

GIT

Introduzione

Perché GIT?

- Ogni azienda sta lavorando con GIT ed è un requisito per il 99% dei lavori.
- Git è un DVCS (Distributed Version Control System)
- Git permette di LAVORARE IN TEAM
- Git consente di tenere traccia della cronologia completa dei progetti
- Git consente di salvare i vostri preziosi progetti!

Senza Git

RICCARDO'S CODE



v 1.0.0



v 1.0.1



v 1.0.2

DIEGO'S CODE

v 1.0.0



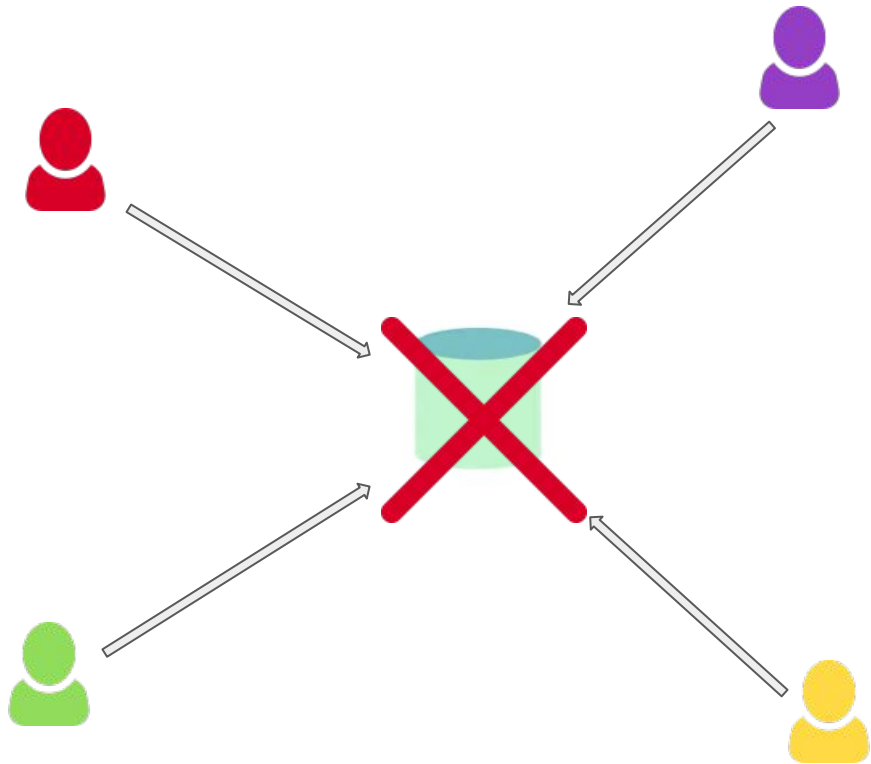
v 1.0.1



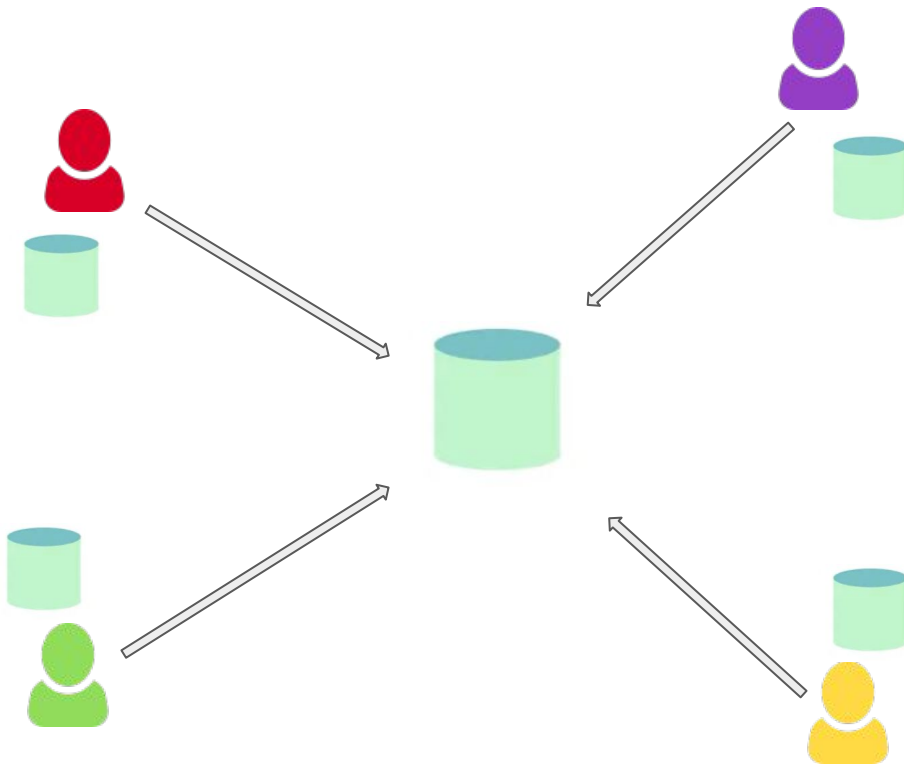
v 1.0.2



Senza Git



Con Git



Come usare Git?

- Da riga di comando
- Da editor di codice
- Con le GUI di Git (GitHub Desktop, GitKraken, SourceTree, GitFork, ...)

Verificare l'installazione di Git

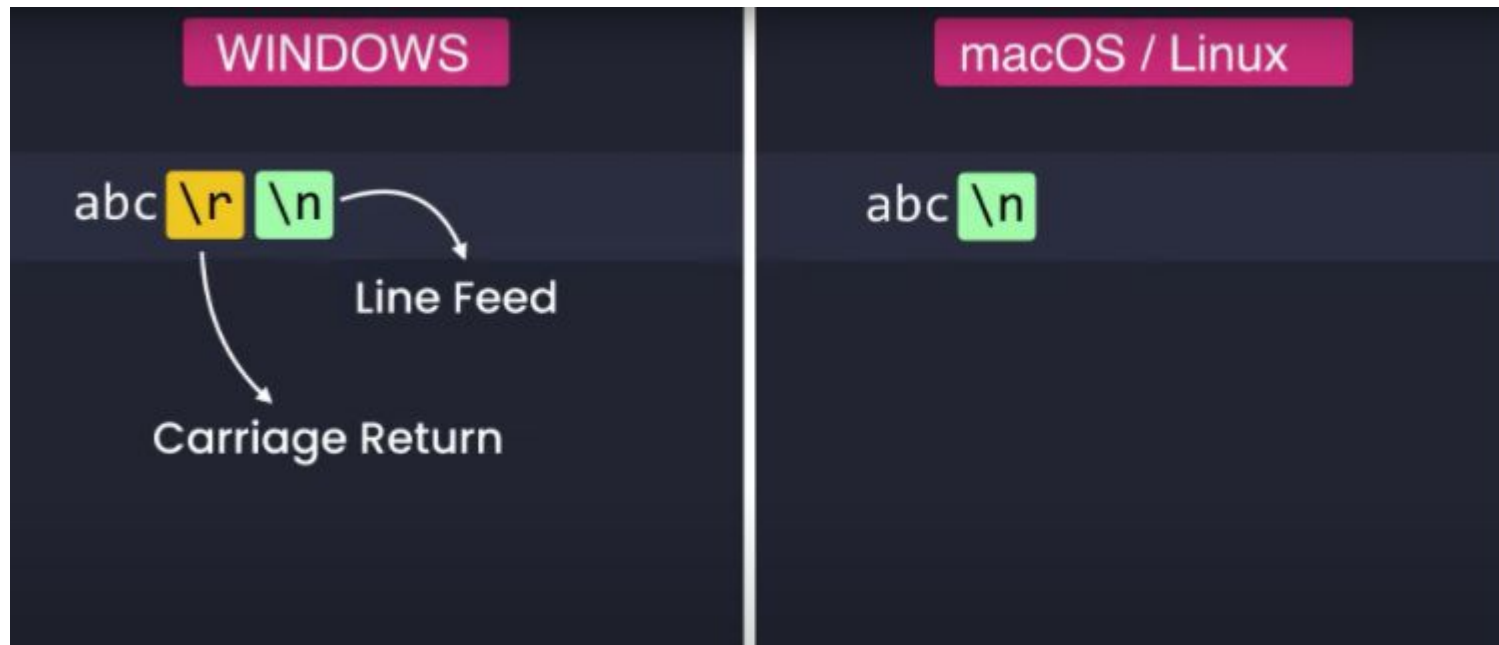
- **Mac** → command + space: apre il terminale
- **Windows** → windows + r: cmd
- **Linux** → bash

```
$ git --version
```


Git config

- `$ git config --global user.name "Your Name"`
- `$ git config --global user.email "useyour@email.com"`

Git config



Flusso Git



Staging Area

- Area specifica in cui portare tutti i file che ci si propone di salvare
- Qui è possibile esaminare le modifiche
- Se tutto è a posto si può procedere...
- ... oppure si possono semplicemente rimuovere uno o più file e selezionare solo quelli che interessano

`$ git add file1.js` // stage di un solo file

`$ git add .` // stage di tutti i files

Repository locale

- Repository → 'Git project'
- Contiene l'intera collection di files e cartelle associati al progetto
- È organizzato come una sequenza di *fotografie* dei files
- Ogni fotografia è un *commit* con un id, un messaggio, la data, l'autore...

```
$ git commit -m "my first commit"
```

```
// crea un'istantanea del codice attualmente in stage
```

Pubblicazione

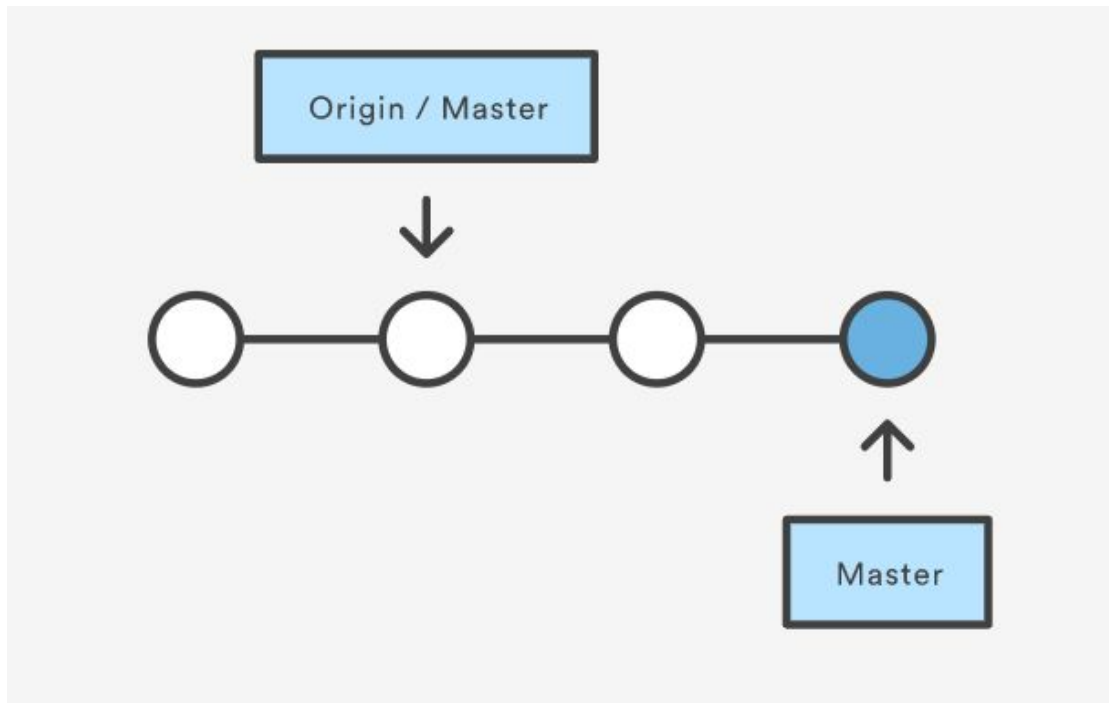
- È il momento nel quale si rendono permanenti i salvataggi
- È il momento nel quale il repository locale viene sincronizzato con **origin**, il repository remoto
- È organizzato come una sequenza di *fotografie* dei files
- Questo consentirà a tutti i collaboratori al progetto di recuperare il codice salvato in qualsiasi momento e da qualsiasi luogo!

`$ git push -u origin main`

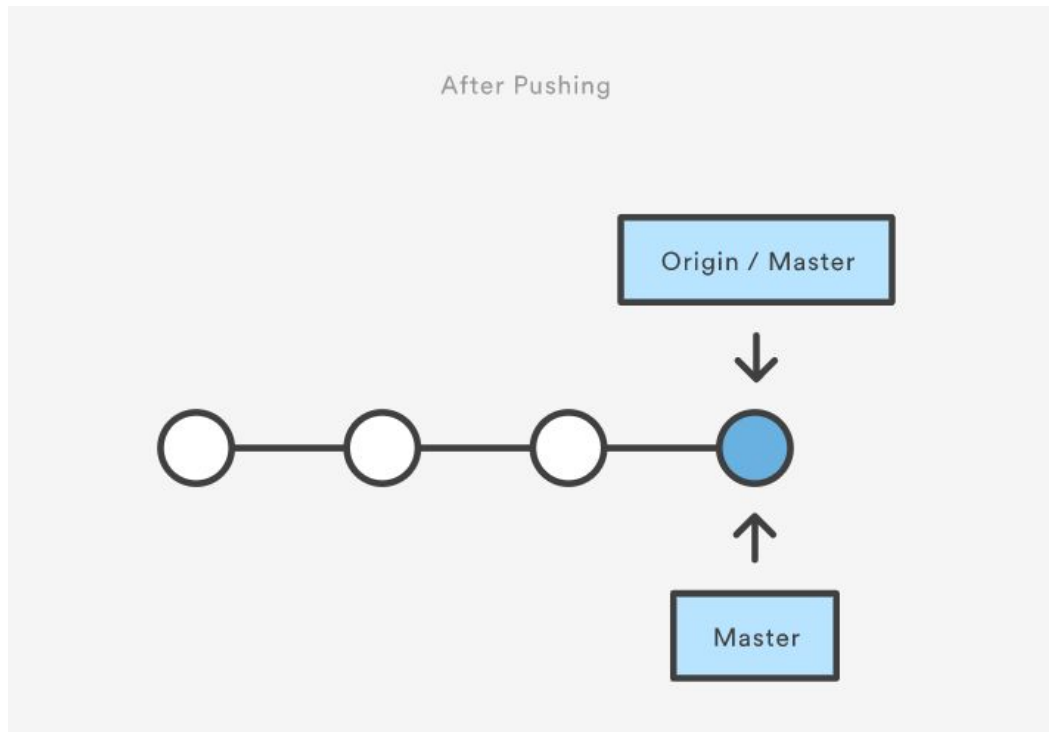
// invia il codice salvato localmente nell'origine upstream (GitHub)

// per i commit successivi si può semplicemente usare "git push"

Prima del push



Dopo il push



Principali comandi di base Git

Scrivendo **git** sulla console avrete la lista dei comandi

Per creare un nuovo repository localmente	<code>git init</code>
Per aggiungere file all'area di staging	<code>git add</code> oppure <code>git add ~filename~</code>
Per controllare lo stato dell'area di staging	<code>git status</code>
Per committare nuove modifiche	<code>git commit -m "messaggio di commit"</code>
Per creare un nuovo ramo	<code>git checkout -b ~nome ramo~</code>

Per passare da un ramo all'altro

```
git checkout ~nome ramo~
```

Per unire rami insieme

```
git merge ~nome ramo~
```

Per aggiungere un repository remoto

```
git remote add ~nome remoto~  
~https://yourremoteurl~
```

Per estrarre modifiche da un repository
remoto

```
git pull ~nome remoto~ ~nome ramo~
```

Per spingere le modifiche a un repository
remoto

```
git push ~nome remoto~ ~nome ramo~
```

Pratica e collaborazione: \$ git init / \$ git clone e gestione dei branch

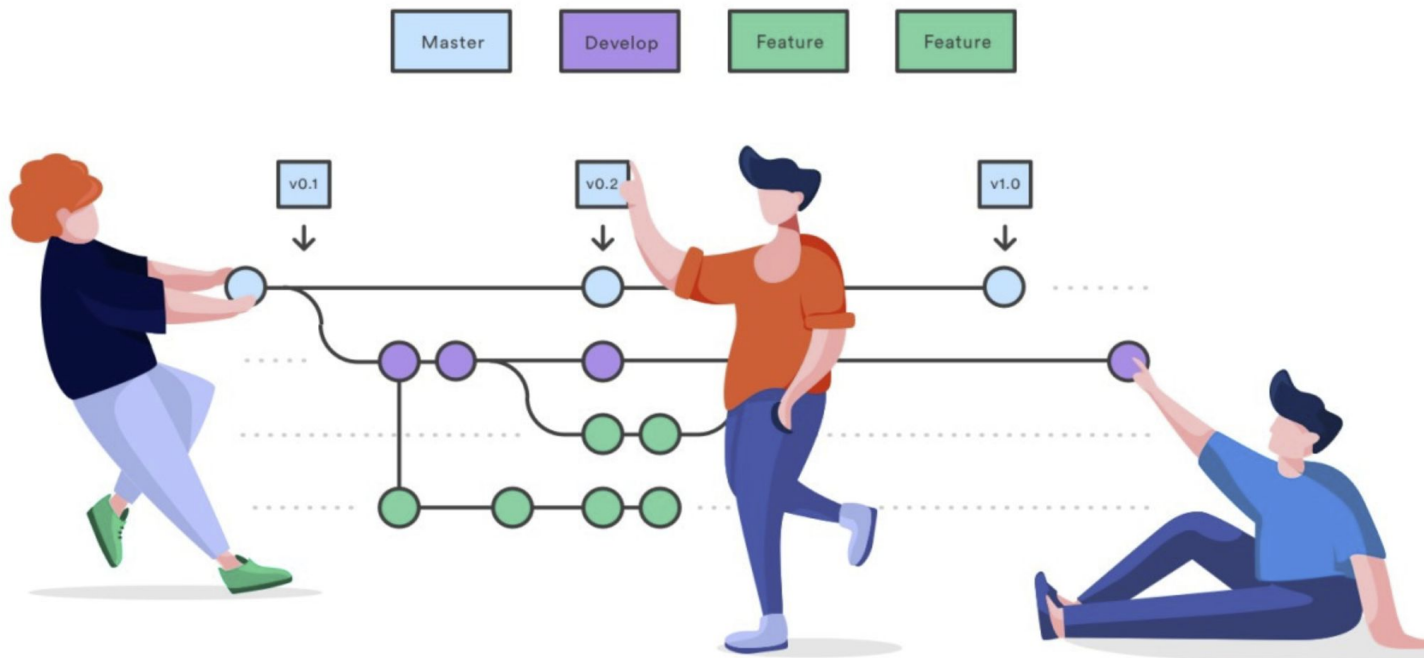
Git fetch & pull

- Fetch scarica i commit più recenti dal repository remoto senza aggiornare il repository locale
- Pull scarica i commit remoti e li unisce al repository locale

\$ git fetch

\$ git pull

Collaborazione



Git branch

- Un **branch** (ramo) è un percorso diverso staccato da quello principale
- È una tecnica molto utile per avere più flussi paralleli (perfetto per collaborare)
- In un ramo è possibile creare/gestire nuove funzionalità o correzioni di bug senza coinvolgere la parte principale del progetto
- Una volta completato il ramo, è possibile eventualmente unire quei commit nel ramo principale

Git branch

- Si crea un branch ogni volta che qualcuno dei collaboratori deve lavorare su una funzionalità senza interventi altrui sul codice.
- Una volta terminata e testata la funzionalità, sarà il momento di unirla nella base di codice principale del repository: allora la vedranno tutti.

\$ git checkout -b [new_branch_name]

// crea un nuovo ramo con il nome specificato e sposta lì il flusso personale

\$ git checkout [branch_name]

// consente di spostarsi in qualsiasi altro ramo esistente

Git branch

- Altri comandi utili:

\$ git status // informa sul branch attualmente posizionato e sui suoi dettagli

\$ git branch // elenca i branch locali

\$ git branch -r // elenca i branch remoti

\$ git branch -a // elenca tutti i branch

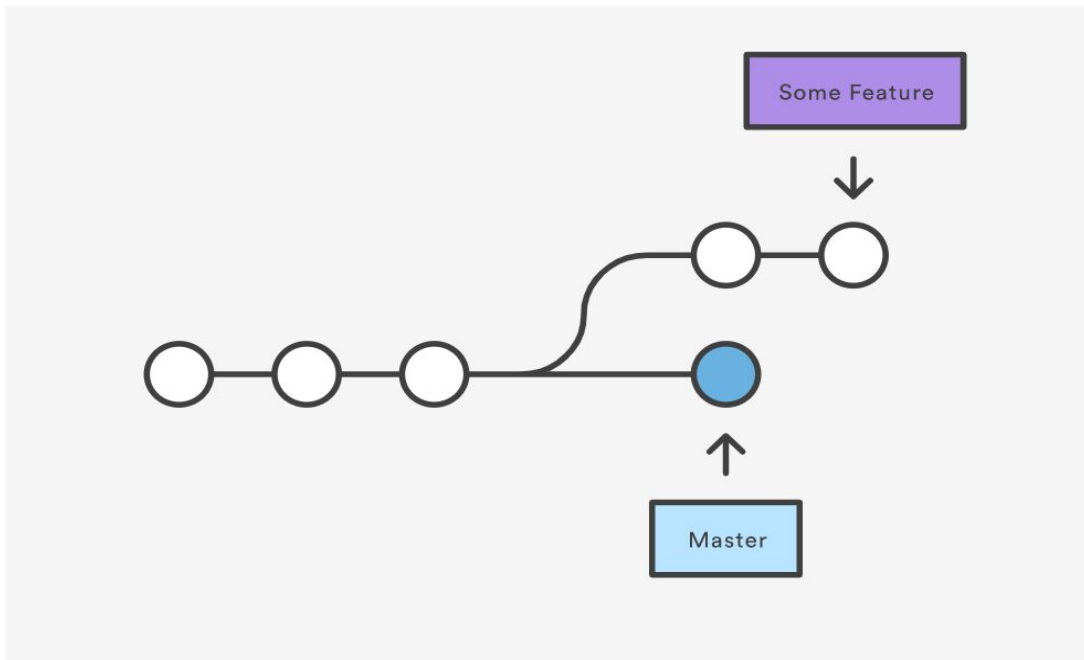
Git branch

- Per effettuare il push in un branch:

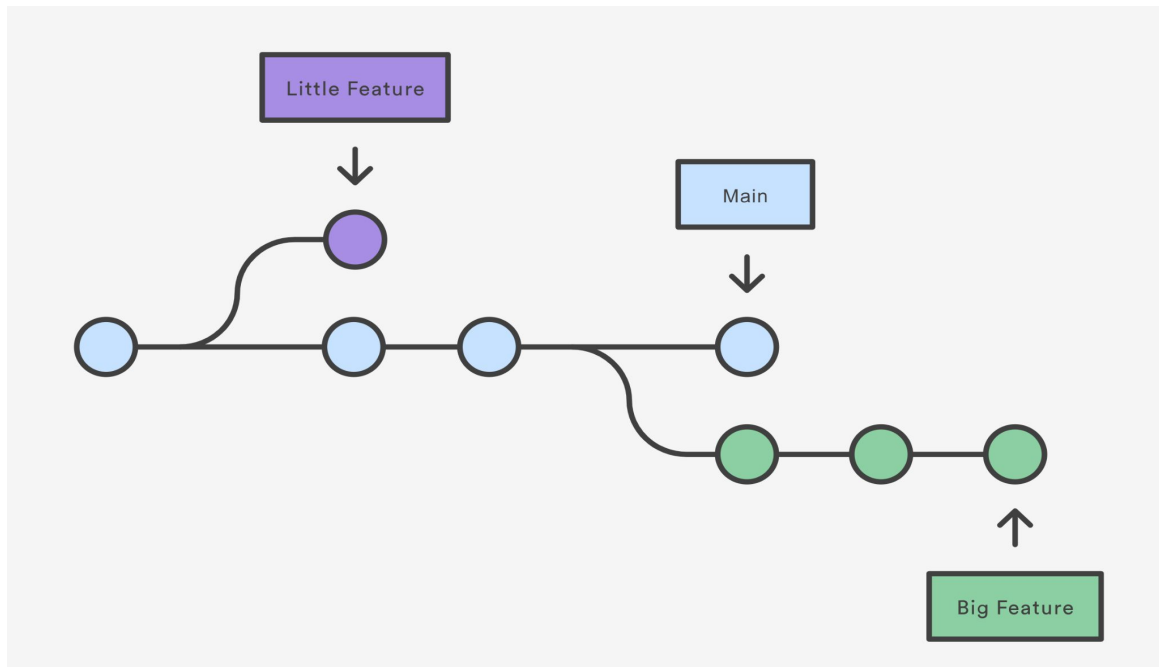
```
$ git push -u origin [branch_name]
```

// una volta spostatisi dalla posizione precedente (principale), bisogna specificare dove dovrebbe andare il nuovo codice nell'origine upstream (GitHub)

Git branch



Git branch - flusso



Git merge

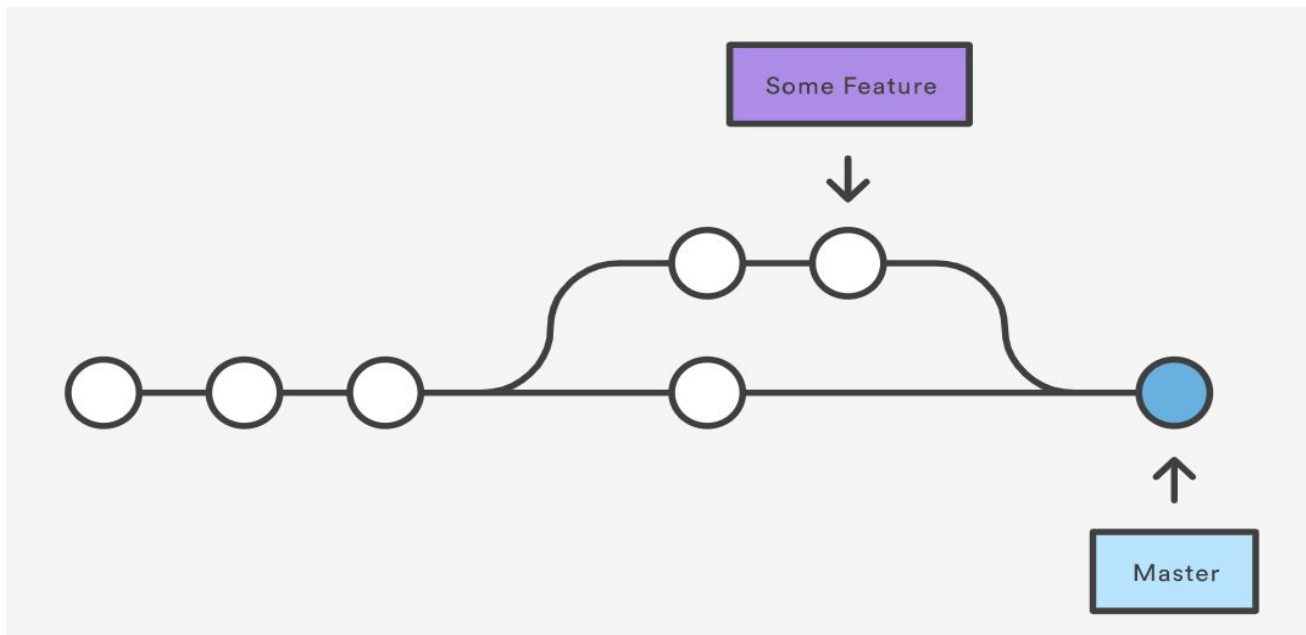
- Quando si collabora, è piuttosto comune unire diversi rami insieme
- Ciò è necessario per riunire due caratteristiche separate e comporre il nuovo stato del progetto
- Si possono unire tutte le modifiche apportate su un ramo separato al ramo corrente con il comando di unione:

`$ git merge [new_branch_name]`

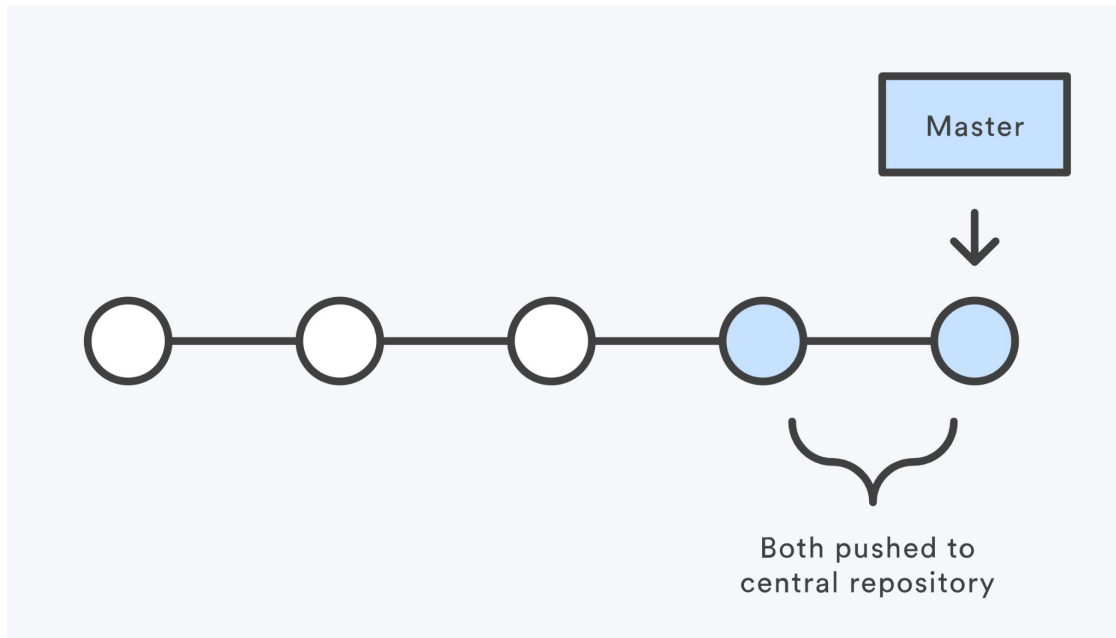
// unisce il ramo specificato in quello corrente

( Potrebbero crearsi conflitti!)

Git merge



Git merge



Git merge – risolvere i conflitti

- Il presentarsi di conflitti è comune nella gestione di un progetto Git
- Il più delle volte Git è in grado di unire e integrare automaticamente nuove modifiche
- Le funzionalità di Git non possono nulla quando due persone hanno lavorato sulle stesse righe di un file o se una persona ha eliminato un file mentre un'altra lo stava modificando
- In queste situazioni, lo sviluppatore che tenta di unire è il responsabile della risoluzione manuale dei conflitti

Git merge – risolvere i conflitti

```
58     res.send(question)
59   } catch (error) {
60     console.log(error)
61     next("While reading questions list a problem occurred!")
62   }
63 })
64
65 router.post("/", cloudinaryMulter.single("image"), async (req, res, next) => {
66   try {
67     const toCreate = JSON.parse(req.body.question)
68     toCreate.img = req.file.path
69     const newQuestion = new QuestionsModel(toCreate)
70     const { _id } = await newQuestion.save()
71     res.status(201).send(_id)
72   } catch (error) {
73     next(error)
74   }
75 })
76
77 router.put("/:id", async (req, res, next) => {
78   try {
79     const question = await QuestionsModel.findByIdAndUpdate(
80       req.params.id,
```

```
61     console.log(error)
62     next("While reading questions list a problem occurred!")
63   }
64 })
65
66 router.post(
67   "/",
68   jwt,
69   adminOnly,
70   cloudinaryMulter.single("image"),
71   async (req, res, next) => {
72     try {
73       const toCreate = JSON.parse(req.body.question)
74       toCreate.img = req.file.path
75       const newQuestion = new QuestionsModel(toCreate)
76       const { _id } = await newQuestion.save()
77       res.status(201).send(_id)
78     } catch (error) {
79       next(error)
80     }
81   }
82 )
83
```

Output

conflict 1 of 1

```
64   })
65
66   router.post("/", async (req, res, next) => {
67     try {
68       const newQuestion = new QuestionsModel(req.body)
69       const { _id } = await newQuestion.save()
70
71       res.status(201).send(_id)
72     } catch (error) {
73       next(error)
74     }
75   })
76
77   router.post(
78     "/withImage",
79     cloudinaryMulter.single("image"),
80     async (req, res, next) => {
81       try {
82         const toCreate = JSON.parse(req.body.question)
83         toCreate.img = req.file.path
84         const newQuestion = new QuestionsModel(toCreate)
85         const { _id } = await newQuestion.save()
86         res.status(201).send(_id)
87       }
88     }
89   })
90 }
```

Git merge – risolvere i conflitti

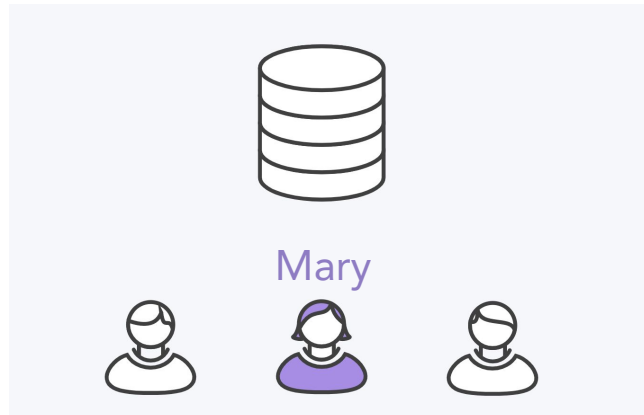
```
65 Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
66 <<<<<< Updated upstream (Current Change)
67 router.post("/", cloudinaryMulter.single("image"), async (req, res, next)
68   try {
69     const toCreate = JSON.parse(req.body.question)
70     toCreate.img = req.file.path
71     const newQuestion = new QuestionsModel(toCreate)
72     const { _id } = await newQuestion.save()
73     res.status(201).send(_id)
74   } catch (error) {
75     next(error)
76   }
77 })
78
79 router.put("/:id", async (req, res, next) => {
80   =====
81   router.post(
82     "/",
83     jwt,
84     adminOnly,
85     cloudinaryMulter.single("image"),
86     async (req, res, next) => {
87       try {
88         const toCreate = JSON.parse(req.body.question)
```

Lavorare su un Progetto

Esempio di flusso collaborativo

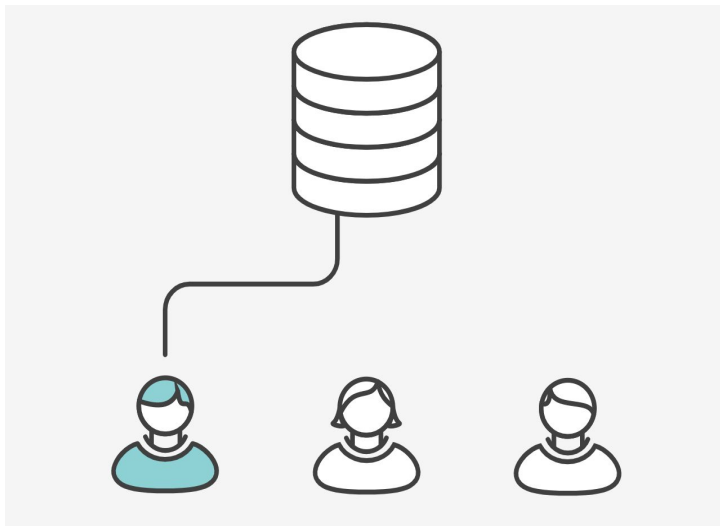
Lavorare su un progetto

- Facciamo un esempio generale di come un piccolo team tipico collaborerebbe utilizzando il flusso di lavoro Git
- Vedremo come due sviluppatori lavoreranno su funzionalità separate e condivideranno i loro contributi tramite un repository centralizzato



Lavorare su un progetto

- John pubblica la sua funzionalità



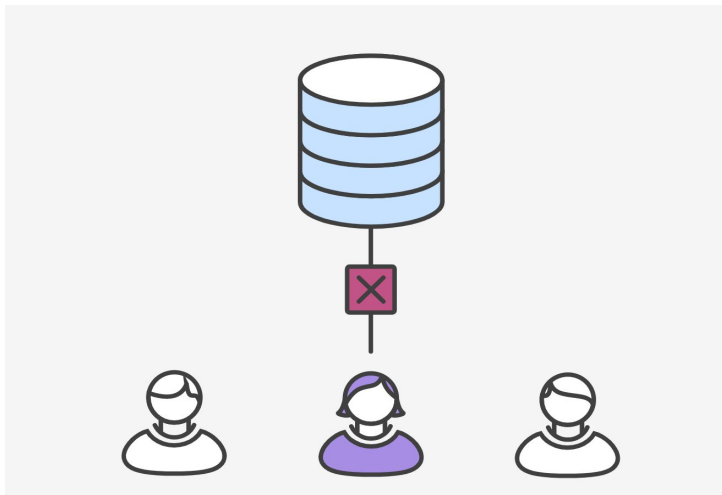
Una volta che John ha terminato la sua funzionalità, dovrebbe pubblicare i suoi commit locali nel repository centrale in modo che altri membri del team possano accedervi. Può farlo con il comando:

git push origin main

Poiché il repository centrale non è stato aggiornato da quando John lo ha clonato, ciò non comporterà alcun conflitto e il push funzionerà come previsto.

Lavorare su un progetto

- Mary cerca di effettuare un push

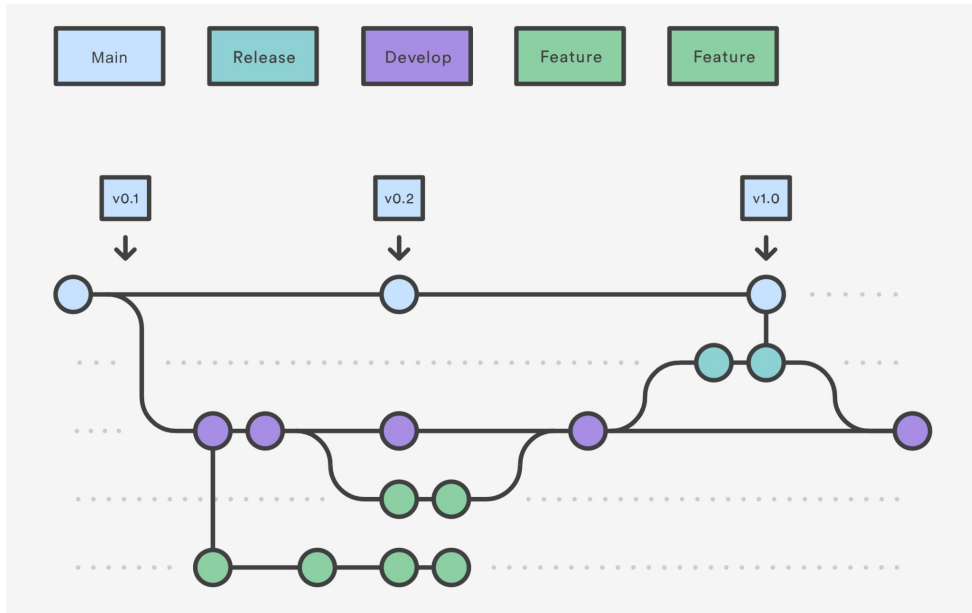


Ma, poiché la sua cronologia locale si è discostata dal repository centrale, Git rifiuterà la richiesta con un messaggio di errore.

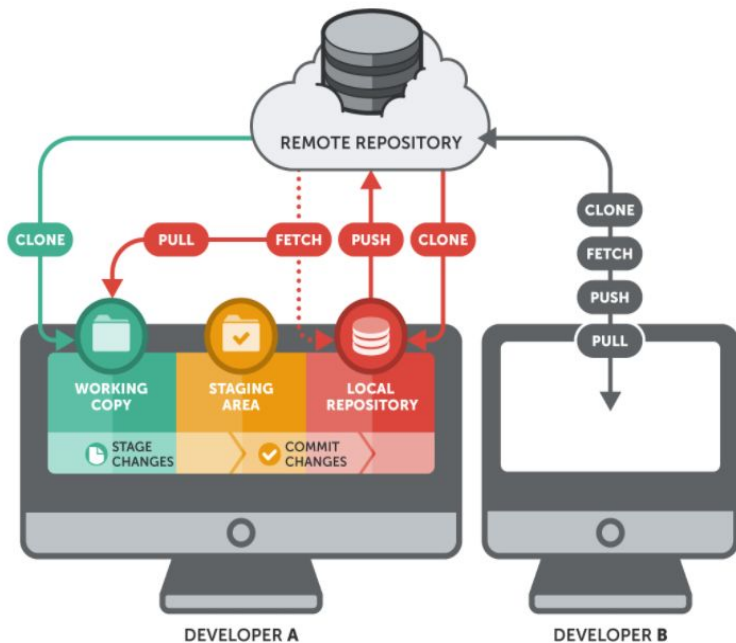
Ciò impedisce a Mary di sovrascrivere i commit ufficiali. **Ha bisogno di estrarre** gli aggiornamenti di John nel suo repository, integrarli con le sue modifiche locali e **quindi riprovare**.

Versioning e cronologia dei branch

- Ed ecco come potrebbe potenzialmente apparire un flusso git completo:

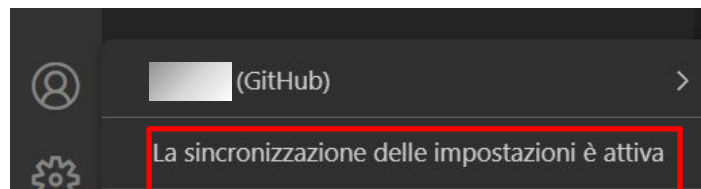
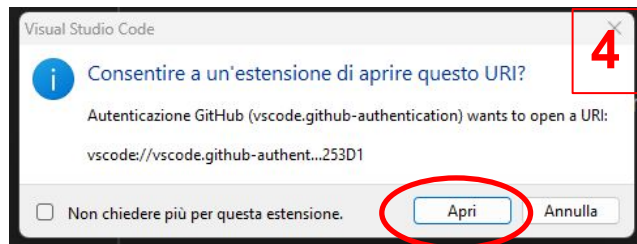
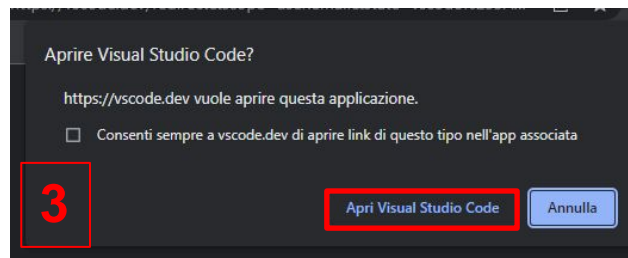
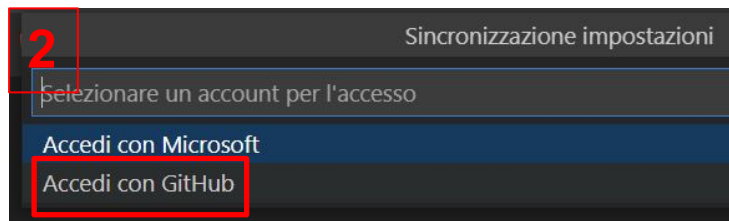
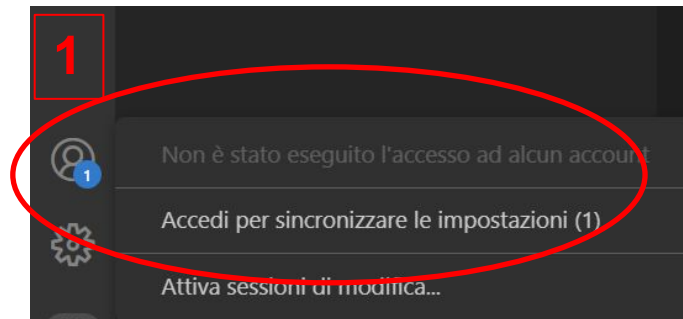


Riepilogo

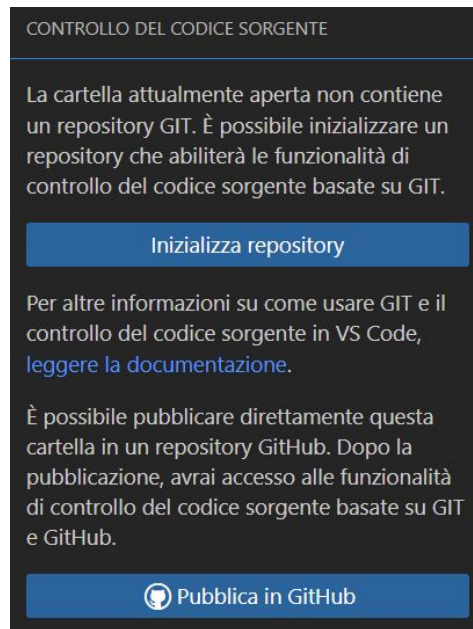
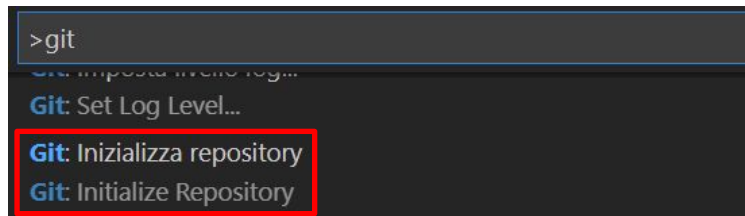


Integrazione GIT-VSCode

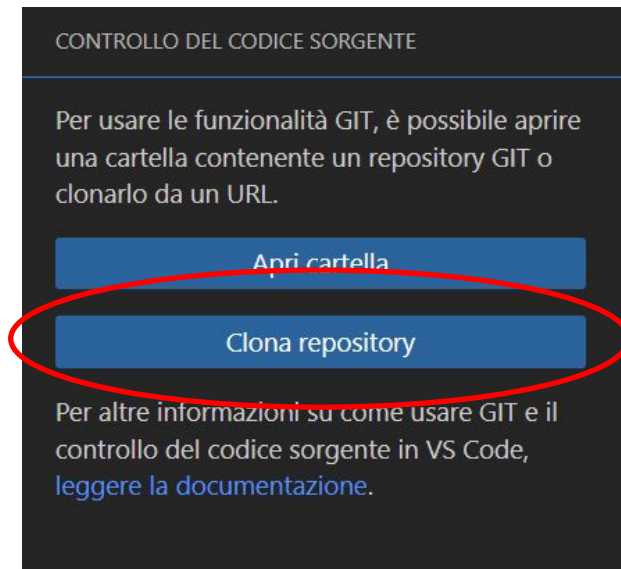
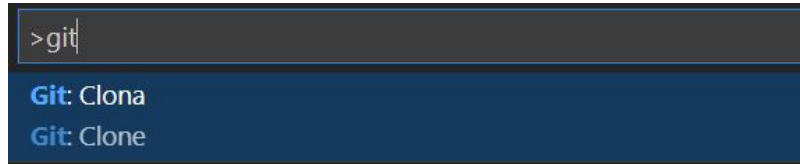
Cliccando su “Account” nell’ambiente di VSCode, è possibile accedere con il proprio account GitHub, seguendo le istruzioni a video.



Per inizializzare un nuovo repository è possibile usare la palette dei comandi o il pannello di controllo.

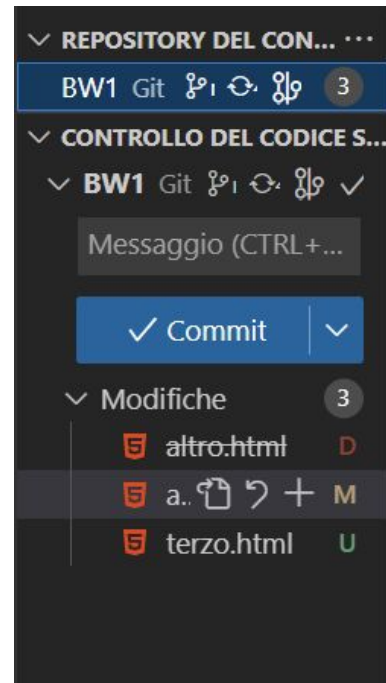


Per clonare un repository esistente è possibile usare la palette dei comandi o il pannello di controllo e seguire le istruzioni a video.

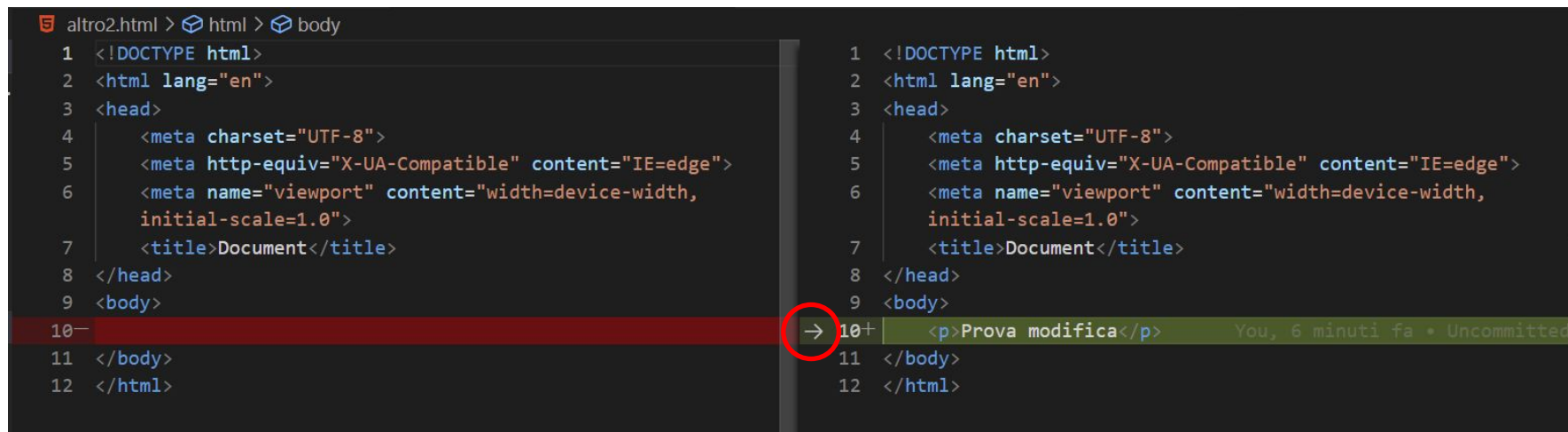


Il pannello di esplora risorse funziona esattamente come in qualsiasi Progetto locale, il pannello di Git invece presenterà sia i file presenti in remote che quelli in locale, con una lettera accanto (che si riferisce alle differenze tra la cartella locale e la cartella remota):

- **"U"** verde: file aggiunti;
- **"M"** gialla: file modificati;
- **"D"** rossa: file eliminati (nome file barrato);
- **"!"** rosso: file in conflitto



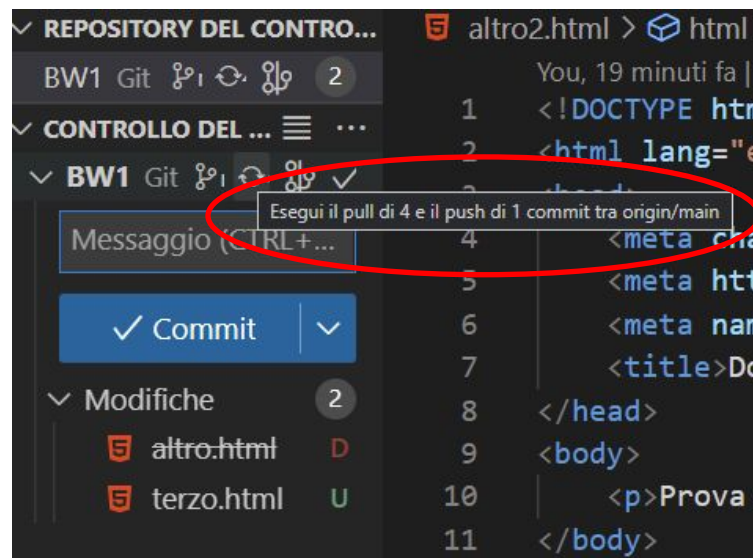
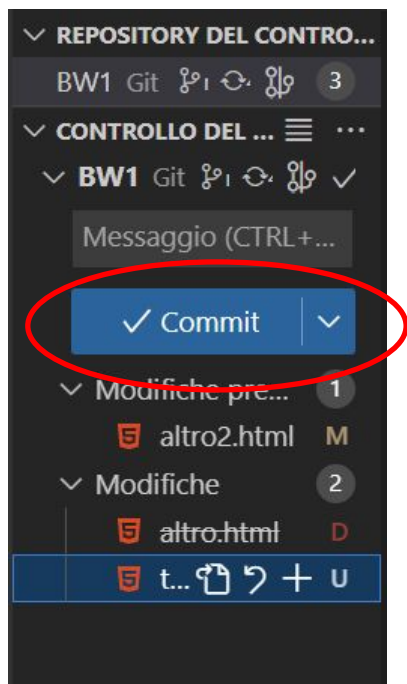
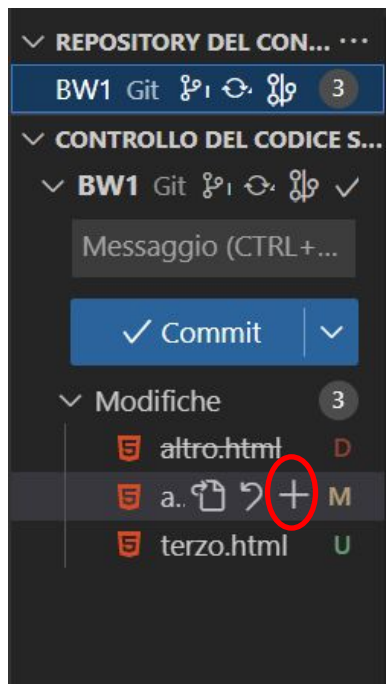
Cliccando su un file modificato, l'area di lavoro di VS Code mostrerà lo stato del file prima e dopo la modifica, consentendo eventualmente di tornare al codice precedente facendo click sulla freccia accanto alle righe modificate.



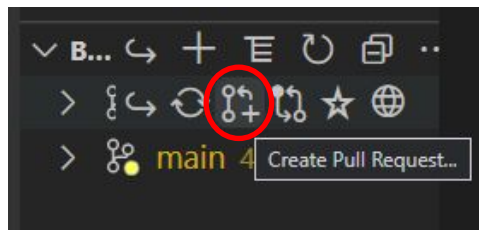
```
altro2.html > html > body
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width,
  initial-scale=1.0">
7   <title>Document</title>
8 </head>
9 <body>
10-
11 </body>
12 </html>
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width,
  initial-scale=1.0">
7   <title>Document</title>
8 </head>
9 <body>
10+ <p>Prova modifica</p> You, 6 minuti fa • Uncommitted
11 </body>
12 </html>
```

Cliccando infine sul simbolo “+” accanto a un file questo sarà aggiunto allo staging, e sarà poi possibile eseguirne il commit e la sincronizzazione con le operazioni di pull e push.



Al termine della sincronizzazione può essere effettuata una richiesta di pull dal proprio branch verso il branch main: si aprirà così una finestra di github.com dalla quale chi gestisce il branch potrà comunicare le modifiche al gestore del ramo main, che successivamente potrà quindi effettuare il merge o eventualmente rifiutare le modifiche.



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main

←

compare: pasquale

✓ Able to merge. These branches can be automatically merged.

richiesta pull

Write

Preview

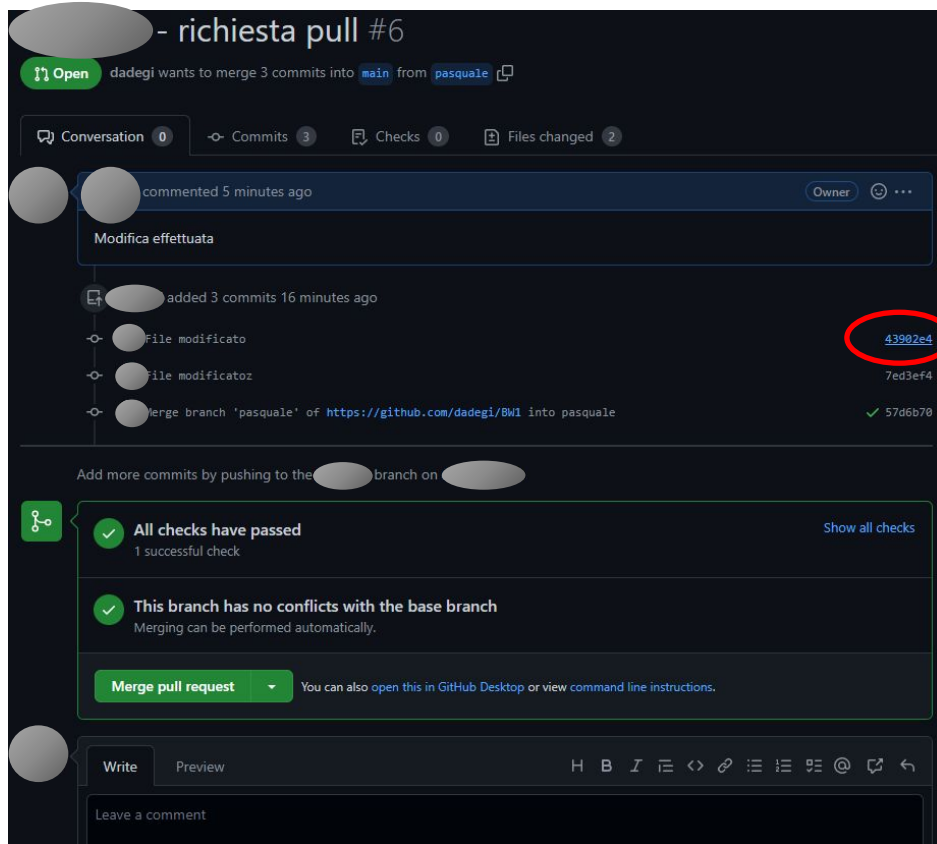
H B I ≡ <> 🔗 ≡ ≡ ≡ @ ↗ ↶

Modifica effettuata

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

Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).



The screenshot shows a GitHub pull request titled "richiesta pull #6". The pull request is open, and the user "dadegi" wants to merge 3 commits into the "main" branch from the "pasquale" branch. The interface includes tabs for Conversation (0), Commits (3), Checks (0), and Files changed (2). A comment from the pull request owner states "Modifica effettuata" (Modification made). Below the comment, a commit history is shown, including "added 3 commits 16 minutes ago" and "Merge branch 'pasquale' of https://github.com/dadegi/Bw1 into pasquale". A red circle highlights the commit hash "43902e4". At the bottom, a green box indicates that all checks have passed and the branch has no conflicts with the base branch, allowing for an automatic merge. The "Merge pull request" button is visible.

- richiesta pull #6

Open dadegi wants to merge 3 commits into `main` from `pasquale`

Conversation 0 Commits 3 Checks 0 Files changed 2

commented 5 minutes ago **Owner**

Modifica effettuata

added 3 commits 16 minutes ago

- File modificato **43902e4**
- File modificato 7ed3ef4
- Merge branch 'pasquale' of <https://github.com/dadegi/Bw1> into pasquale 57d6b70

Add more commits by pushing to the `pasquale` branch on `dadegi`

All checks have passed 1 successful check [Show all checks](#)

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request You can also [open this in GitHub Desktop](#) or [view command line instructions](#).

Write Preview `H B I L < > @ ↩ ↪`

Leave a comment

- richiesta pull #6

Open

dadegi wants to merge 3 commits into `main` from `pasquale`

Conversation 0

Commits 3

Checks 1

Files changed 2

Changes from 1 commit

File filter

Conversations

Jump to

Settings

Review changes

File modificato

< Prev

Next >

pasquale (#6)

committed 18 minutes ago

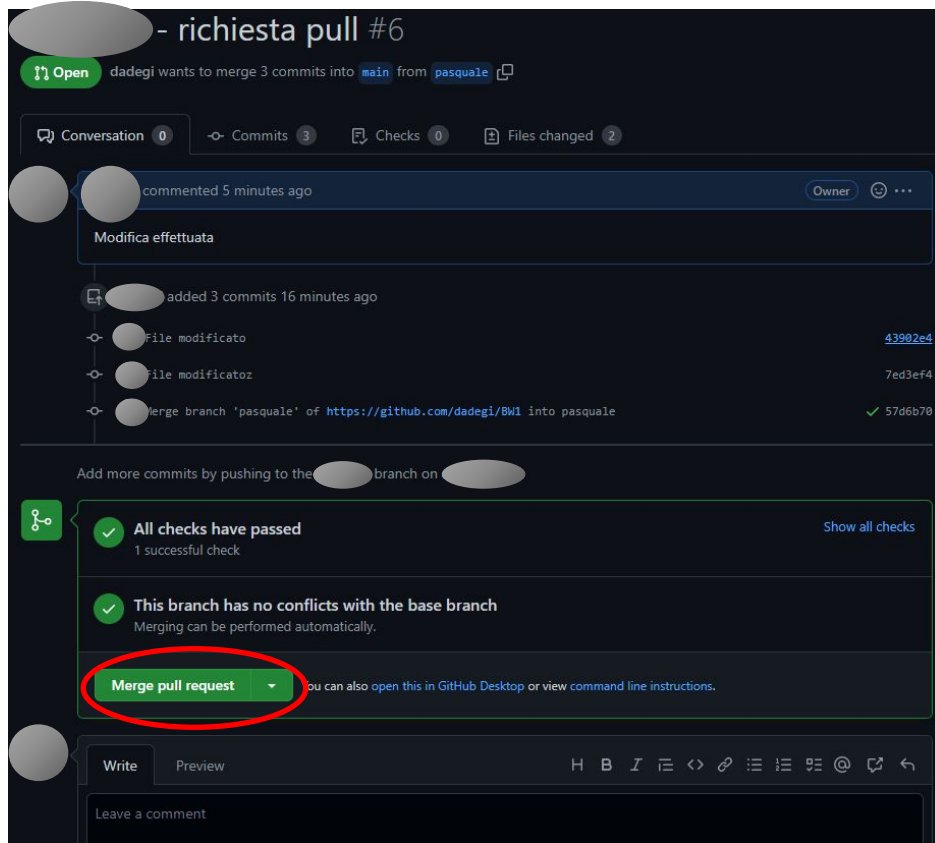
commit 43902e474641b4b14a38d76b165046e71f18e141

12

terzo.html

... @@ -0,0 +1,12 @@

```
1 + <!DOCTYPE html>
2 + <html lang="en">
3 + <head>
4 +   <meta charset="UTF-8">
5 +   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6 +   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7 +   <title>Document</title>
8 + </head>
9 + <body>
10 +   <h2>Per finire ho modificato questo</h2>
11 + </body>
12 + </html>
```



- richiesta pull #6

Open dadegi wants to merge 3 commits into `main` from `pasquale`

Conversation 0 Commits 3 Checks 0 Files changed 2

commented 5 minutes ago Owner

Modifica effettuata

added 3 commits 16 minutes ago

- File modificato [43902e4](#)
- File modificato [7ed3ef4](#)
- Merge branch 'pasquale' of <https://github.com/dadegi/Bw1> into pasquale [57d6b70](#)

Add more commits by pushing to the `pasquale` branch on `dadegi`

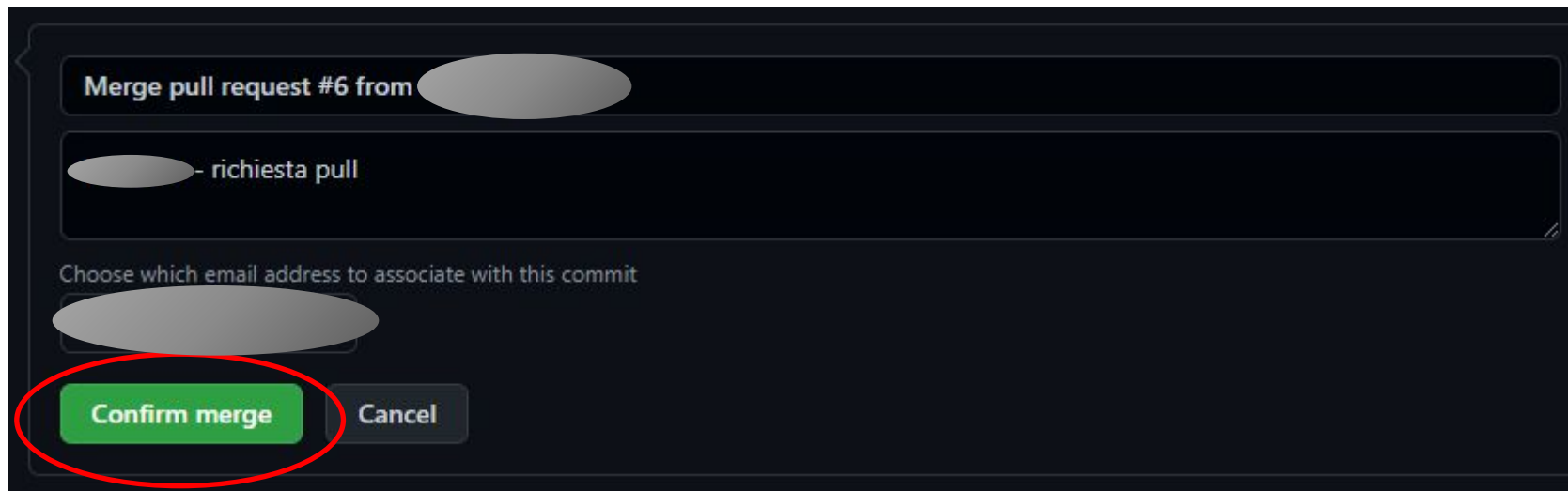
All checks have passed [Show all checks](#)
1 successful check

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request You can also [open this in GitHub Desktop](#) or [view command line instructions](#).

Write Preview H B I

Leave a comment



Merge pull request #6 from [redacted]

[redacted] - richiesta pull

Choose which email address to associate with this commit

[redacted]

Confirm merge Cancel

The image shows a dark-themed Git merge confirmation dialog. At the top, it says 'Merge pull request #6 from' followed by a redacted repository name. Below that, it shows the pull request title '[redacted] - richiesta pull'. Then, it asks 'Choose which email address to associate with this commit' with a redacted email address below it. At the bottom, there are two buttons: 'Confirm merge' (highlighted with a red circle) and 'Cancel'.



GRAZIE
Epicode