



Corso Git & GitHub – Lezione 3

Merging, lavoro in remoto e collaborazione

Elisabetta Ferri, Giorgio Micaglio, Gianmarco Puleo, Michele Tognoni

Associazione Italiana Studenti di Fisica
Comitato Locale di Trento

May 15, 2024

Overview

1. Merging

- 1.1 Tipi di merging
- 1.2 Conflitti base di merge

2. Lavoro in remoto

- 2.1 Repository remote
- 2.2 GitHub
- 2.3 Creare, inizializzare e clonare una repo remota
- 2.4 Workflow remoto

3. Collaborazione

- 3.1 Workflow centralizzato
- 3.2 Esempio di workflow

Merging

AISF, Comitato Locale di Trento

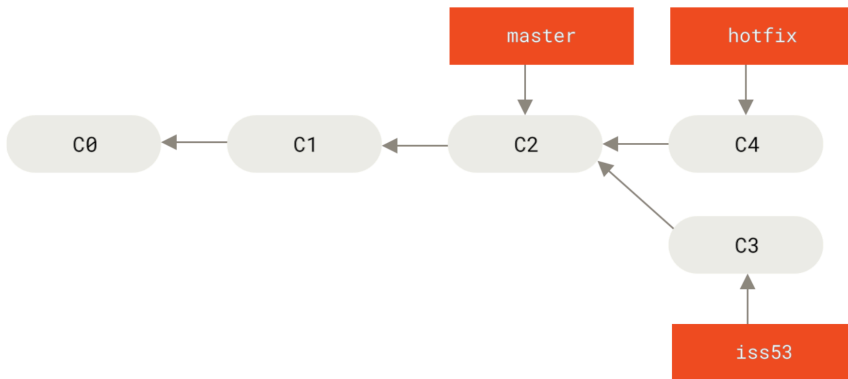
Tipi di merging

Ci sono vari tipi di merging:

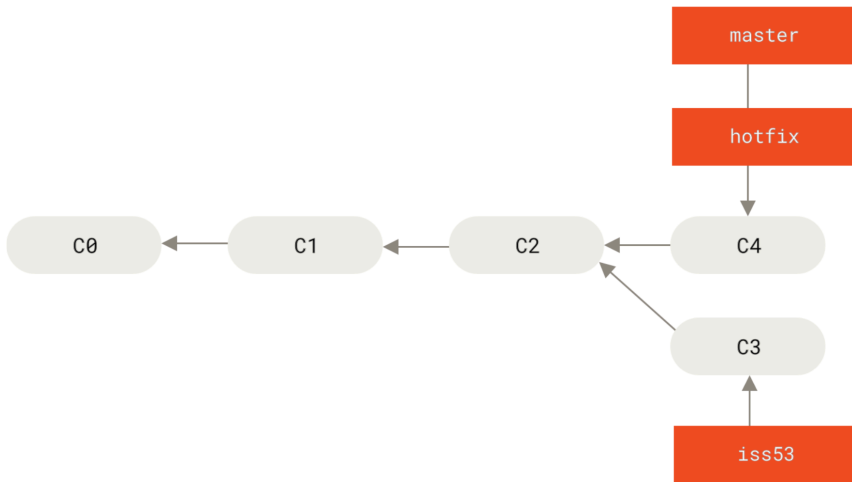
- fast-forward
- 3-way (per questo tipo vi sono diverse strategie usate da Git, come ad esempio `ort` e `recursive`)
- rebasing (che non è un merge)

Per approfondimenti: <https://git-scm.com/docs/git-merge>

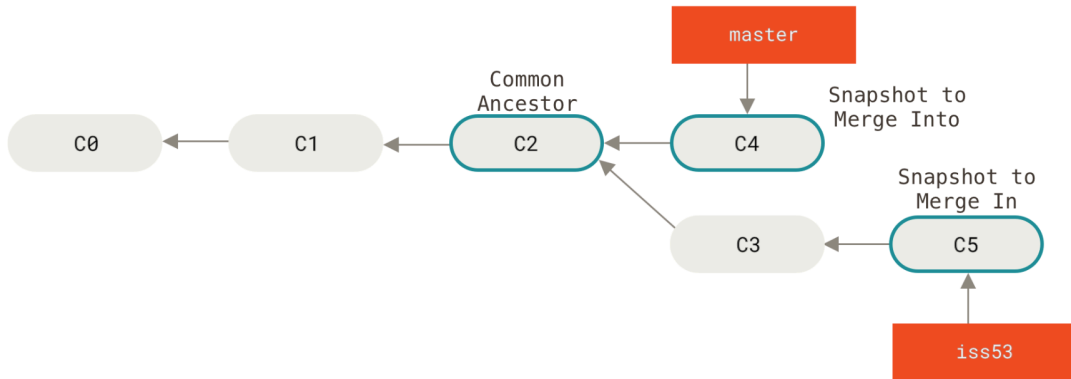
Tipi di merging: fast-forward



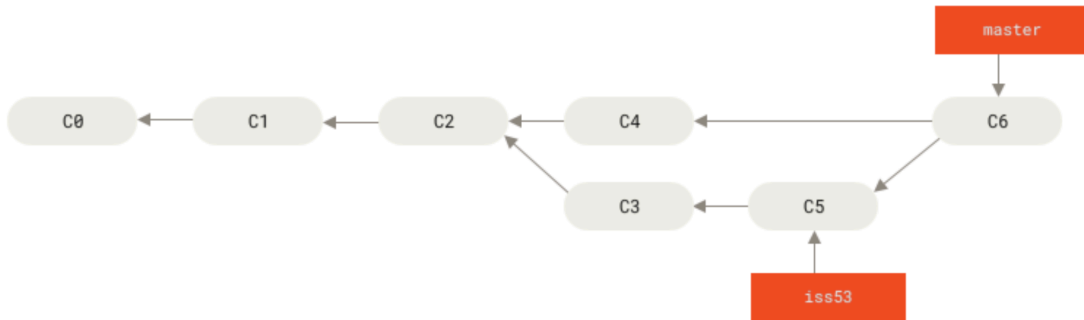
Tipi di merging: fast-forward



Tipi di merging: 3-way merge



Tipi di merging: 3-way merge



Conflitti base di merge

BRACE YOURSELF

**MERGE CONFLICTS ARE
COMING**

memegenerator.net

Conflitti base di merge

Quando si modifica lo stesso file in due branch diverse e ne si vuole fare il merge, git darà un errore del genere:

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Il processo di merging viene interrotto e il file viene modificato in questo modo:

Conflitti base di merge

```
<<<<<< HEAD:index.html
<div id="footer">
contact : email.support@github.com
</div>
=====
<div id="footer">
please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```

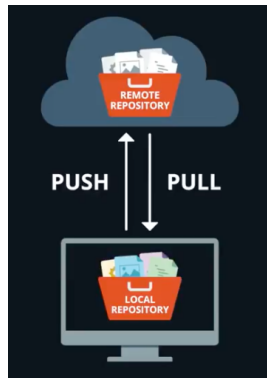
Una volta risolto il conflitto, si aggiunge il file modificato all'index e, come al solito, si procede con il commit.

Lavoro in remoto

AISF, Comitato Locale di Trento

Repository remote

- Sono versioni di un progetto Git hostate su Internet o, più in generale, su una rete
- Si possono configurare più repo remote per progetto
- Collaborare con altri significa condividere il proprio lavoro attraverso una di queste
- Ciò viene fatto attraverso azioni di **push** (uploading) e **pull** (downloading)
- In questo corso useremo le repo remote hostate da GitHub



GitHub

- Uno dei più grandi host di repo Git, punto centrale di collaborazione per milioni di sviluppatori
- Usato da molti progetti open-source per hosting, tracciamento di problemi, revisione del codice e altro



Nota bene

Sono necessarie registrazione e configurazione con SSH (lezione 1), che potete verificare con `ssh -T git@github.com`

GitHub

**USING
AIRDROP**



**USING
GOOGLE DRIVE**



**USING A
PRIVATE DISCORD**



**USING
GITHUB**



Creare una repo remota

Per iniziare a lavorare in remoto, abbiamo bisogno di creare una repo remota su GitHub.

Bisogna aprire la homepage del sito di GitHub (<https://github.com>) e cliccare, in alto a sinistra, su **New**.

Appare una schermata del genere

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

Owner *

giorgioni

Repository name *

/

Great repository names are short and memorable. Need inspiration? How about [turbo-laserson](#)?

Description (optional)

☒ Public

Anyone on the Internet can see this repository. You choose who can commit.

☐ Private

You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

License: None

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

☐ You are creating a public repository in your personal account.

Create repository

Creare una repo remota

- Inserire il nome della repository
- Selezionare tra repo pubblica (visibile a tutti, ma read-only) e privata
- NON inizializzare la repo con README.md (lasciare la casella vuota)
- NON inizializzare la repo con .gitignore (scegliere .gitignore template: None)
- NON inizializzare la repo con licenza (scegliere License: None)

Creare una repo remota

- Inserire il nome della repository
- Selezionare tra repo pubblica (visibile a tutti, ma read-only) e privata
- NON inizializzare la repo con README.md (lasciare la casella vuota)
- NON inizializzare la repo con .gitignore (scegliere .gitignore template: None)
- NON inizializzare la repo con licenza (scegliere License: None)

Tranquilli!

Queste azioni possono essere eseguite in seguito

- Cliccare in basso a destra sul bottone verde
- La nuova repo remota si trova all'URL
`https://github.com/user-name/repo-name`

Inizializzare una repo remota

Ora che avete creato la repo remota potete procedere in due modi:

1. Inizializzare la repo remota con una repo locale di git che avete già
2. Clonare la repo remota (vuota) in locale

Vediamo il primo modo (**inizializzazione**), dovete lanciare questi comandi:

```
git branch -M main  
git remote add origin git@github.com:USERNAME/REPONAME.git  
git push -u origin main
```

Il primo serve a rinominare la branch principale in `main`, il secondo ad aggiungere un collegamento remoto alla repo che avete appena creato, il terzo ad inviare i contenuti della repo sul server.

Cloning di una repo remota

Ora ci occupiamo di **clonare** una repo remota dal server di GitHub in locale.

Nota bene

Questo metodo è da usare esclusivamente se non avete ancora inizializzato un progetto locale

Per farlo, usiamo

```
git clone git@github.com:user-name/repo-name.git
```

Questo comando clona la repo remota in una nuova directory, chiamata repo-name.
Potete clonare qualsiasi repo pubblicata su GitHub!

Workflow - remote

Ora andiamo a vedere in dettaglio come lavorare con le repo remote. Per farlo, dobbiamo ovviamente spostarci nella directory della repo con `cd`.

Il comando

```
git remote
```

mostra i server remoti che abbiamo configurato per la repo locale. Per entrambi i casi che abbiamo visto, il comando dovrebbe restituire il nome `origin`, che git dà di default al server da qui clona.

Workflow - fetch

Per prendere i dati dal server, possiamo lanciare

```
git fetch <remote>
```

Questo comando prende i dati dalla repo remota (che sono stati aggiunti dall'ultima volta che si è lanciato il `fetch`) e li porta in locale.

Workflow - fetch

Per prendere i dati dal server, possiamo lanciare

```
git fetch <remote>
```

Questo comando prende i dati dalla repo remota (che sono stati aggiunti dall'ultima volta che si è lanciato il `fetch`) e li porta in locale.

Nota bene

Fare `fetch` permette solo il download in locale, non esegue il merge. Si deve lanciare eventualmente `git merge`, altrimenti...

Workflow - pull

Il comando

```
git pull
```

fa automaticamente un fetch e un merge dalla repo remota a quella locale.

Workflow - pull

Il comando

```
git pull
```

fa automaticamente un fetch e un merge dalla repo remota a quella locale.

Nota bene

La prima volta che userete questo comando, vedrete un warning che vi indica di impostare la variabile `pull.rebase`

Per impostare il comportamento di default di Git (fast-forward se possibile, altrimenti merge commit), bisogna usare `git config --global pull.rebase "false"`

Workflow - push

Per inviare dati al server, usiamo il comando

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

- Lo facciamo quando il progetto è ad un punto tale da voler essere condiviso
- Ogni commit eseguito viene così inviato al server, e il branch del server viene aggiornato (si sposta sul nostro ultimo commit).

Workflow - push

Per inviare dati al server, usiamo il comando

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

- Lo facciamo quando il progetto è ad un punto tale da voler essere condiviso
- Ogni commit eseguito viene così inviato al server, e il branch del server viene aggiornato (si sposta sul nostro ultimo commit).

Nota bene

`git push` funziona solo se nessun collaboratore lo ha usato nel frattempo! Vedremo in seguito come risolvere...

Workflow - push



Collaborazione

AISF, Comitato Locale di Trento

Workflow centralizzato

- È quello che si usa nella maggior parte dei casi semplici, anche se ce ne sono altri (integration manager, dictator-lieutenants, ecc...)
- È comodo per lavorare con pochi collaboratori (2-4), ma non è limitato
- È familiare anche a chi non ha mai usato Git
- È composto da:
 1. **Hub centrale** (o **repository**): accetta codice e lascia sincronizzare gli sviluppatori con essa
 2. **Nodi**: gli sviluppatori stessi, che interagiscono solo con l'hub

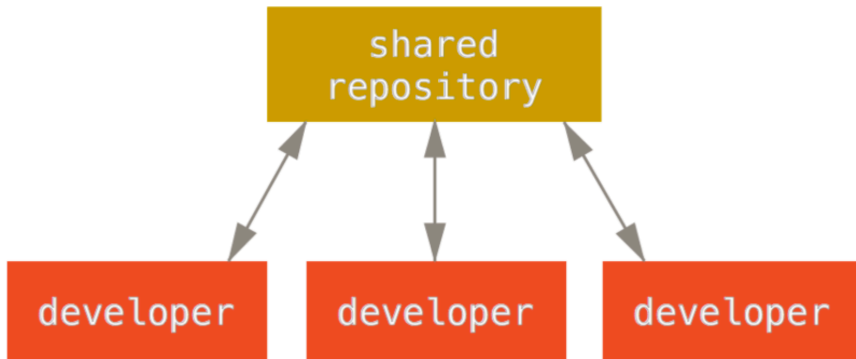
Workflow centralizzato

- È quello che si usa nella maggior parte dei casi semplici, anche se ce ne sono altri (integration manager, dictator-lieutenants, ecc...)
- È comodo per lavorare con pochi collaboratori (2-4), ma non è limitato
- È familiare anche a chi non ha mai usato Git
- È composto da:
 1. **Hub centrale** (o **repository**): accetta codice e lascia sincronizzare gli sviluppatori con essa
 2. **Nodi**: gli sviluppatori stessi, che interagiscono solo con l'hub

Nota bene

In questo workflow, i nodi interagiscono solo con l'hub, e non tra di loro.

Workflow centralizzato



Esempio di workflow

- Alice e Bob clonano dall'hub ed eseguono cambiamenti, il primo che fa il push verso l'hub (Alice) non ha problemi
- Il secondo invece (Bob), prima di eseguire il push deve:
 1. Fare il fetch dalla repository remota, che contiene il lavoro di Alice
 2. Eseguire il merge in locale
 3. Risolvere i conflitti senza sovrascrivere i cambiamenti di Alice

Esempio di workflow

- Alice e Bob clonano dall'hub ed eseguono cambiamenti, il primo che fa il push verso l'hub (Alice) non ha problemi
- Il secondo invece (Bob), prima di eseguire il push deve:
 1. Fare il fetch dalla repository remota, che contiene il lavoro di Alice
 2. Eseguire il merge in locale
 3. Risolvere i conflitti senza sovrascrivere i cambiamenti di Alice

Nota bene

Se Bob prova a fare il push prima di fetch e merge, il server lo blocca.

Esempio di workflow

- Alice e Bob clonano dall'hub ed eseguono cambiamenti, il primo che fa il push verso l'hub (Alice) non ha problemi
- Il secondo invece (Bob), prima di eseguire il push deve:
 1. Fare il fetch dalla repository remota, che contiene il lavoro di Alice
 2. Eseguire il merge in locale
 3. Risolvere i conflitti senza sovrascrivere i cambiamenti di Alice

Nota bene

Se Bob prova a fare il push prima di fetch e merge, il server lo blocca.

Nota benissimo

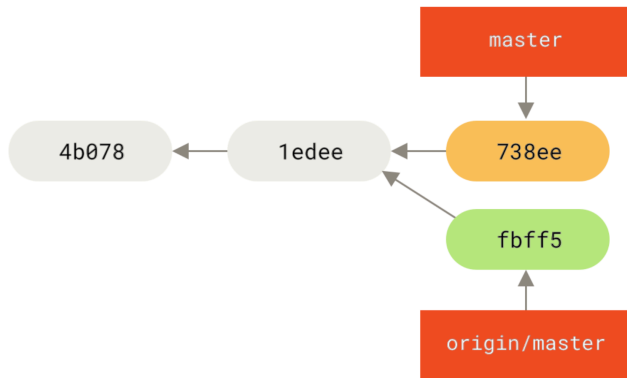
NON FATE `git push --force`, cancellerete i commit dei collaboratori

Esempio di workflow



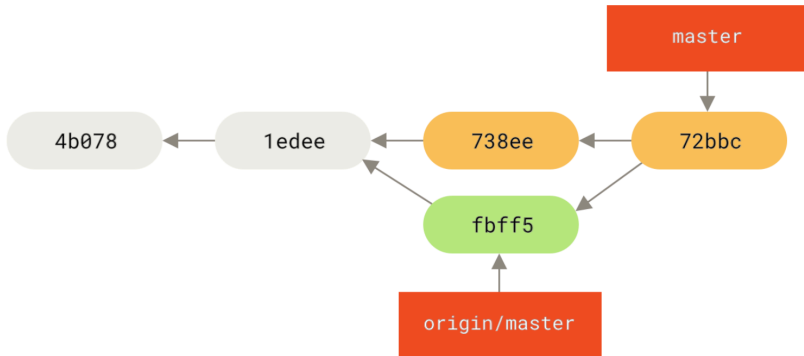
Repo di Bob - fetch

Bob fa il fetch dalla repository remota, che contiene il lavoro di Alice (fbff5)



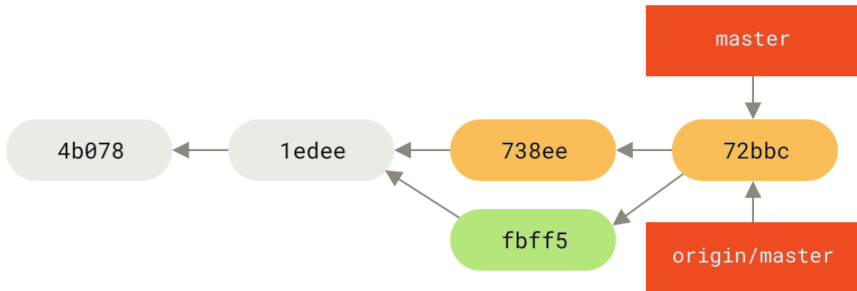
Repo di Bob - merge

Bob esegue il merge in locale (72bbc)



Repo di Bob - push

Bob esegue il push

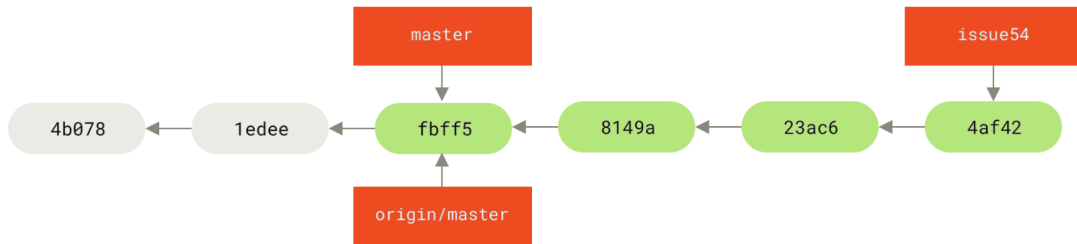


Esempio di workflow

- Nel frattempo, Alice ha creato una nuova branch `issue` in cui ha fatto 3 commit
- Sapendo che Bob ha fatto un push del suo lavoro, vuole darci uno sguardo e fa il `fetch`
- Alice si sposta nella branch `master`
- Fa il merge di `issue` in `master` (fast-forward)
- Fa il merge di `origin/master` (il commit di Bob)
- Esegue il push

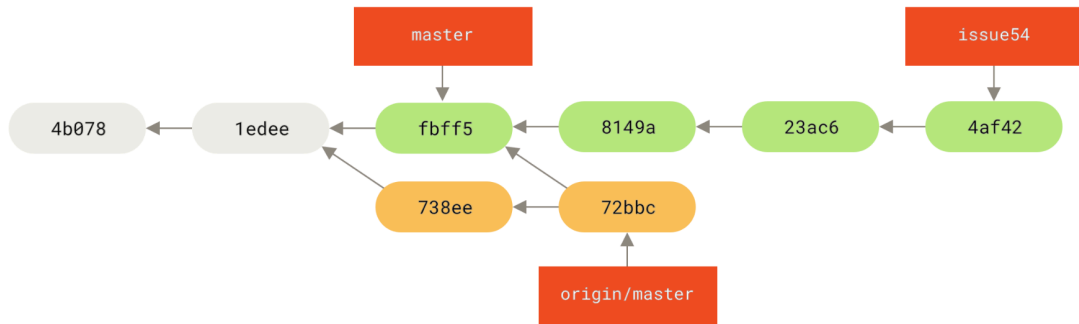
Repo di Alice

Alice ha eseguito 3 commit nella branch `issue54`



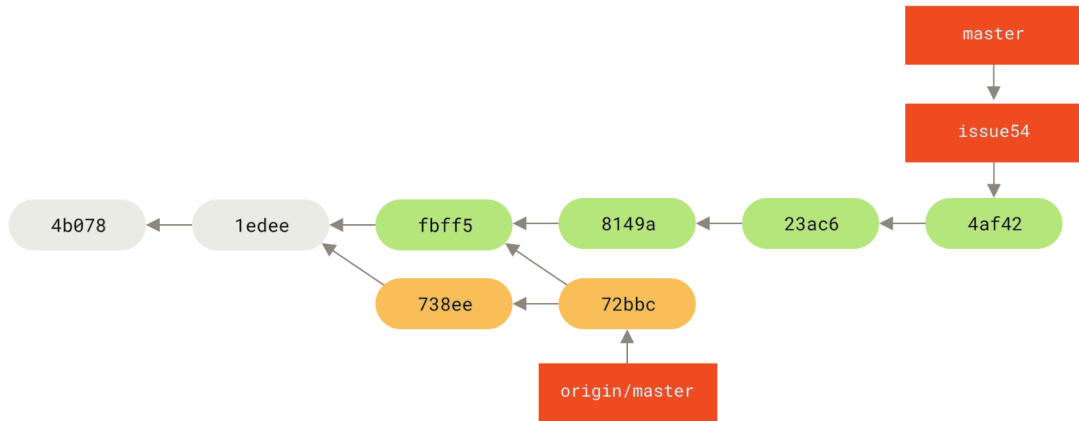
Repo di Alice - fetch

Alice fa il fetch del lavoro di Bob



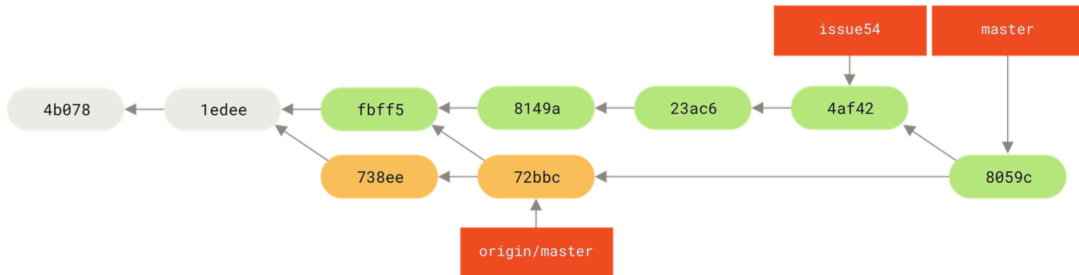
Repo di Alice - merge fast-forward

Alice fa il merge fast-forward di `issue54` in `master`



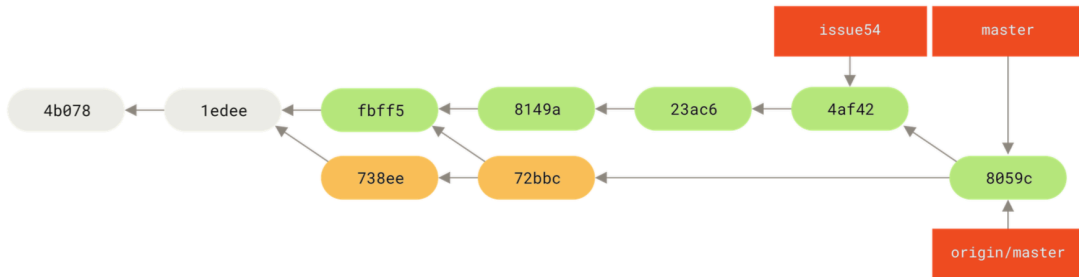
Repo di Alice - merge 3-way

Alice fa il merge 3-way di origin/master in master

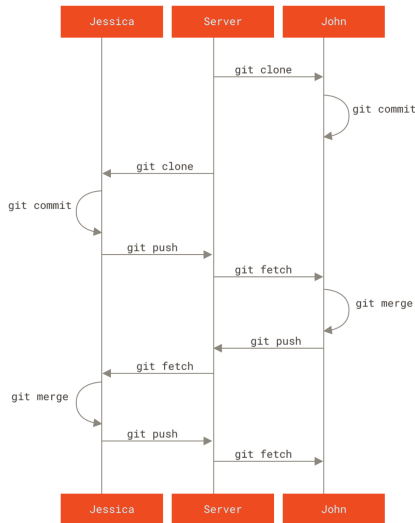


Repo di Alice - push

Alice fa il push di master



Esempio di workflow (riassunto)



Grazie per la vostra attenzione

Elisabetta Ferri, Giorgio Micaglio, Gianmarco Puleo, Michele Tognoni

Associazione Italiana Studenti di Fisica
Comitato Locale di Trento

May 15, 2024