

# Git per lo sviluppo collaborativo

Ingegneria del Software  
A.A. 2020/2021

Prof. Andrea Fornaia

# Cos'è Git

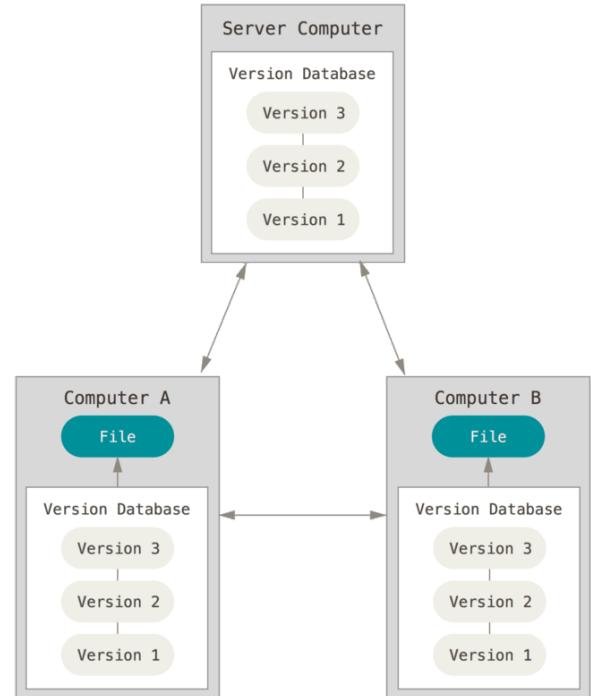
- Git è uno strumento per **controllo di versione** (Version Control System, VCS) ovvero per tener traccia delle modifiche fatte ad un insieme di file nel tempo
- Molto usato dai programmatori per coordinare le modifiche fatte al codice durante il **processo di sviluppo di un progetto software**
- Con Git è possibile:
  - Esaminare ed eventualmente ripristinare lo stato del progetto in un momento precedente
  - Mostrare le differenze tra due stati del progetto
  - Dividere la scrittura del codice di un progetto in linee di sviluppo indipendenti, chiamate “branch”, che evolvono separatamente
  - Riunire periodicamente le modifiche fatte in più branch di sviluppo, tramite operazioni di “merge”
  - Permettere a più sviluppatori di lavorare simultaneamente allo stesso progetto, condividendo e combinando il loro lavoro

# Le origini di Git

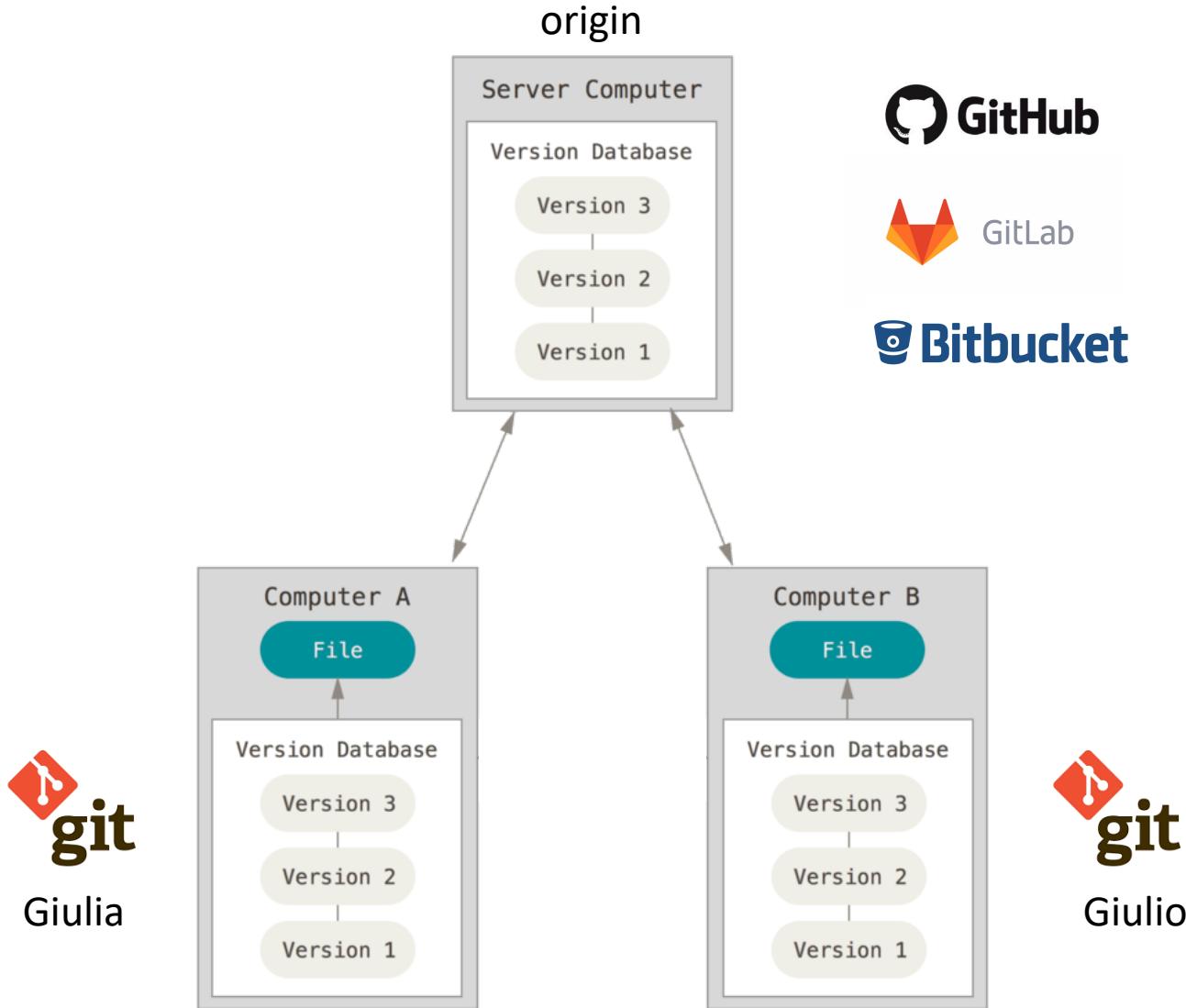
- Le origini di Git sono legate allo sviluppo del **Kernel Linux**:
  - Inizialmente le modifiche venivano applicate tramite patch (1991)
  - Successivamente si è passati all'uso di un sistema di controllo di versione distribuito (DVCS) proprietario, chiamato BitKeeper (2002)
  - In fine, Linus Torvalds e la comunità degli sviluppatori di Linux decisero di creare il loro DVCS, ovvero Git

# Distributed VCS

- Nei sistemi di controllo di versione distribuiti (DVCS) ogni sviluppatore ha una copia completa della storia di un progetto , chiamata “repository”, su cui ha il pieno controllo
- L'accesso alla rete è quindi necessario solo quando si vuole condividere il proprio lavoro con gli altri sviluppatori
- Seppure Git non lo imponga, tipicamente gli sviluppatori condividono un repository centralizzato intermedio con cui sincronizzare il proprio lavoro, ospitato da servizi come GitHub, GitLab o BitBucket



# Scenario tipico



# Installare Git

```
$ git --version  
git version 2.30.1 (Apple Git-130)
```

- Mac
  - via Xcode.
  - Installer:
    - <https://sourceforge.net/projects/git-osx-installer/files/>
  - Homebrew:
    - \$ sudo brew install git
  - MacPort:
    - \$ sudo port install git
- Windows
  - Installer:
    - <https://git-for-windows.github.io>
- Linux
  - Package manager:
    - \$ sudo apt-get install git

# Configurazione di Git

```
$ git config --global user.name "Andrea Fornaia"  
$ git config --global user.email fornaia@example.com  
  
$ git config --list  
user.name=Andrea Fornaia  
user.email=fornaia@example.com  
...  
  
$ git config user.name  
Andrea Fornaia
```

- Al primo utilizzo, è necessario fornire una configurazione minima, indicando quale **user.name** e **user.email** usare globalmente (per tutti i repository git creati dall'utente sul pc)
- Queste informazioni verranno usate durante il salvataggio delle versioni (commit) di un progetto per indicare l'autore del salvataggio
- Un *commit* è un salvataggio dello stato del progetto

# Command Line di Git

```
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
```

These are common Git commands used in various situations:

- start a working area** (see also: `git help tutorial`)
  - `clone` Clone a repository into a new directory
  - `init` Create an empty Git repository or reinitialize an existing one
- work on the current change** (see also: `git help everyday`)
  - `add` Add file contents to the index
  - `mv` Move or rename a file, a directory, or a symlink
  - `restore` Restore working tree files
  - `rm` Remove files from the working tree and from the index
  - `sparse-checkout` Initialize and modify the sparse-checkout
- examine the history and state** (see also: `git help revisions`)
  - `bisect` Use binary search to find the commit that introduced a bug
  - `diff` Show changes between commits, commit and working tree, etc
  - `grep` Print lines matching a pattern
  - `log` Show commit logs
  - `show` Show various types of objects
  - `status` Show the working tree status
- grow, mark and tweak your common history**
  - `branch` List, create, or delete branches
  - `commit` Record changes to the repository
  - `merge` Join two or more development histories together
  - `rebase` Reapply commits on top of another base tip
  - `reset` Reset current HEAD to the specified state
  - `switch` Switch branches
  - `tag` Create, list, delete or verify a tag object signed with GPG
- collaborate** (see also: `git help workflows`)
  - `fetch` Download objects and refs from another repository
  - `pull` Fetch from and integrate with another repository or a local branch
  - `push` Update remote refs along with associated objects

# GUI, IDE e Command Line

- Git mette a disposizione un'interfaccia a riga di comando molto completa
- Git è uno strumento complesso, con molti comandi e molti modi diversi per raggiungere obiettivi simili: l'uso della command line (per quanto *incoraggiato*) potrebbe... *scoraggiare*
- Esistono degli strumenti visuali (GUI) per poter operare su un repository git
  - GitKraken
  - Sourcetree
  - GitHub Desktop
- Client Git sono comunemente integrati negli IDE (ambienti di sviluppo integrati) come in Eclipse, IntelliJ e VS Code

# GitKraken

The screenshot shows the GitKraken application interface for the `electron` repository on the `master` branch. The main view displays a timeline of commits, each represented by a vertical line connecting the commit author and the commit message. The commits are color-coded by author: Samuel Maddock (SM, teal), GitHub (G, blue), and others (purple, pink, yellow). The commit details include the author, date, and a list of changes. A sidebar on the left shows repository statistics: 1181 issues, 131 pull requests, and 1049 tags. The bottom right corner indicates the user has a **PRO** license.

**repository:** electron    **branch:** master

**Issues:** Select an issue tracker for this repo

- GitHub (checked)
- GitHub Enterprise
- GitKraken Boards (checked)
- GitLab
- GitLab Self-Managed
- Jira Cloud (checked)
- Jira Server
- Trello
- None

**TAGS:** 1049/1049    **SUBMODULES:** 2    **GITHUB ACTIONS:** 0

**Commits:**

- feat(extensions): expose ExtensionRegistryObserver events in Session (#25385) - SM (9d0d9a)
- update patches - AL (commit 881ac9)
- update printing.patch given print settings mojoific... - AL (commit 881ac9)
- fix: provide asynchronous cleanup hooks in n-api - SV (commit 9d0d9a)
- fix: update node certdata to NSS 3.56 (#25362) - T (commit 9d0d9a)
- fix: update node certdata to NSS 3.56 (#25361) - T (commit 9d0d9a)
- fix: check for destroyed webcontents in converter (...) - JR (commit 9d0d9a)
- docs: remove unused StreamProtocolResponse / S... - T (commit 9d0d9a)
- docs: remove unused StreamProtocolResponse / S... - T (commit 9d0d9a)
- fix: order menu items before filtering excess separ... - T (commit 9d0d9a)
- fix: decompress devtools discovery html - JR (commit 9d0d9a)
- fix: decompress devtools discovery html - JR (commit 9d0d9a)
- fix: decompress devtools discovery html (#25576) - JR (commit 9d0d9a)
- fix: honor pageRanges when printing - SV (commit 9d0d9a)
- Address extra null checks & update test - SM (9d0d9a)
- fix: order menu items before filtering excess separ... - SA (commit 9d0d9a)
- 2418471: PDF Viewer update: Add missing aria-lab... - JK (commit 9d0d9a)
- fix: honor pageRanges when printing - SV (commit 9d0d9a)
- 2387901: Accessing C++ enums in Java ... - JK (commit 9d0d9a)
- update patches - JK (commit 9d0d9a)
- printing-ranges-honor... - EB (commit 9d0d9a)
- roller/chromium/11-x-y - EB (commit 9d0d9a)
- v12.0.0-nightly.20200923 - EB (commit 9d0d9a)
- roller/chromium/10-x-y - EB (commit 9d0d9a)
- 9-x-y - EB (commit 9d0d9a)
- fix: unsubscribe from observers when window is cl... - T (commit 9d0d9a)
- fix: unsubscribe from observers when window is cl... - T (commit 9d0d9a)
- fix: unsubscribe from observers when window is cl... - T (commit 9d0d9a)
- chore: bump chromium in DEPS to ddb5b6db5e35... - FB (commit 9d0d9a)
- chore: bump node in DEPS to v14.12.0 - FB (commit 9d0d9a)

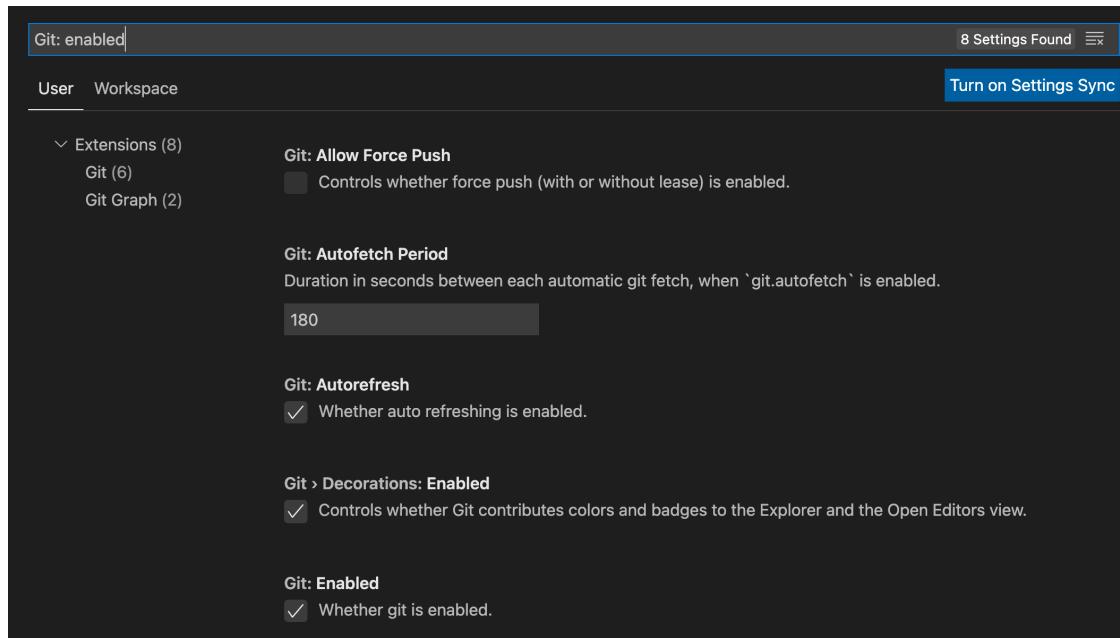
**Commit Details:** commit: 9d0d9a

**Author:** Samuel Maddock  
authored 9/23/2020 @ 12:29 PM  
GitHub committed 9/23/2020 @ 12:29 PM

**Changes:** 4 modified

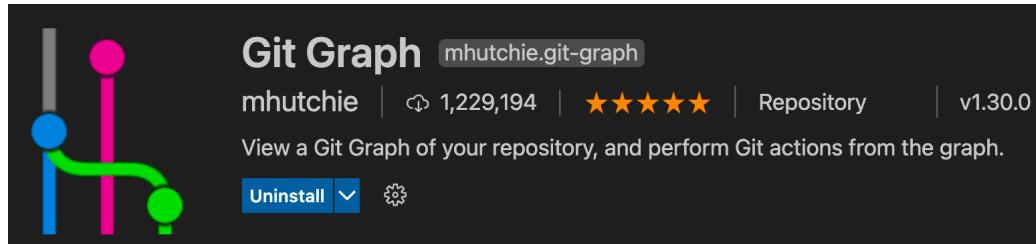
- docs/api/session.md
- shell/browser/api/electron\_api\_session.cc
- shell/browser/api/electron\_api\_session.h
- spec-main/extensions-spec.ts

# Abilitare Git su VS Code



- VS Code viene fornito con un pratico supporto all'uso di git. Non è necessario installare estensioni aggiuntive per un uso di base
- Nei nostri esempi, useremo anche l'estensione Git Graph (da installare)
- Dopo aver installato Git, è necessario verificare che VS Code sia abilitato al suo utilizzo
  - (Windows/Linux) File > Preferences > Settings
  - (Mac) Code > Preferences > Settings
  - Scrivere Git: Enabled nella barra di ricerca
  - Assicurarsi che la check box sia segnata

# (Opzionale) Aggiungere l'estensione Git Graph

A screenshot of the VS Code interface with the "Git Graph" tab active. The sidebar shows "README.md", "Git Graph", and ".gitignore". The main area displays a git commit history. On the left is a small graph visualization showing branches and commits. The commits are listed in a table with columns: Graph, Description, Date, Author, and Commit. The commits are:

Graph	Description	Date	Author	Commit
o ↗ master   origin ↗ origin/HEAD	Merge branch 'master' of https://github...	28 Apr 2021 22:31	Andrea Fornai	eb342dcd
	README.md aggiornato da VS Code	28 Apr 2021 22:26	Andrea Fornai	fc844041
	Aggiornato README.md da GitHub	28 Apr 2021 22:25	Andrea Fornai	81048355
	README aggiornato	28 Apr 2021 22:18	Andrea Fornai	5e48b34a
	Initial commit	28 Apr 2021 22:10	Andrea Fornai	cf89c72f

- Git Graph estende le funzionalità già offerte in maniera nativa da VS Code, aggiungendo la visualizzazione della storia delle versioni (commit) tramite grafo
- Molto utile per capire meglio il funzionamento interno di git e il modo in cui organizza i commit e opera sui branch di sviluppo
- Utile anche per ripercorrere la storia del progetto

# Creare un repository su GitHub

- GitHub è un servizio per la condivisione del codice e lo sviluppo collaborativo
- Permette di creare dei repository git centrali *remoti* con cui gli sviluppatori possono condividere il proprio codice e sincronizzare le modifiche apportate da altri sviluppatori
- È necessario creare un account. La mail usata per iscriversi dovrebbe combaciare con quella indicata nel campo di configurazione **user.email** (non per motivi di autenticazione, ma per associare visivamente l'autore del commit con il nome utente su GitHub)
- Dopo aver fatto il login è possibile creare un **repository**
  - Pubblico: il codice sarà visibile a chiunque (solo in lettura)
  - Privato: il codice sarà visibile solo agli utenti da noi autorizzati

# Creazione Repository Privato

## Create a new repository

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

Owner \*      Repository name \*



afornia

/ tutorial



Great repository names are short and memorable. Need inspiration? How about [curly-octo-palm-tree](#)?

Description (optional)

Un semplice repository creato per un tutorial



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:

Skip this step if you're importing an existing repository.

Add a README file

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

Add .gitignore

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

.gitignore template: Java

Choose a license

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

This will set `master` as the default branch. Change the default name in your [settings](#).

[Create repository](#)

Necessario specificare un nome univoco nel nostro namespace e il livello di visibilità (Public o Private). È possibile aggiungere una breve descrizione

Un **README** file serve a fornire una pagina iniziale di descrizione e documentazione del progetto. Viene usato il linguaggio di markup **Markdown**

Un **.gitignore** è un file nascosto che indica quali file non è necessario condividere, come file di log o file binari generati dalla compilazione del codice. GitHub fornisce dei template pronti specifici per ciascun linguaggio (es. per Java verranno ignorati i **.class**)

# Risultato

master ▾ 1 branch 0 tags

Go to file Add file ▾ Code ▾

 **afornaia** Initial commit cf89c72 now 1 commit

 .gitignore Initial commit now

 README.md Initial commit now

**README.md** 

**tutorial**

Un semplice repository creato per un tutorial

**About** 

Un semplice repository creato per un tutorial

 Readme

---

**Releases**

No releases published [Create a new release](#)

---

**Packages**

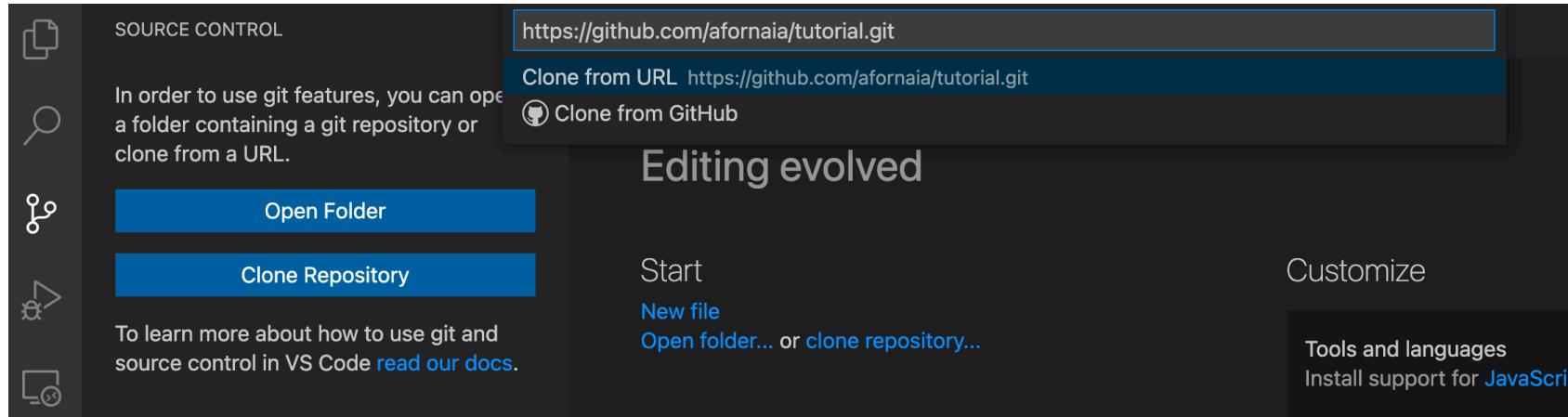
No packages published [Publish your first package](#)

# Clonare un Repository remoto

The screenshot shows a GitHub repository page for 'aforaia/tutorial'. The top navigation bar includes 'master', '1 branch', '0 tags', 'Go to file', 'Add file', and a green 'Code' button with a dropdown arrow. The dropdown menu is open, showing 'Clone' options: 'HTTPS' (selected), 'SSH', and 'GitHub CLI'. Below this is a URL: 'https://github.com/aforaia/tutorial.g'. A note says 'Use Git or checkout with SVN using the web URL.' Other options in the dropdown are 'Open with GitHub Desktop' and 'Download ZIP'. To the right of the dropdown, there's an 'About' section with a brief description: 'Un semplice repository creato per un tutorial', a 'Readme' link, and sections for 'Releases' (no releases) and 'Packages' (no packages). The main repository area shows files like '.gitignore', 'README.md', and 'tutorial'. The 'tutorial' file contains the text: 'Un semplice repository creato per un tutorial'.

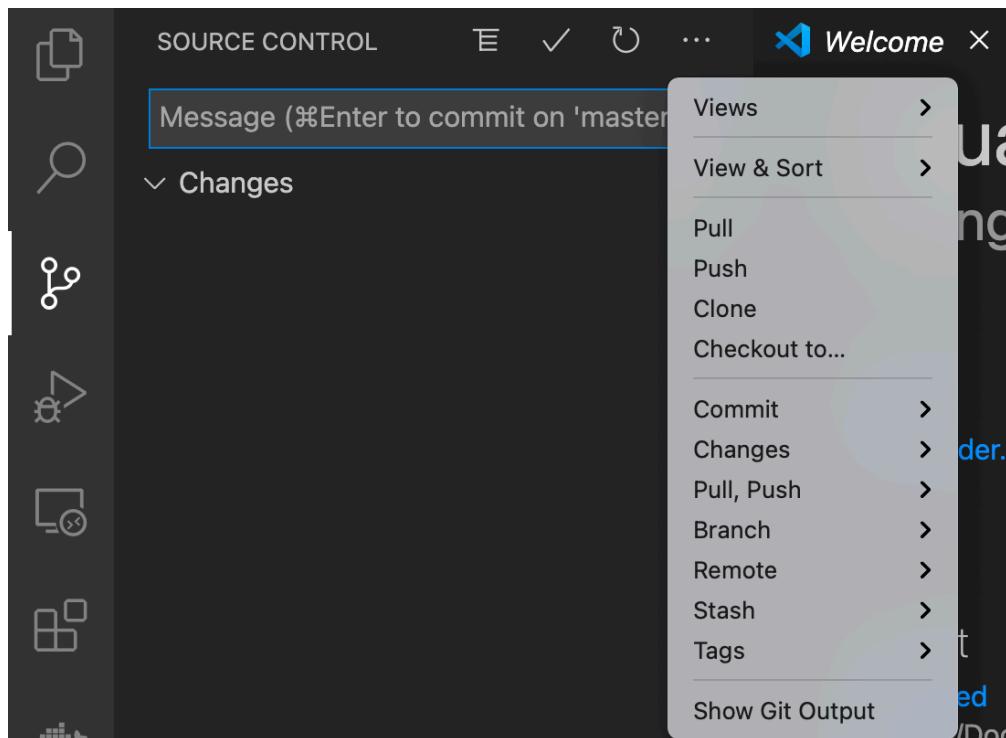
- Per clonare un repository remoto, ovvero creare una copia locale sul nostro pc per lo sviluppo, è necessario indicare il riferimento al repository (URL)
- Esistono due modi principali: tramite HTTP e SSH
  - HTTP: in fase di sincronizzazione tra il repository locale e quello remoto vengono chieste le credenziali di GitHub (tipicamente solo la prima volta, poi vengono salvate dal sistema operativo)
  - SSH: permette le operazioni di sincronizzazione senza password, tramite l'uso di chiavi pubbliche e private
- Per semplicità, vedremo il primo metodo

# Clone di un repository con VS Code



- Dalla barra laterale selezionare l'icona “source control” e poi “Clone Repository”
- Incollare l’URL copiata precedentemente da GitHub
- Richiederà una cartella dove clonare il repository
- Nella cartella verrà scaricato la copia completa del repository remoto (presente nella cartella nascosta `.git`) è verranno caricati i file del progetto presenti nel commit più recente sul branch principale (**master/main**)
- La cartella con dentro i file della versione su cui stiamo lavorando viene detta **working directory**

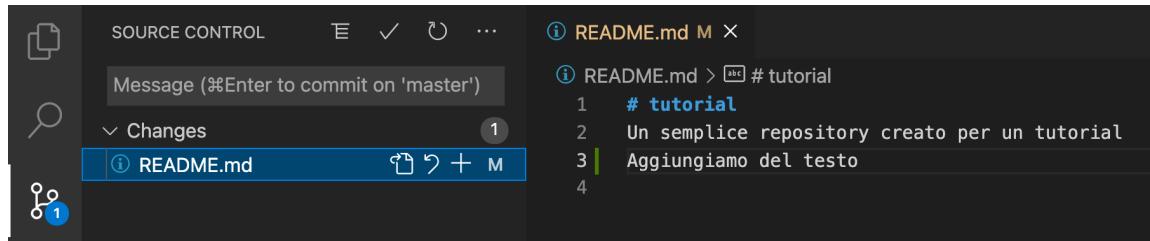
# Comandi disponibili



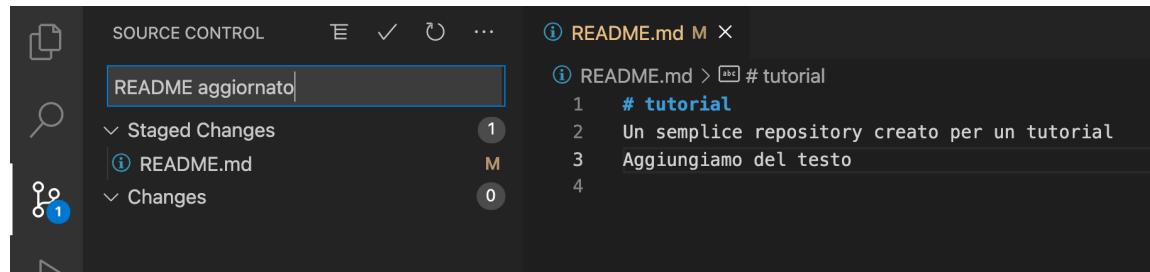
- I comandi più comunemente usati sono richiamabili da interfaccia
- Di questi vedremo i più importanti tra quelli di base
  - Clone (già visto)
  - Commit
  - Pull, Push (sincronizzazione)
  - Branch
  - Checkout

# Creiamo un commit

- Modifichiamo un file, es. README.md. Queste modifiche verranno evidenziate ed elencate nella sezione **Changes** che comprende le modifiche apportate ai file ma non ancora aggiunte nell'area di **staging**

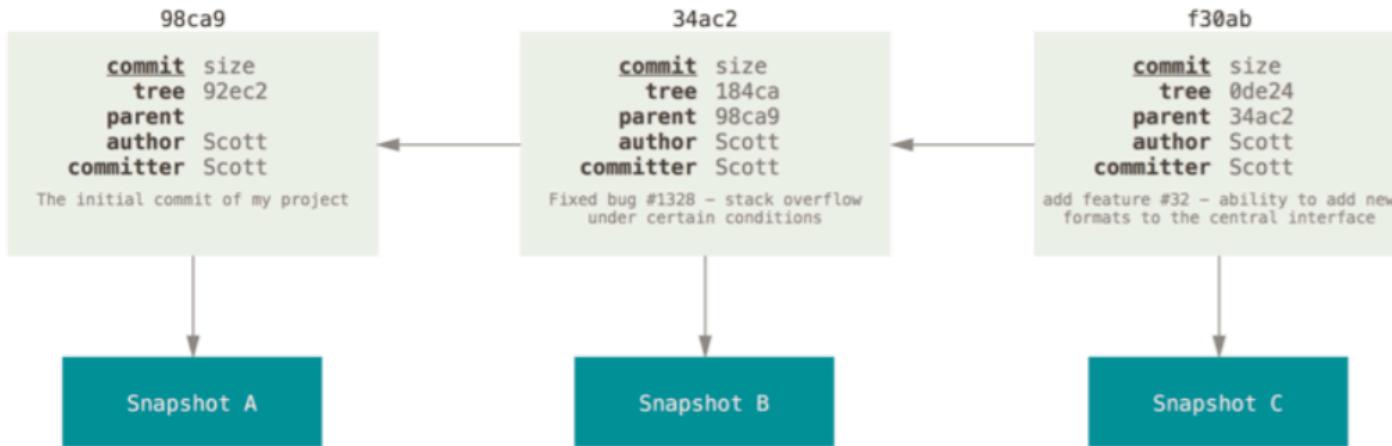


- La "M" sta per "modificato". Per fare in modo che questa modifica venga aggiunta al prossimo commit deve essere **esplicitamente** aggiunto all'area *di preparazione* (staging area)



- Aggiungiamo un messaggio di commento **descrittivo e significativo** per il commit (Es. README aggiornato, ma sarebbe meglio README aggiornato con del testo d'esempio)
- Completiamo il commit con CMD + INVIO (CTRL + INVIO per Windows/Linux)

# Struttura di un Commit



- **Snapshot (tree):** una struttura gerarchica di file e cartelle (lo stato aggiornato del progetto)
- **Author:** nome, email, ora e data (o “timestamp”) di chi ha fatto il commit
- **Committer:** chi ha aggiunto questo commit al repository (in certi casi può differire dall’autore)
- **Message:** testo usato per commentare le modifiche apportate dal commit
- **Lista di parent (0, 1 or 2):** stati immediatamente precedenti da cui deriva il commit
  - Root: 0, Normal Commit: 1, Merge Commit: 2
- Ogni commit ha un **identificativo univoco** che è l'**hash SHA-1** del commit stesso, es. 98ca9 è il prefisso (non ambiguo) dell’id del commit (e quindi dell’hash, che sarebbe in realtà più lungo)

# Carichiamo le modifiche su GitHub

- Al momento il nostro repository locale ha un commit in più sul branch principale (master) rispetto al repository remoto. Questo viene riassunto in basso a sinistra in VS Code
- Nell'esempio, master è 0 commit indietro e 1 commit avanti rispetto al master su GitHub

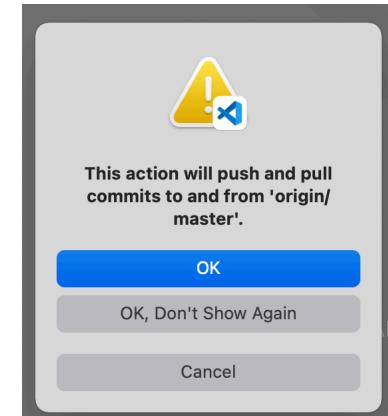


- Per caricare le modifiche dobbiamo **sincronizzare** il branch master tra i repository (locale e remoto) premendo l'icona a due frecce circolari
- Questo in realtà causerà due operazioni:
  - Il download di eventuali commit presenti sul master remoto (**pull**)
  - L'upload dei commit presenti in locale (**push**)

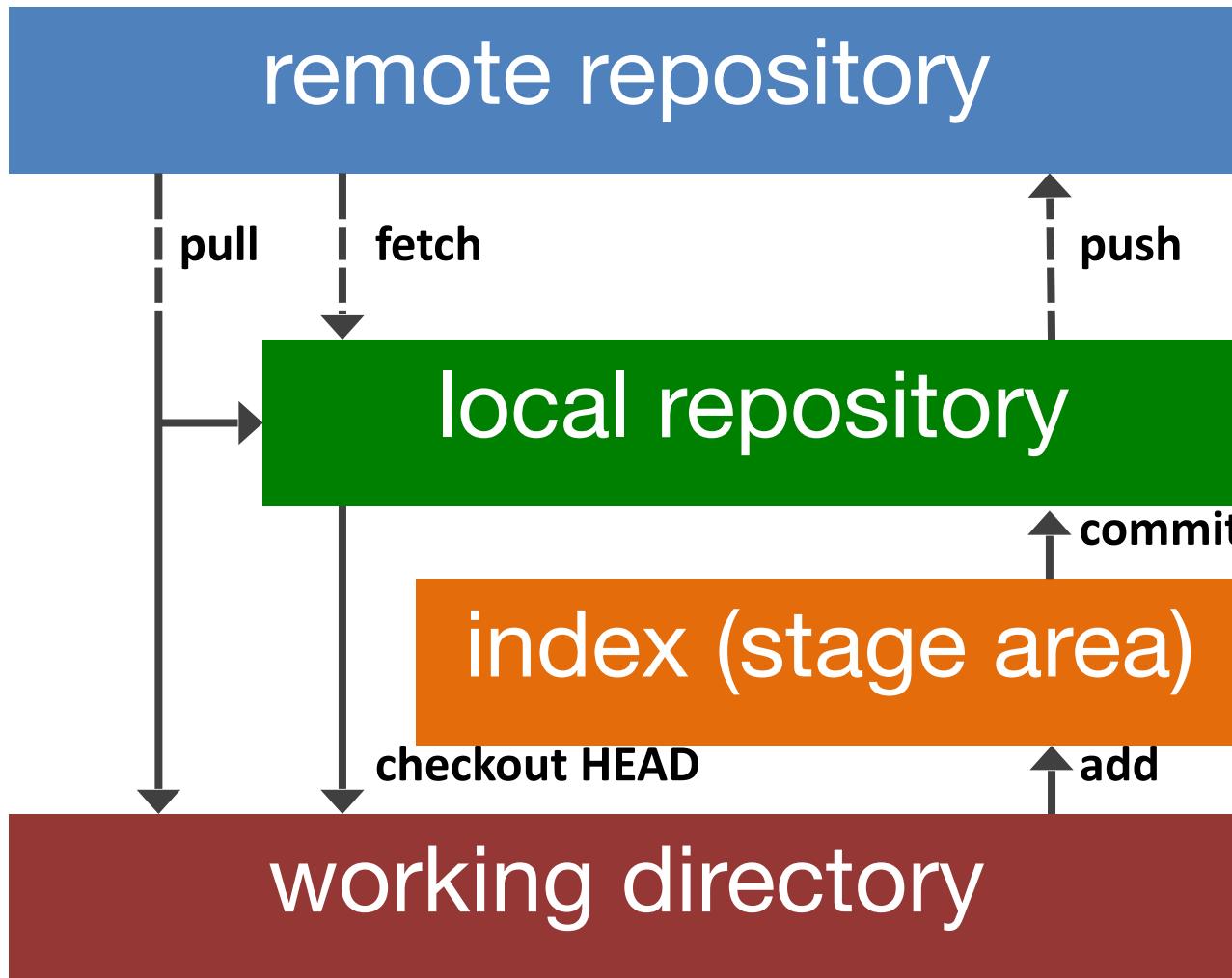
The screenshot shows the GitHub desktop application interface. At the top, it displays the repository name 'aforaia' and the branch 'master'. Below this, the commit history is shown:

- aforaia README aggiornato (5e48b34, 5 minutes ago, 2 commits)
- .gitignore Initial commit (13 minutes ago)
- README.md README aggiornato (5 minutes ago)

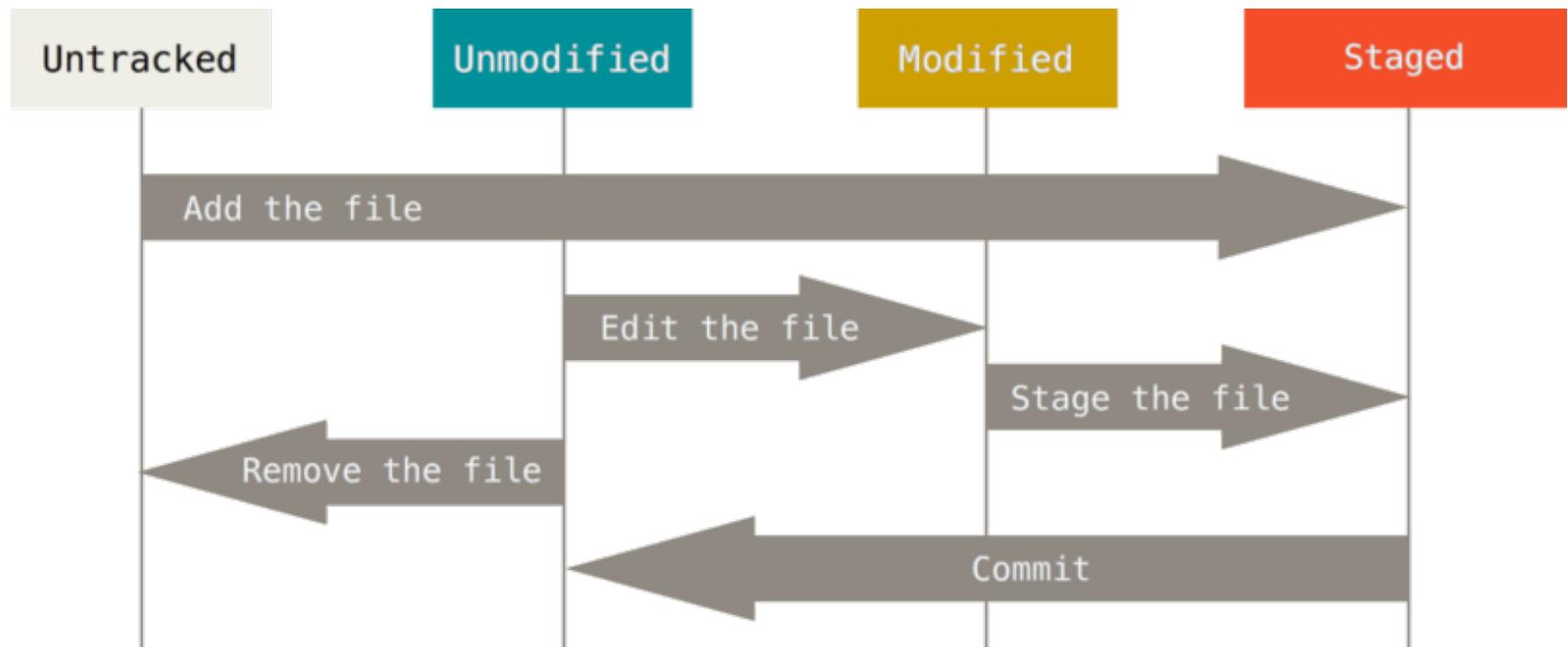
On the left, there's a file tree with 'README.md' and a 'tutorial' folder. The 'tutorial' folder contains a file named 'testo'. A note at the bottom states: "Un semplice repository creato per un tutorial Aggiungiamo del testo". On the right side of the interface, there are sections for 'About', 'Releases', and 'Packages'.



# Riepilogo Workflow



# Stati di un file



# Il Git Graph

- Il git graph (o commit graph) è un grafo diretto e aciclico della storia dei commit del nostro progetto
- I commit possono avere degli archi uscenti verso i commit parent
- Il grafo viene completato dai **branch**, ovvero dei puntatori che indicano l'ultimo commit presente in quel ramo di sviluppo. **HEAD** punta invece al commit attivo, ovvero quello caricato nella working directory. Quelli che iniziano con **origin/** indicano la posizione del **branch in remoto** (GitHub)
- La posizione dei branch viene aggiornata con le operazioni di commit
- Es. questo è lo stato del commit graph appena dopo aver fatto il commit ma prima del push su GitHub

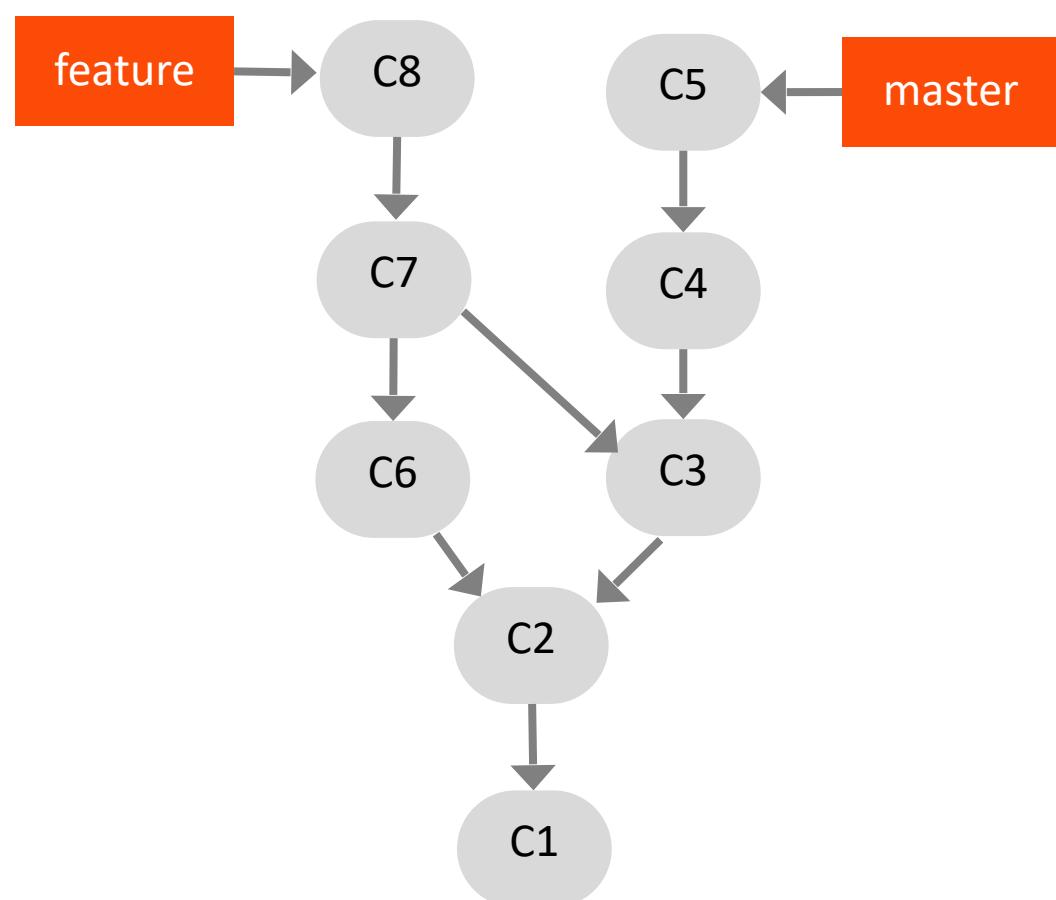
Branches:		Show All	<input checked="" type="checkbox"/> Show Remote Branches					
Graph	Description			Date	Author	Commit		
	master	origin/HEAD	origin/master	README aggiornato	28 Apr 2021 22:18	Andrea Fornaia	5e48b34a	
				Initial commit	28 Apr 2021 22:10	Andrea Fornaia	cf89c72f	

- Questo è lo stato dopo il push: origin/master che era indietro di un commit, ora è invece allo stesso livello del branch master. Anche HEAD è stato aggiornato, ad indicare che la versione nella working directory è proprio l'ultima versione presente su master

Branches:		Show All	<input checked="" type="checkbox"/> Show Remote Branches					
Graph	Description			Date	Author	Commit		
	master	origin	origin/HEAD	README aggiornato	28 Apr 2021 22:18	Andrea Fornaia	5e48b34a	
				Initial commit	28 Apr 2021 22:10	Andrea Fornaia	cf89c72f	

# Branch – History non lineare

Git supporta ed incoraggia uno sviluppo non-lineare. I commits possono essere rappresentati con un **DAG**



Un commit può avere:

- Nessun predecessore (first commit, **root**)
- Un predecessore (**normal** commit)
- Due predecessori (**merge** commit)

Possiamo avere più branch, in questo esempio:

- master {C1; C2; C3; C4; C5}
- feature {C1; C2; C3; C6; C7; C8}

Notare che i **possono sovrapporsi**

I riferimenti **non sono bidirezionali**: cancellando un branch renderei alcuni commit irraggiungibili, e verranno cancellati in seguito da un **garbage collector**

# Modifichiamo il file in remoto

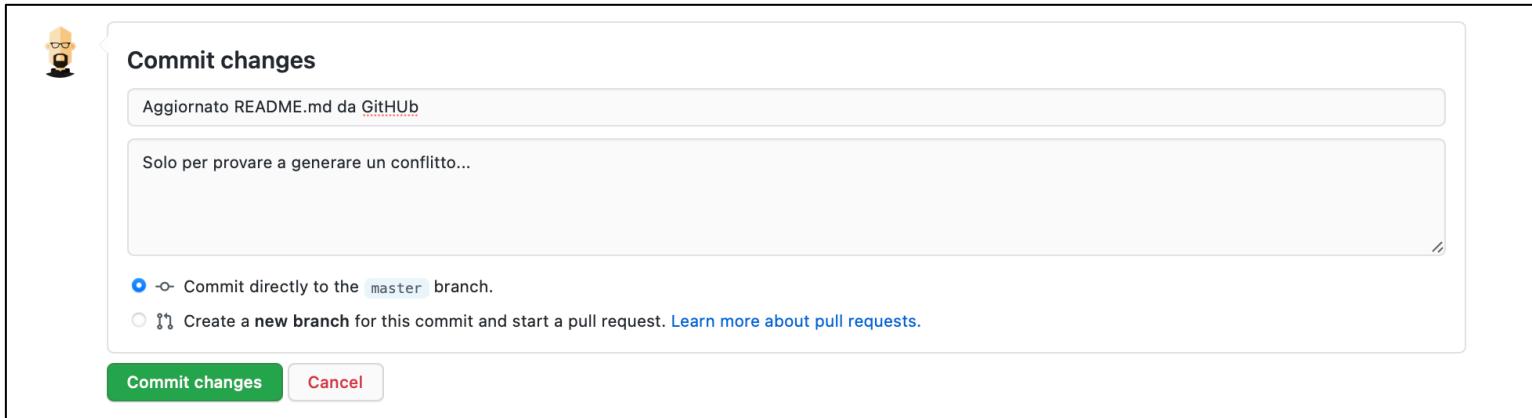
- Per simulare delle modifiche fatte da un altro sviluppatore e caricate sul repository remoto, modifichiamo il file README direttamente da GitHub e creiamo un nuovo commit sul master remoto



The screenshot shows the GitHub interface for editing the README.md file in the 'tutorial' repository. The file content is as follows:

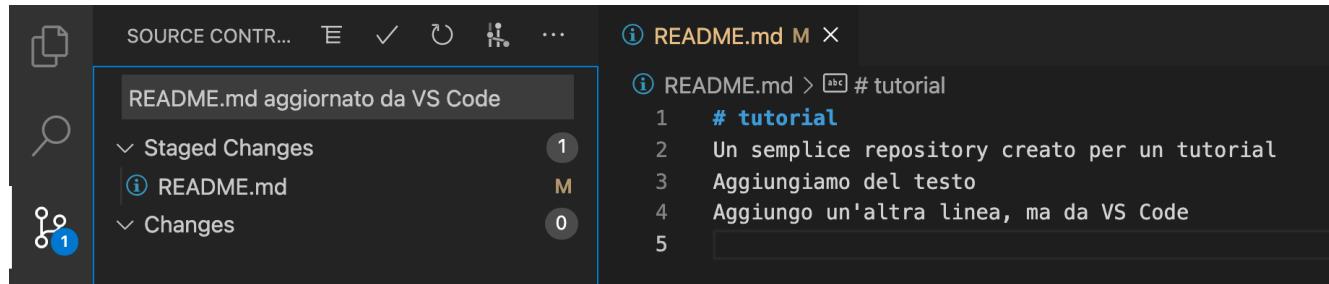
```
1 # tutorial
2 Un semplice repository creato per un tutorial
3 Aggiungiamo del testo
4 Aggiungo un'altra linea, ma dà GitHub
5
```

At the top, there are buttons for 'Edit file' and 'Preview changes'. On the right, there are settings for 'Spaces' (set to 2), 'Soft wrap', and a 'Cancel changes' button.



# In contemporanea, modifichiamo lo stesso file in locale

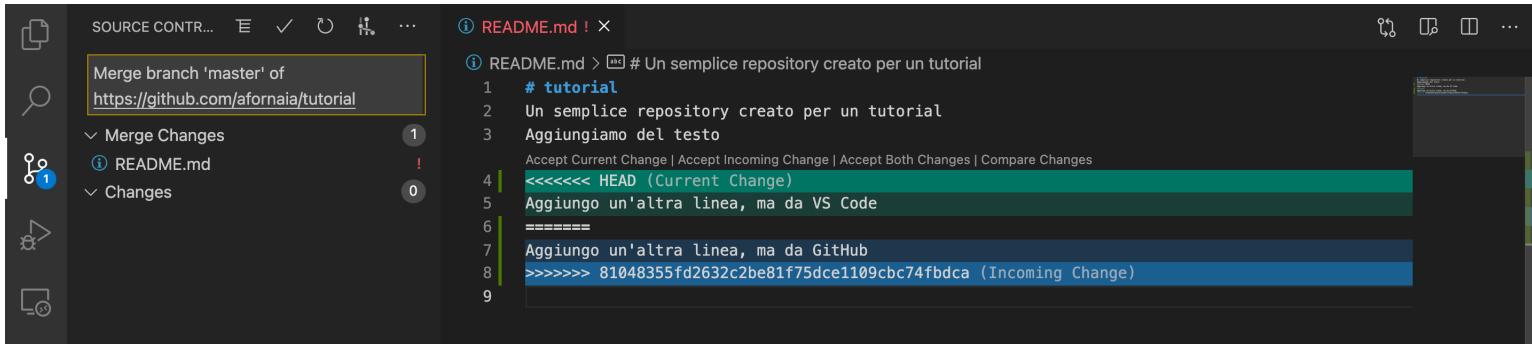
- Modifichiamo il README in locale con una riga diversa. Nota, dal momento che non abbiamo ancora sincronizzato la copia locale con quella remota (pull) non abbiamo ancora le modifiche apportate in remoto



The screenshot shows the VS Code interface with a dark theme. On the left is the sidebar with icons for file, search, and repository. The main area shows a file tree with 'README.md aggiornato da VS Code' at the top, indicating it's been modified locally. Under 'Staged Changes', there is a single file 'README.md' with a status 'M'. The right panel displays the contents of 'README.md':

```
# tutorial
Un semplice repository creato per un tutorial
Aggiungiamo del testo
Aggiungo un'altra linea, ma da VS Code
```

- Creando un commit e facendo il sync (pull e push) git si accorgerà che il master locale è indietro di un commit, e prima di caricare in remoto le modifiche locali, cercherà in automatico di **unire (merge)** le modifiche del commit fatto in remoto con quelle del commit fatto in locale
- Avendo entrambi modificato la stessa porzione dello stesso file, git non riuscirà a fare il merge in automatico, dando origine ad un **conflitto**

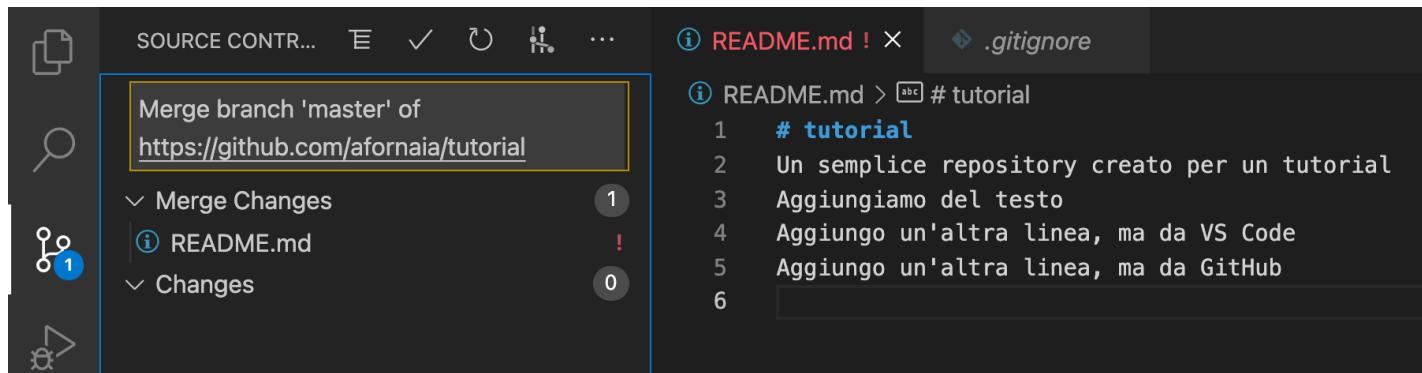


The screenshot shows the VS Code interface with a dark theme. The file tree on the left shows a 'Merge branch 'master'' notification with a link to 'https://github.com/aforaia/tutorial'. Under 'Merge Changes', there is a file 'README.md' with a status '!' and a 'Changes' section with a status '0'. The right panel shows the content of 'README.md' with a merge conflict:

```
# tutorial
Un semplice repository creato per un tutorial
Aggiungiamo del testo
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<< HEAD (Current Change)
Aggiungo un'altra linea, ma da VS Code
=====
Aggiungo un'altra linea, ma da GitHub
>>>>> 81048355fd2632c2be81f75dce1109cbc74fbda (Incoming Change)
```

# Risolvere un conflitto

- Per risolvere un conflitto bisogna decidere il modo in cui unire le modifiche apportate dai due commit, e poi creare un commit
- Questo commit, essendo originato dall'unione di due commit, avrà due parent e viene detto **merge commit**



The screenshot shows the VS Code interface during a merge process. On the left, there's a sidebar with icons for file operations like copy, search, and history. The main area has a dark background with several tabs open. One tab is titled "Merge branch 'master' of https://github.com/aforaia/tutorial". Inside this tab, there's a message box containing the text "Merge branch 'master' of https://github.com/aforaia/tutorial". Below this message box, there are three items: "Merge Changes" (with 1 change), "README.md" (with ! conflict), and "Changes" (with 0 changes). To the right of the message box, the "README.md" file is shown with its content. The content includes a header "# tutorial", followed by four lines of text: "Un semplice repository creato per un tutorial", "Aggiungiamo del testo", "Aggiungo un'altra linea, ma da VS Code", and "Aggiungo un'altra linea, ma da GitHub". The line "Aggiungo un'altra linea, ma da GitHub" is preceded by a conflict marker "abc". The status bar at the bottom shows the current branch is "master".

- Nell'esempio, abbiamo deciso di tenere entrambe le modifiche. Creiamo un commit aggiungendo nella stagi area le modifiche apportate al file con il conflitto (segnato con !)
- Dopo aver creato il commit di merge bisogna fare il sync. Questo completa il merge in caso di conflitto.



# Repository remoto dopo il push

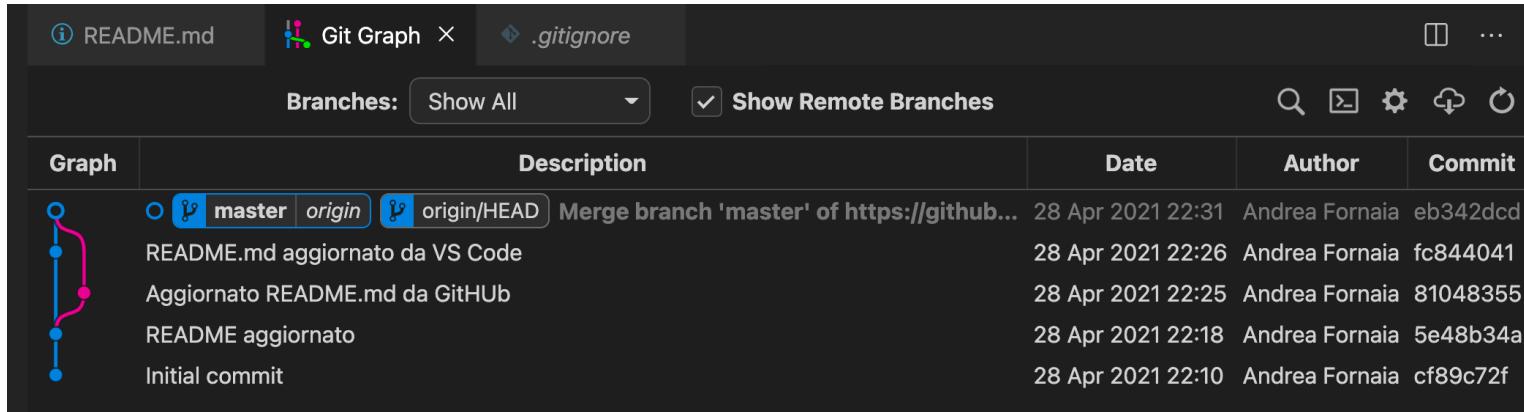
The screenshot shows a GitHub repository page for a file named 'README.md' in the 'tutorial' branch of the 'master' branch. The commit history shows a single merge commit from 'afornaia' that merges the 'master' branch into 'tutorial'. The commit message is 'Merge branch 'master' of <https://github.com/afornaia/tutorial>'. The commit was made 2 minutes ago with the SHA 'eb342dc'. There is 1 contributor listed. The file contains 5 lines (5 sloc) and 156 Bytes. The content of the file is:

```
tutorial

Un semplice repository creato per un tutorial Aggiungiamo del testo Aggiungo un'altra linea, ma da VS Code Aggiungo un'altra linea, ma da GitHub
```

At the bottom of the file content, there is a note: 'Un semplice repository creato per un tutorial Aggiungiamo del testo Aggiungo un'altra linea, ma da VS Code Aggiungo un'altra linea, ma da GitHub'

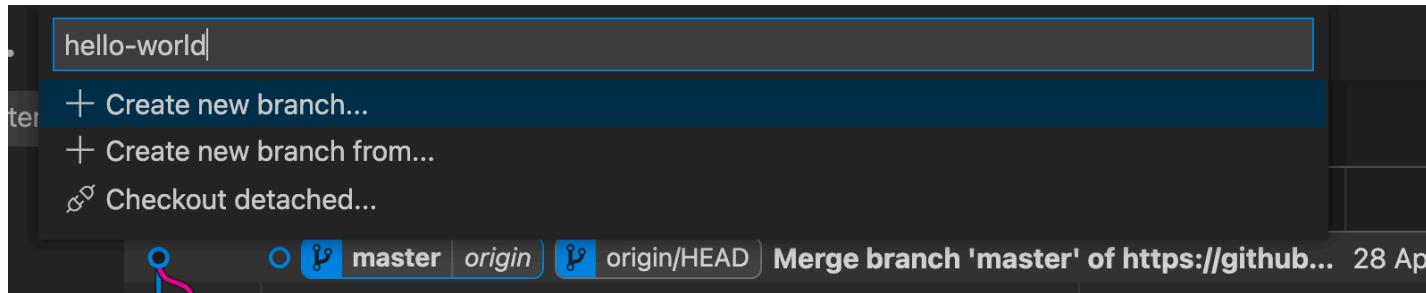
# Git graph dopo il merge



- Non sempre (per fortuna) i merge avvengono con conflitti. Per ridurre le situazioni di conflitto invece di lavorare sempre e solo sul branch master
- È buona norma creare dei **branch temporanei di sviluppo** usati per sviluppare una singola **feature**. Questi vengono detti **feature branch**.

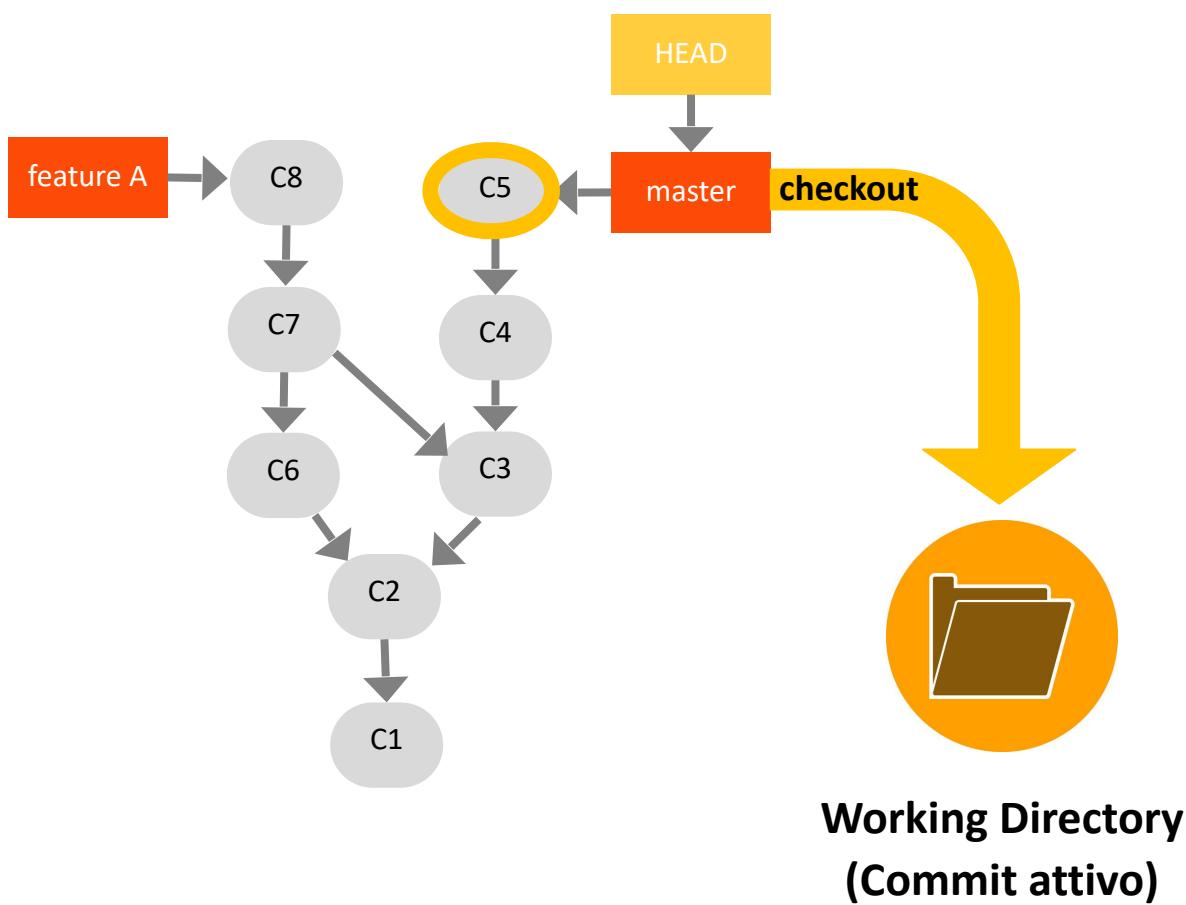
# Creare un feature branch

- Non sempre (per fortuna) i merge avvengono con conflitti. Per ridurre le situazioni di conflitto invece di lavorare sempre e solo sul branch master
- È buona norma creare dei **branch temporanei di sviluppo** usati per sviluppare una singola **feature**. Questi vengono detti **feature branch**.
- Supponiamo di voler creare una nuova feature, ovvero stampare un semplice “Hello World”
- Invece di fare le modifiche direttamente su master, facciamole su un branch nuovo, chiamato **hello-world**
- Nell’immagine sotto si può vedere il nuovo puntatore nel git graph



Graph	Description	Date	Author	Commit
	o <b>hello-world</b> o <b>master</b>   <b>origin</b> o <b>origin/HEAD</b> Merge branch 'master' ... 28 Apr 2021 22:31 Andrea Fornaia eb342dcd	28 Apr 2021 22:31	Andrea Fornaia	eb342dcd

# Un solo branch attivo per volta



Possiamo fare **switch** tra un branch ed un altro tramite il **checkout di un branch** senza perdere il nostro lavoro

HEAD è un puntatore speciale che punta al **branch attivo**

La working directory nel nostro file system conterrà il contenuto dello snapshot (file e cartelle) del **commit attivo**

# Commit sul feature branch

- Aggiunta la classe Main, creiamo un nuovo commit. Questo verrà aggiunto al branch **hello-world** mentre il branch master rimarrà immutato
- Il vantaggio dei branch è anche permettere di saltare velocemente da una versione ad un'altra (es. master) tramite **checkout**

The screenshot shows the VS Code interface. On the left is the Source Control sidebar with a tooltip 'Aggiunta classe Main' over the 'Staged Changes' section. The main editor window shows a Java file named Main.java with the following code:

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

The screenshot shows the GitHub commit history for the 'hello-world' branch. The commits are:

- Aggiunta classe Main (commit 392a6cb1, Andrea Fornaia, 28 Apr 2021 22:38)
- Merge branch 'master' of https://github.com/... (commit eb342dcd, Andrea Fornaia, 28 Apr 2021 22:31)
- README.md aggiornato da VS Code (commit fc844041, Andrea Fornaia, 28 Apr 2021 22:26)
- Aggiornato README.md da GitHub (commit 81048355, Andrea Fornaia, 28 Apr 2021 22:25)
- README aggiornato (commit 5e48b34a, Andrea Fornaia, 28 Apr 2021 22:18)
- Initial commit (commit cf89c72f, Andrea Fornaia, 28 Apr 2021 22:10)

# Modifiche in parallelo su branch diversi

- Se sviluppatori diversi lavorano su branch diversi, non ci sono conflitti fino a che non viene deciso di fare un merge
- Nell'esempio è stato aggiunto in remoto un commit su master. Dopo aver fatto il sync del master in locale questo avanza il branch master, non entrando in conflitto con le modifiche sul branch **hello-world**

The screenshot shows two side-by-side GitHub interface sections. On the left, under the path 'tutorial / Service.java in master', there is an 'Edit new file' section containing the following Java code:

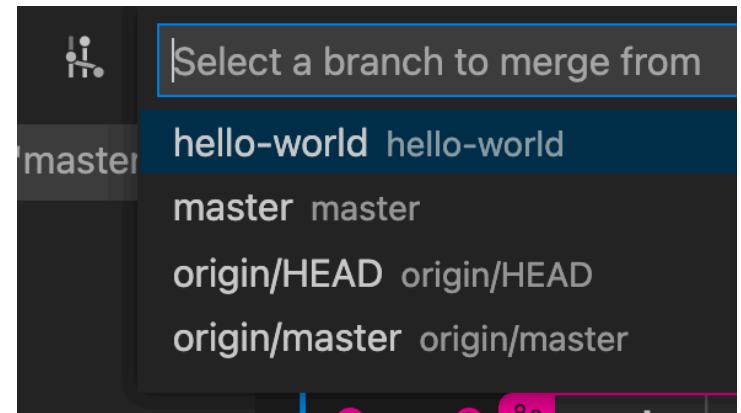
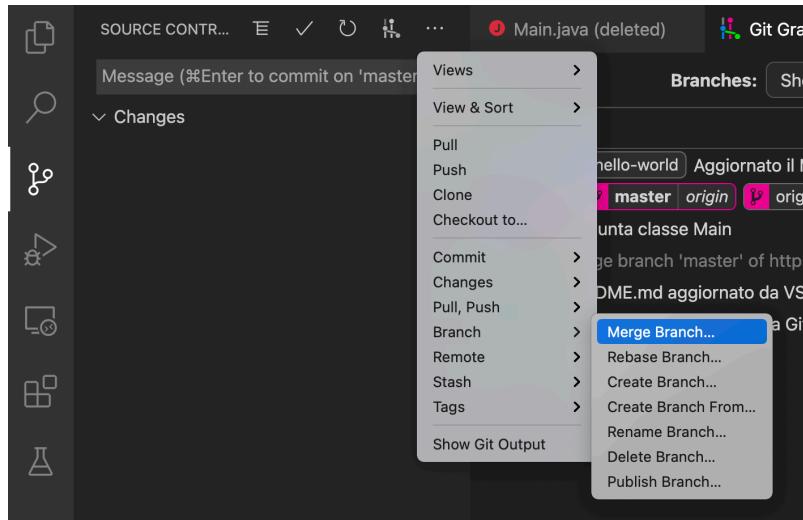
```
1 public class Service {  
2     public void run() {  
3         System.out.println("Service executed");  
4     }  
5 }
```

Below this is a 'Preview' section. On the right, a 'Commit new file' dialog box is open, showing the file name 'Service.java' and a note 'Create Service.java'. It includes an optional extended description field and two radio button options: one selected for 'Commit directly to the master branch' and another for 'Create a new branch for this commit and start a pull request'. At the bottom are 'Commit new file' and 'Cancel' buttons.

Graph	Description	Date	Author	Commit
	o master   origin   origin/HEAD Create Service.java h hello-world Aggiunta classe Main	28 Apr 2021 22:42	Andrea Fornaia	b12b344e
	Merge branch 'master' of https://github.com/afornaia/tutorial	28 Apr 2021 22:38	Andrea Fornaia	392a6cb1
	README.md aggiornato da VS Code	28 Apr 2021 22:31	Andrea Fornaia	eb342dcd
	Aggiornato README.md da GitHub	28 Apr 2021 22:26	Andrea Fornaia	fc844041
	README aggiornato	28 Apr 2021 22:25	Andrea Fornaia	81048355
	Initial commit	28 Apr 2021 22:18	Andrea Fornaia	5e48b34a
		28 Apr 2021 22:10	Andrea Fornaia	cf89c72f

# Merge del feature branch

- Quando le modifiche sul feature branch sono completate possiamo unirle al master
- Facciamo il **checkout** del branch **master** e successivamente il **merge** del branch **hello-world**
- Questo unirà le modifiche dei due commit, facendo avanzare il branch master (lasciando invece la posizione del branch hello-world invariata)



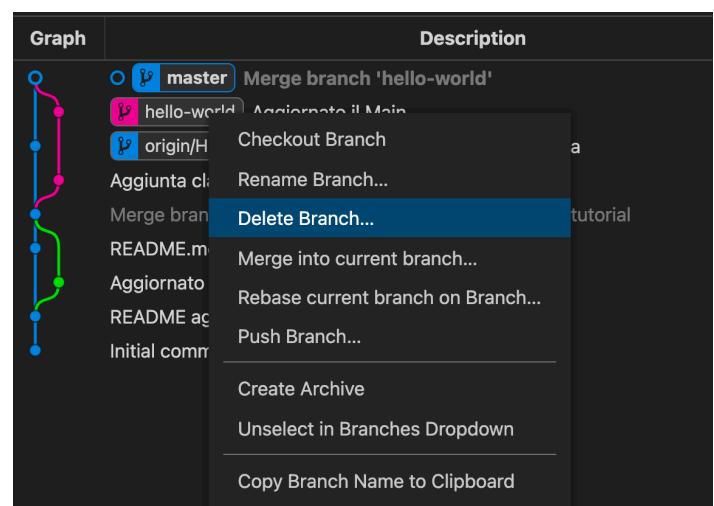
# Commit graph dopo il merge

The screenshot shows the VS Code interface with the Git Graph extension open. The Explorer sidebar on the left lists files like .gitignore, Main.java, README.md, and Service.java. The Git Graph tab is selected in the top bar. The main area displays a commit history table and a visual commit graph.

Graph	Description	Date	Author	Commit	
master	Merge branch 'hello-world'	28 Apr 2021 22:45	Andrea Fornia	6984b97c	
hello-world	Aggiornato il Main	28 Apr 2021 22:44	Andrea Fornia	b6da240a	
origin/HEAD	origin/master	Create Service.java	28 Apr 2021 22:42	Andrea Fornia	b12b344e
	Aggiunta classe Main	28 Apr 2021 22:38	Andrea Fornia	392a6cb1	
	Merge branch 'master' of https://github.com/aforanai/tutorial	28 Apr 2021 22:31	Andrea Fornia	eb342dcd	
	README.md aggiornato da VS Code	28 Apr 2021 22:26	Andrea Fornia	fc844041	
	Aggiornato README.md da GitHub	28 Apr 2021 22:25	Andrea Fornia	81048355	
	README aggiornato	28 Apr 2021 22:18	Andrea Fornia	5e48b34a	
	Initial commit	28 Apr 2021 22:10	Andrea Fornia	cf89c72f	

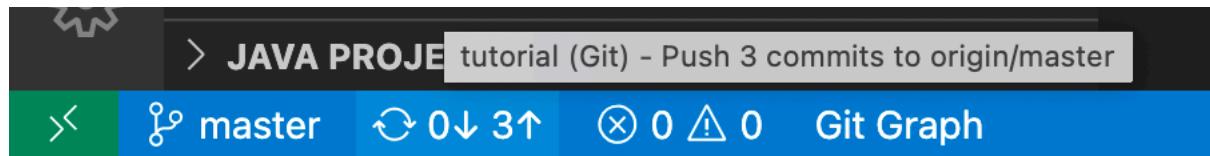
The visual commit graph on the left shows a timeline of commits. A blue line represents the master branch, which merges a pink line (the hello-world branch) at the top. A green line represents the origin/master branch, which merges the master branch. The commits are represented by colored circles: blue for master, pink for hello-world, and green for origin/master.

- Completata la feature, il branch temporaneo non ci serve più e possiamo eliminarlo



# Push dello stato aggiornato del master

- Le modifiche apportate con il merge sul master possono ora essere condivise. In remoto avrò sia la classe Service che la classe Main



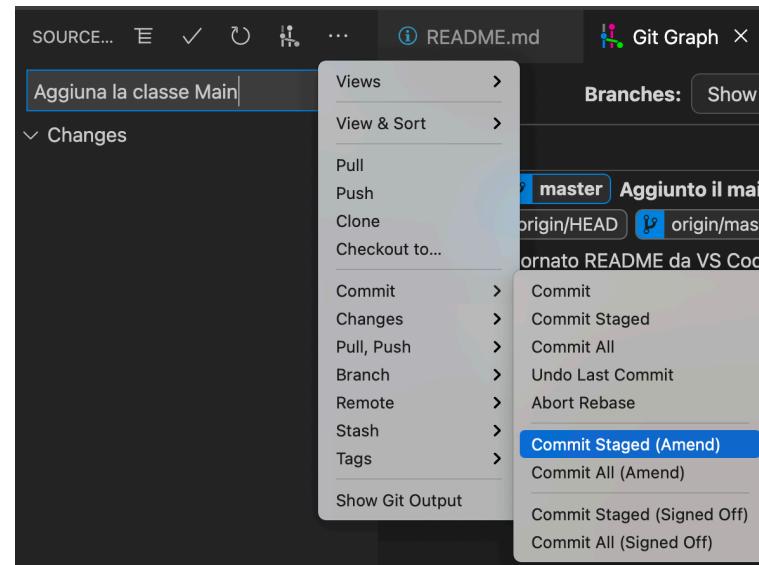
Graph	Description	Date	Author	Commit
	o master origin origin/HEAD Merge branch 'hello-world'	28 Apr 2021 22:45	Andrea Fornaia	6984b97c
	Aggiornato il Main	28 Apr 2021 22:44	Andrea Fornaia	b6da240a
	Create Service.java	28 Apr 2021 22:42	Andrea Fornaia	b12b344e
	Aggiunta classe Main	28 Apr 2021 22:38	Andrea Fornaia	392a6cb1
	Merge branch 'master' of https://github.com/afornaia/tutorial	28 Apr 2021 22:31	Andrea Fornaia	eb342dcd
	README.md aggiornato da VS Code	28 Apr 2021 22:26	Andrea Fornaia	fc844041
	Aggiornato README.md da GitHub	28 Apr 2021 22:25	Andrea Fornaia	81048355
	README aggiornato	28 Apr 2021 22:18	Andrea Fornaia	5e48b34a
	Initial commit	28 Apr 2021 22:10	Andrea Fornaia	cf89c72f

# Amend

- Permette di aggiungere all'ultimo commit delle nuove modifiche. Queste modifiche comprendono sia modifiche al file che modifiche ai metadati (es. il messaggio del commit)
- Nota: il nuovo commit sostituirà quello precedente, ed avrà un id diverso (avendo modificato il contenuto del commit, l'hash sarà diverso)
- Nell'esempio mostrato, abbiamo solo deciso di rifare il commit per cambiare il messaggio. Notare l'id diverso

Prima	Graph	Description	Date	Author	Commit
		Aggiunto il main	29 Apr 2021 17...	Andrea Fornaia	6dd1628d

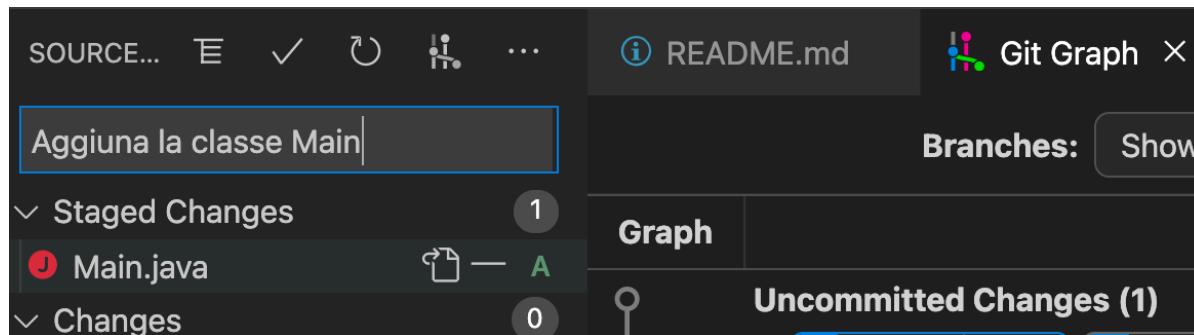
Amend, in cui cambiamo  
solo il messaggio, ma  
potremmo aggiungere file  
alla staging area



Dopo	Graph	Description	Date	Author	Commit
		Aggiunta la classe Main	29 Apr 2021 17...	Andrea Fornaia	d40ee230

# Undo dell'ultimo commit

- È possibile facilmente eliminare l'ultimo commit tramite la funzione di “undo”
- Commit > Undo Last Commit
- Non è un vero comando di git, (equivale a git reset --soft HEAD~1 che vuol dire “sposta HEAD un commit indietro senza cancellare le modifiche fatte”)
- Il commit verrà cancellato e i file modificati torneranno nella staging area
- Questo è particolarmente utile quando abbiamo completato un merge e vogliamo annullarlo



dopo l'undo del commit creato al passo precedente: il commit non c'è più, ma le modifiche sono ancora disponibili, se non ci interessano possono essere tolte dall'area di staging (premendo sull'icona “—”) e poi rimosse (premendo l'icona “discard changes”)