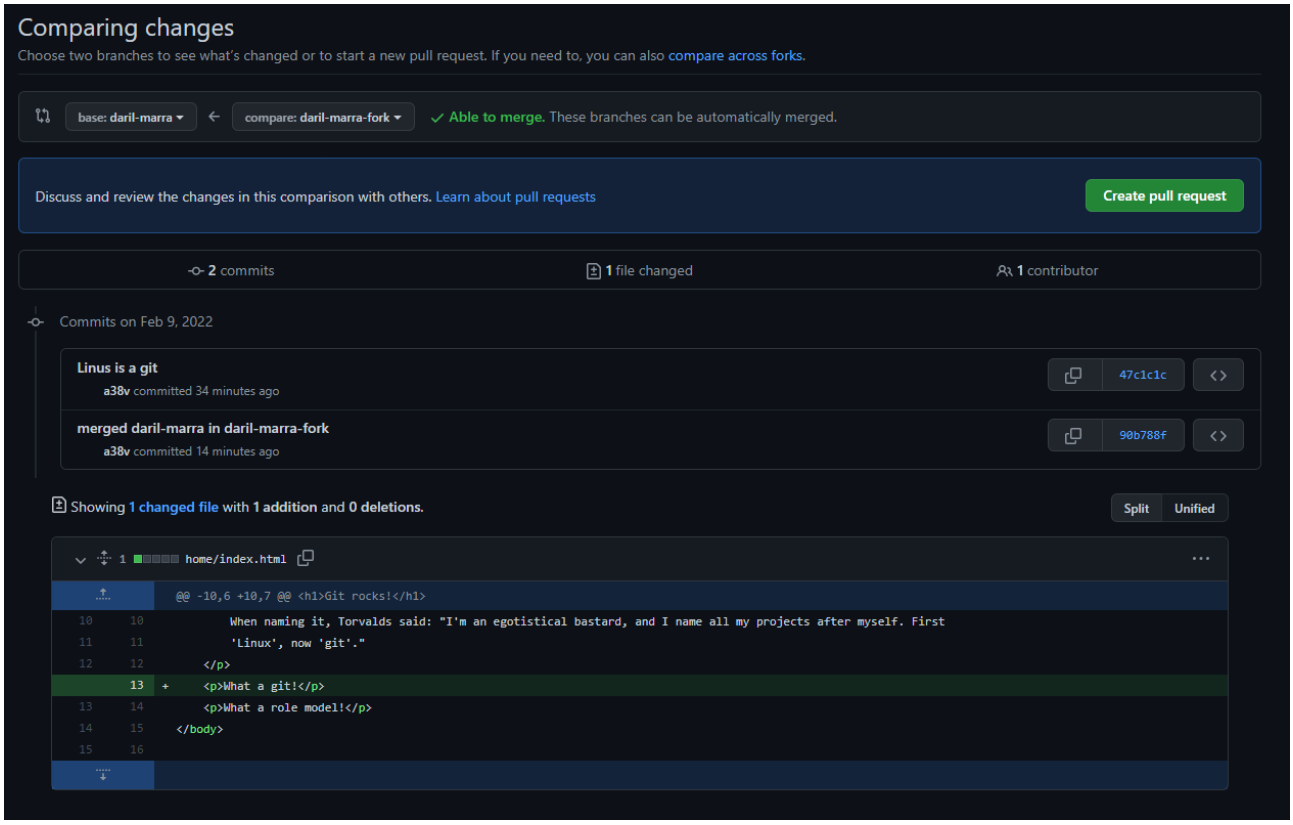
Pushiamo entrambi i branch per poter creare una merge request in remoto su github.

Dopodichè andiamo sulla pagina della repository GitHub https://github.com/daril-marra/gft-academy-0.git

nella sezione “Pull requests”



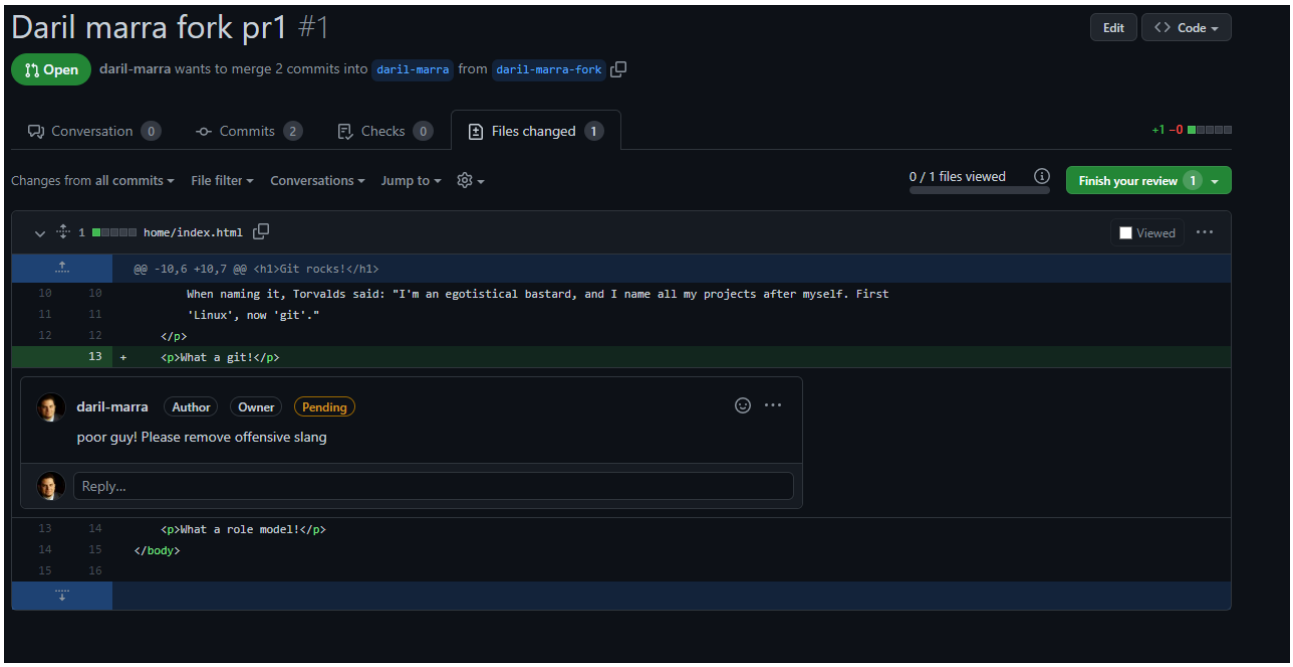
Clicchiamo su “New Pull Request” ed effettuiamo un merge opposto a prima in modo da allineare i due branch



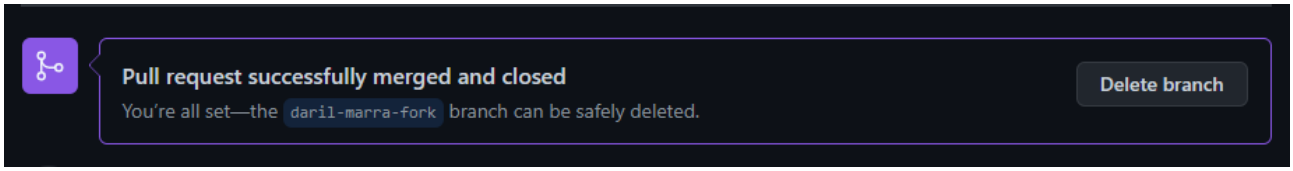
Ora non ci sono più conflitti perché sono già risolti con il commit di merge: creiamo la pull request cliccando su “create pull request” e dandogli un titolo ed una descrizione.

Ora i reviewer che dovranno fare la code review possono accedere alla pull request e verificare i commit, il files diff e commentare il codice in modo da chiedere delucidazione al creatore o fornirgli indicazioni per

modificare il codice.



Il creatore potrà modificare il codice committando e pushando sul branch sorgente (*daril-marra-fork* in questo caso) e infine la PR può essere approvata e i branch mergiati.

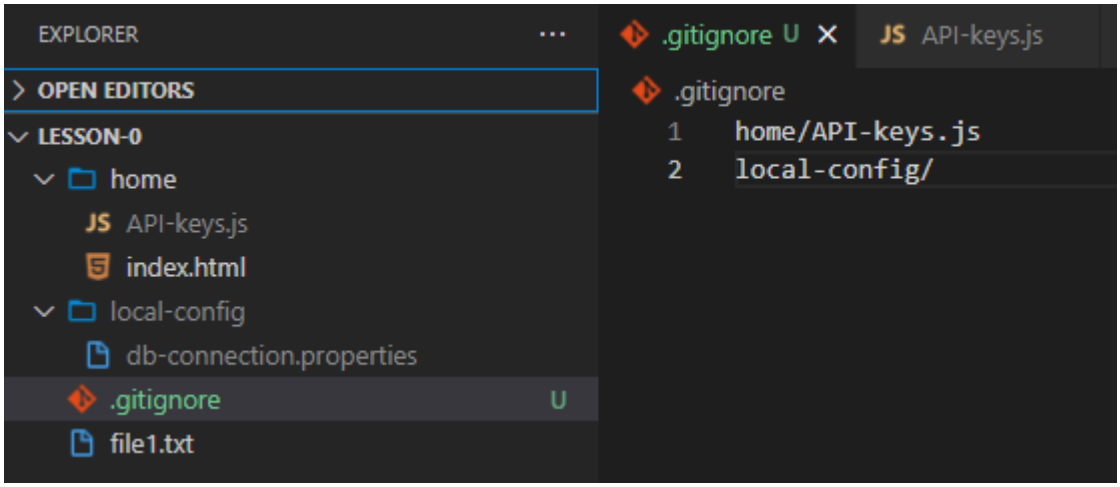


.gitignore

Dobbiamo creare dei files di configurazioni con le chiavi API e la connessione al database, che non vogliamo assolutamente condividere. Con **git status** vediamo che questi files sono all'interno del working tree, anche se ancora untracked, ma sarebbe ben troppo facile in un momento di distrazione utilizzare git add . e committarli per errore.

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git status
On branch daril-marra
Your branch is up to date with 'origin/daril-marra'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        home/API-keys.js
        local-config/

nothing added to commit but untracked files present (use "git add" to track)
```

Creiamo quindi un file .gitignore nella cartella root del progetto e aggiungiamo il file home/API-keys.js e l'intera cartella local-config/



La nostra IDE VS Code già ci fa capire, grazie al colore grigio dei files, che essi sono ignorati, verifichiamolo lanciando **git status**

```
a38v@ITPC009937 MINGW64 /c/workspaces/academy/lesson-0 (daril-marra)
$ git status
On branch daril-marra
Your branch is up to date with 'origin/daril-marra'.

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

nothing added to commit but untracked files present (use "git add" to track)
```

Ecco che i files che volevano ignorare sono spariti e possiamo committare il .gitignore in tutta sicurezza con **git add . + git commit -m "added .gitignore"** di modo che anche gli altri sviluppatori, dopo un **push** nostro e un **pull** loro del branch possano ignorare queste configurazioni, creandosi dei files contenenti la loro API key e i loro dati di connessione al proprio database.

Workflow di sviluppo – esercizio finale

Proviamo ora a simulare le operazioni solitamente fatte durante uno sviluppo con un esercizio.

Ecco la traccia:

1. Assicurarsi di essere collegati alla repo https://github.com/daril-marra/gft-academy-0.git ed eseguire un fetch
2. Creare un branch locale chiamato *dev* e posizionarsi su di esso, sarà il branch di riferimento
3. Resettarlo al branch *origin/dev*
4. A partire dal branch di riferimento, creare e posizionarsi su un branch locale di sviluppo chiamato *user/nome-cognome*
5. Sbizzarrirsi nello sviluppo: modifica file esistenti, creazione di nuovi file e cancellazione di file inutilizzati (senza perderci troppo tempo). Modificare il file *file1.txt*
6. Preparare il commit ed eseguirlo con un messaggio descrittivo
7. Push del branch di sviluppo (*user/nome-cognome*)
8. Pull del branch di riferimento (NON usare **-u** nel comando di pull)
9. Se si riscontrano conflitti: risoluzione di essi e commit di merge
10. Push del branch di sviluppo, ora allineato al branch di riferimento dopo lo sviluppo
11. Creazione della Pull Request su GitHub: quest'ultima parte la facciamo assieme