



# Controllo di versione: Git e GitHub

# Cos'è il controllo di versione



- È una parte della gestione della configurazione del software
  - Tenere traccia e controllare le modifiche del software
  - IEEE 828-2012: Standard for configuration management in systems and software engineering
- Situazioni in cui il controllo di versione è utile:
  - Più sviluppatori lavorano contemporaneamente allo stesso file
  - Manutenzione di vecchie release
- Domande a cui il controllo di versione offre risposta:
  - Cos'è cambiato da ieri? E rispetto a una settimana (o mese) fa?
  - «Chi ha cambiato questa linea di codice?»
  - «Quando è stata aggiunta questa funzionalità?»
  - «Perché è stato modificato questo file?»

# Evoluzione



- Unix diff/patch
- Controllo locale di versione
  - SCCS (1972), RCS (1982)
- Controllo centralizzato di versione
  - CVS (1986), SVN (2000)
- Controllo distribuito di versione
  - BitKeeper (2000), Mercurial (2005), **Git** (2005)



# diff / patch

## **diff**

```
$ diff originalfile updatedfile > patchfile.patch
```

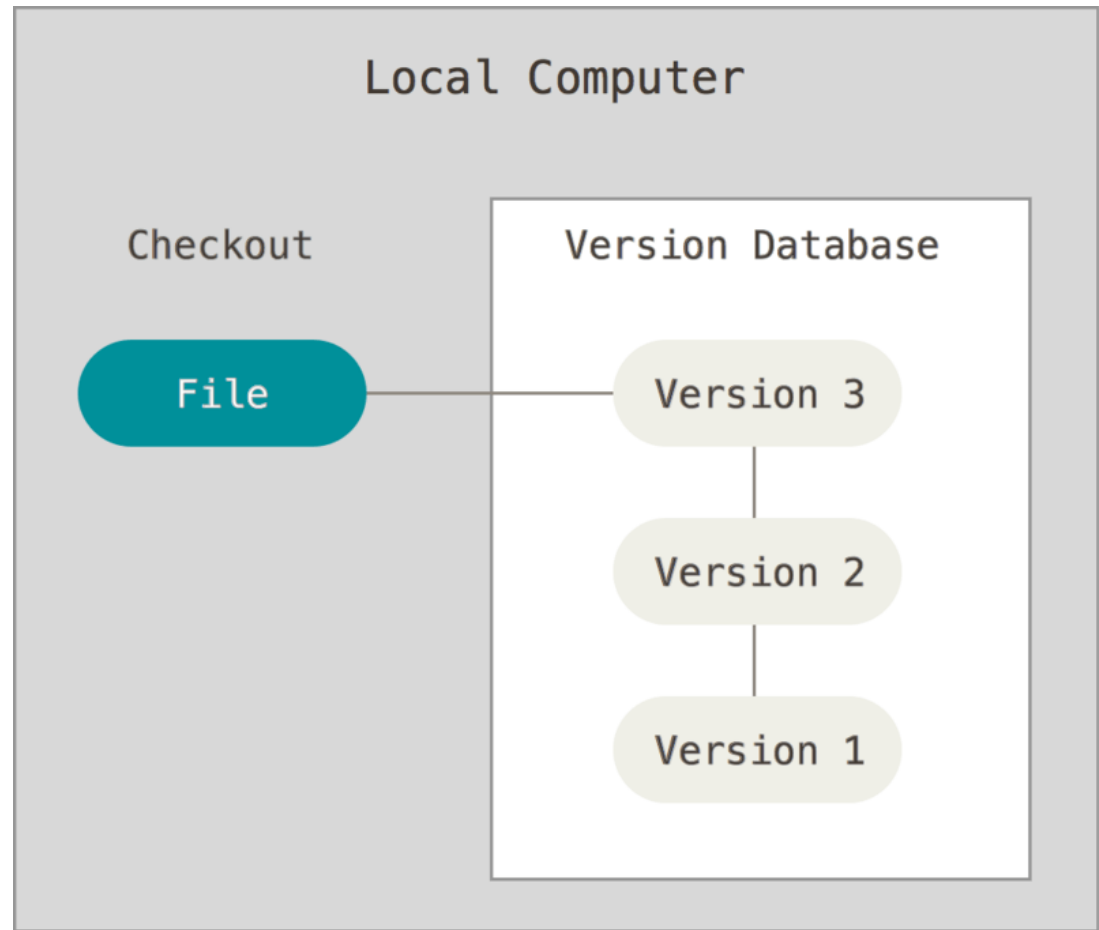
## **patch**

```
$ patch originalfile -i patchfile.patch -o updatedfile
```

# Controllo locale di versione



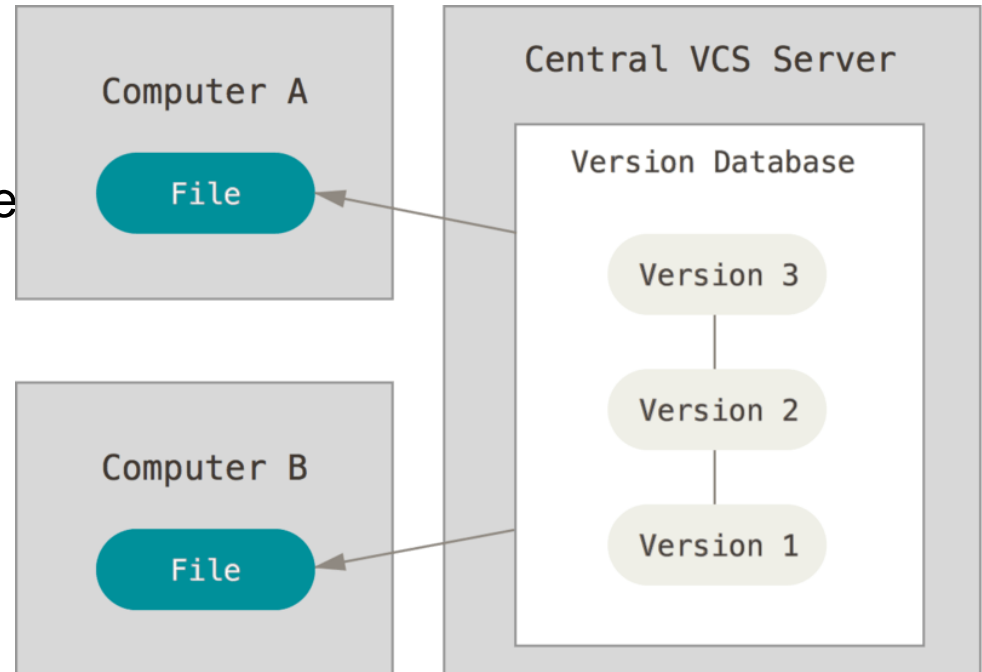
- Lo sviluppatore ha un db locale (**repository**) che tiene traccia di tutte le modifiche
- Esempi: SCCS, RCS
- **Problema:** nessun supporto alla collaborazione



# Controllo centralizzato di versione



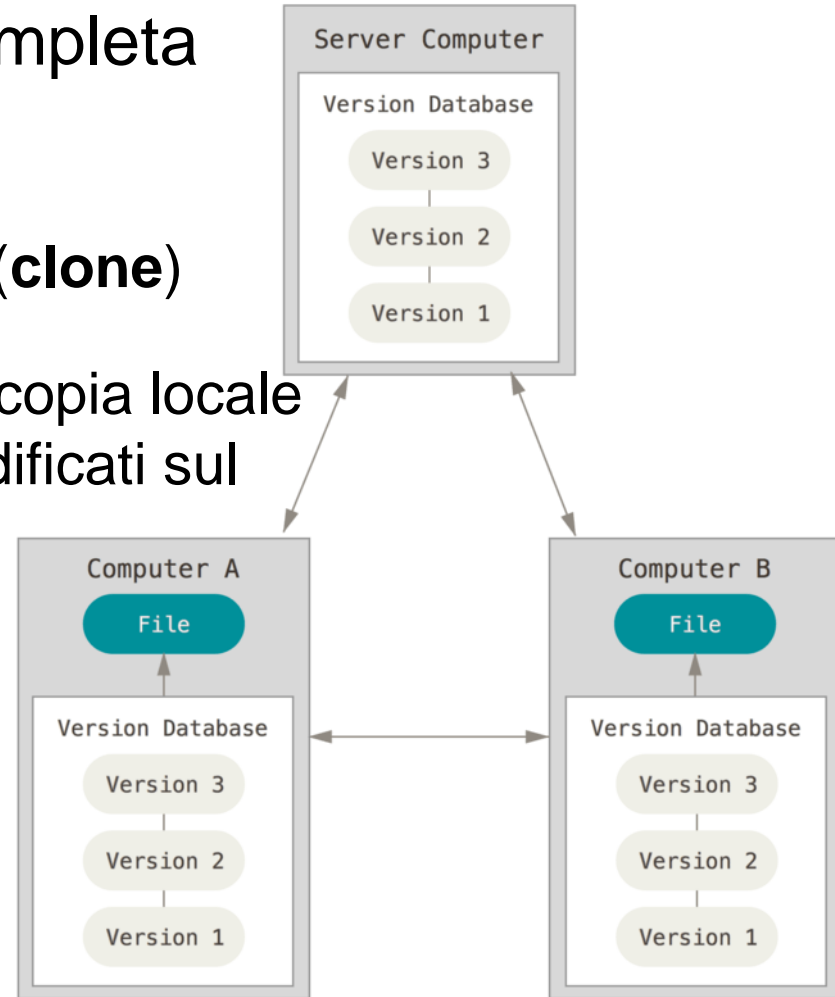
- Un unico repository condiviso su un server contenente tutti i file versionati
- Gli sviluppatori
  - scaricano una copia locale (**checkout**) dal server al computer locale
  - registrano (**commit**) i file modificati sul repository condiviso nel server
- Se c'è un conflitto, chi arriva per ultimo lo risolve (**merge**)
- Esempi: CVS, Subversion (SVN)
- **Problemi:** branching & merging complicati, specialmente per progetti grandi



# Controllo distribuito di versione



- Più repository con storia completa di tutte le versioni
- Gli sviluppatori:
  - Copiano un intero repository (**clone**) o ne creano uno nuovo (**init**)
  - Lavorano offline sui file della copia locale
  - registrano (**commit**) i file modificati sul repository locale
  - Effettuano il merge su un repository remoto (**push**) o chiedono al proprietario di occuparsi lui del merge (**pull**)
- Esempio: **Git**
- **Problema**: curva di apprendimento per via dei molteplici **workflow**



# Client Git



- Command Line (CLI)
  - git
  - gh
- GUI
  - GitHub Desktop
  - ...
- IDE
  - VS Code
  - ...



# Prima volta con git CLI



- Scarica e installa Git <https://git-scm.com>
  - Scarica e installa VS Code
- Apri un terminale
  - Git Bash se Windows

- Configura git

```
$ git config -l
$ git config --global user.name "nome"
$ git config --global user.email "nome@uniba.it"
$ git config --global color.ui true
$ git config --global core.editor "code --wait"
$ git config -l
```

# Autenticazione su GitHub da Git



## con HTTPS

- Usa un personal access token (**PAT**) come password
  - <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>
- Usa Git Credential Manager (**GCM**) per ricordare le credenziali
- Usa GitHub CLI

```
gh auth login
```

## con SSH

- Controlla se esistono chiavi SSH

```
$ ls -al ~/.ssh
```
- Crea una coppia di chiavi SSH

```
$ ssh-keygen -t ed25519 "nome@uniba.it"
> Enter a file in which to save the key
(/c/Users/you/.ssh/id_algorithm): [Press
enter]
> Enter passphrase (empty for no
passphrase): [Type a passphrase]
> Enter same passphrase again: [Type
passphrase again]
```
- Aggiungi la chiave privata SSH al ssh-agent
- Aggiungi la chiave pubblica SSH all'account GitHub

# Git: creare un repository (locale)



## Da zero

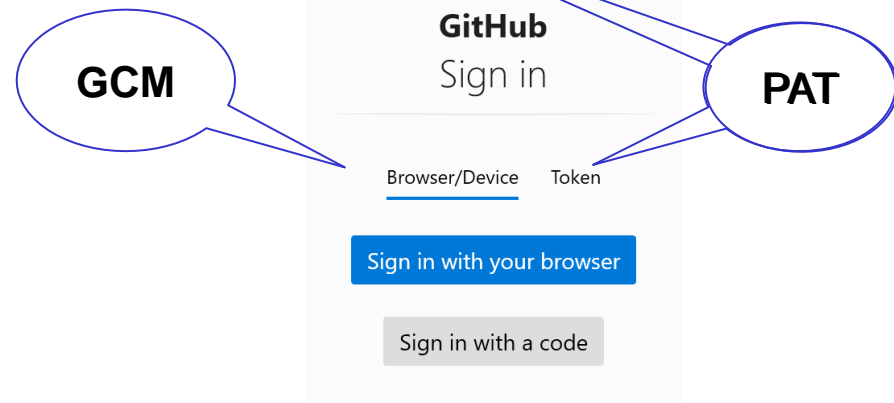
\$ git init <folder\_name>

## Da repository remoto

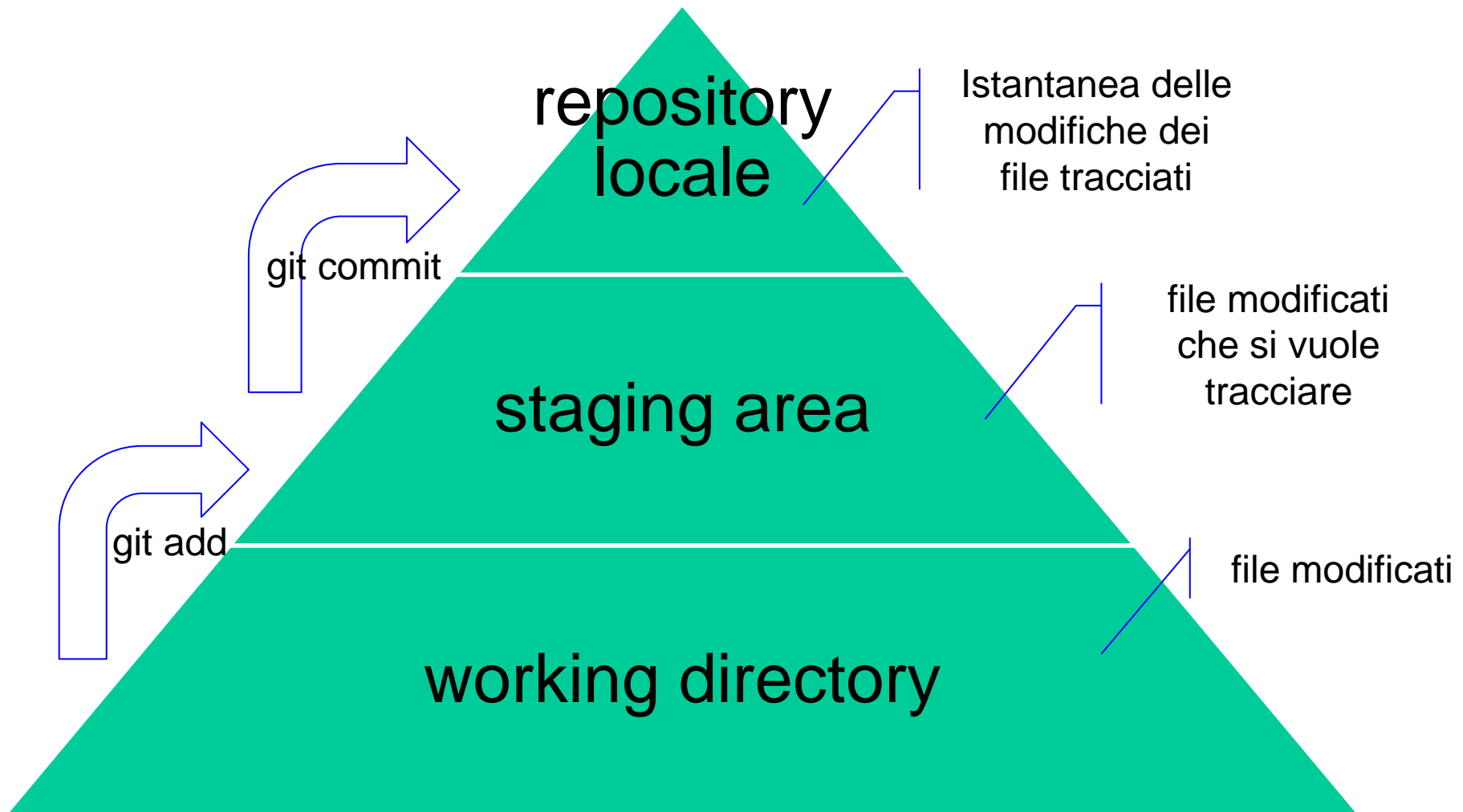
git clone <url>

Se il repository remoto è privato:

```
lanubile@SurfaceFL:~$ git clone https://github.com/softeng-inf-uniba/testing-one.git
Cloning into 'testing-one'...
Username for 'https://github.com': lanubile
Password for 'https://lanubile@github.com':
```



# Stato delle modifiche nel computer locale



# Ricordare i comandi git: cheat sheet



## INSTALL GIT

GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

### GitHub for Windows

<https://windows.github.com>

### GitHub for Mac

<https://mac.github.com>

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

### Git for All Platforms

<http://git-scm.com>

## CONFIGURE TOOLING

Configure user information for all local repositories

```
$ git config --global user.name "[name]"
```

Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```

Sets the email you want attached to your commit transactions

```
$ git config --global color.ui auto
```

Enables helpful colorization of command line output

## CREATE REPOSITORIES

Start a new repository or obtain one from an existing URL

```
$ git init [project-name]
```

Creates a new local repository with the specified name

```
$ git clone [url]
```

Downloads a project and its entire version history

## MAKE CHANGES

Review edits and craft a commit transaction

```
$ git status
```

Lists all new or modified files to be committed

```
$ git diff
```

Shows file differences not yet staged

```
$ git add [file]
```

Snapshots the file in preparation for versioning

```
$ git diff --staged
```

Shows file differences between staging and the last file version

```
$ git reset [file]
```

Unstages the file, but preserve its contents

```
$ git commit -m "[descriptive message]"
```

Records file snapshots permanently in version history

## GROUP CHANGES

Name a series of commits and combine completed efforts

```
$ git branch
```

Lists all local branches in the current repository

```
$ git branch [branch-name]
```

Creates a new branch

```
$ git checkout [branch-name]
```

Switches to the specified branch and updates the working directory

```
$ git merge [branch]
```

Combines the specified branch's history into the current branch

```
$ git branch -d [branch-name]
```

Deletes the specified branch

## REFACTOR FILENAMES

Relocate and remove versioned files

```
$ git rm [file]
```

Deletes the file from the working directory and stages the deletion

```
$ git rm --cached [file]
```

Removes the file from version control but preserves the file locally

```
$ git mv [file-original] [file-renamed]
```

Changes the file name and prepares it for commit

## SUPPRESS TRACKING

Exclude temporary files and paths

```
*.log  
build/  
temp-*
```

A text file named `.gitignore` suppresses accidental versioning of files and paths matching the specified patterns

```
$ git ls-files --other --ignored --exclude-standard
```

Lists all ignored files in this project

## SAVE FRAGMENTS

Shelve and restore incomplete changes

```
$ git stash
```

Temporarily stores all modified tracked files

```
$ git stash pop
```

Restores the most recently stashed files

```
$ git stash list
```

Lists all stashed changesets

```
$ git stash drop
```

Discards the most recently stashed changeset

## REVIEW HISTORY

Browse and inspect the evolution of project files

```
$ git log
```

Lists version history for the current branch

```
$ git log --follow [file]
```

Lists version history for a file, including renames

```
$ git diff [first-branch]...[second-branch]
```

Shows content differences between two branches

```
$ git show [commit]
```

Outputs metadata and content changes of the specified commit

## REDO COMMITS

Erase mistakes and craft replacement history

```
$ git reset [commit]
```

Undoes all commits after `[commit]`, preserving changes locally

```
$ git reset --hard [commit]
```

Discards all history and changes back to the specified commit

## SYNCHRONIZE CHANGES

Register a repository bookmark and exchange version history

```
$ git fetch [bookmark]
```

Downloads all history from the repository bookmark

```
$ git merge [bookmark]/[branch]
```

Combines bookmark's branch into current local branch

```
$ git push [alias] [branch]
```

Uploads all local branch commits to GitHub

```
$ git pull
```

Downloads bookmark history and incorporates changes

<https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>

<https://services.github.com/on-demand/downloads/it/github-git-cheat-sheet/>

# Esercitazione primo commit



# Scrivere commenti significativi in un commit

- Separa il titolo dalla descrizione con una linea vuota
  - Se il commento è breve ci sarà solo il titolo
- Titolo non più lungo di 50 caratteri
  - Se il commento ha una descrizione dettagliata separata dal titolo
- Inizia il titolo con una lettera maiuscola
- Non chiudere il titolo con un punto
- Usa uno stile imperativo
  - Come se stessi dando un comando o dettando un'istruzione

*Ha più senso in inglese*
- Vai a capo nella descrizione dopo 72 caratteri
- **Usa la descrizione per spiegare cosa è stato fatto e perché**, piuttosto che come

|   | COMMENT                            | DATE         |
|---|------------------------------------|--------------|
| ○ | CREATED MAIN LOOP & TIMING CONTROL | 14 HOURS AGO |
| ○ | ENABLED CONFIG FILE PARSING        | 9 HOURS AGO  |
| ○ | MISC BUGFIXES                      | 5 HOURS AGO  |
| ○ | CODE ADDITIONS/EDITS               | 4 HOURS AGO  |
| ○ | MORE CODE                          | 4 HOURS AGO  |
| ○ | HERE HAVE CODE                     | 4 HOURS AGO  |
| ○ | AAAAAAA                            | 3 HOURS AGO  |
| ○ | ADKFJSLKDFJSDKLFJ                  | 3 HOURS AGO  |
| ○ | MY HANDS ARE TYPING WORDS          | 2 HOURS AGO  |
| ○ | HAAAAAAAAANDS                      | 2 HOURS AGO  |

AS A PROJECT DRAGS ON, MY GIT COMMIT  
MESSAGES GET LESS AND LESS INFORMATIVE.

Da <https://chris.beams.io/posts/git-commit>

<https://help.github.com/articles/associating-text-editors-with-git/>

# Comandi base di Git



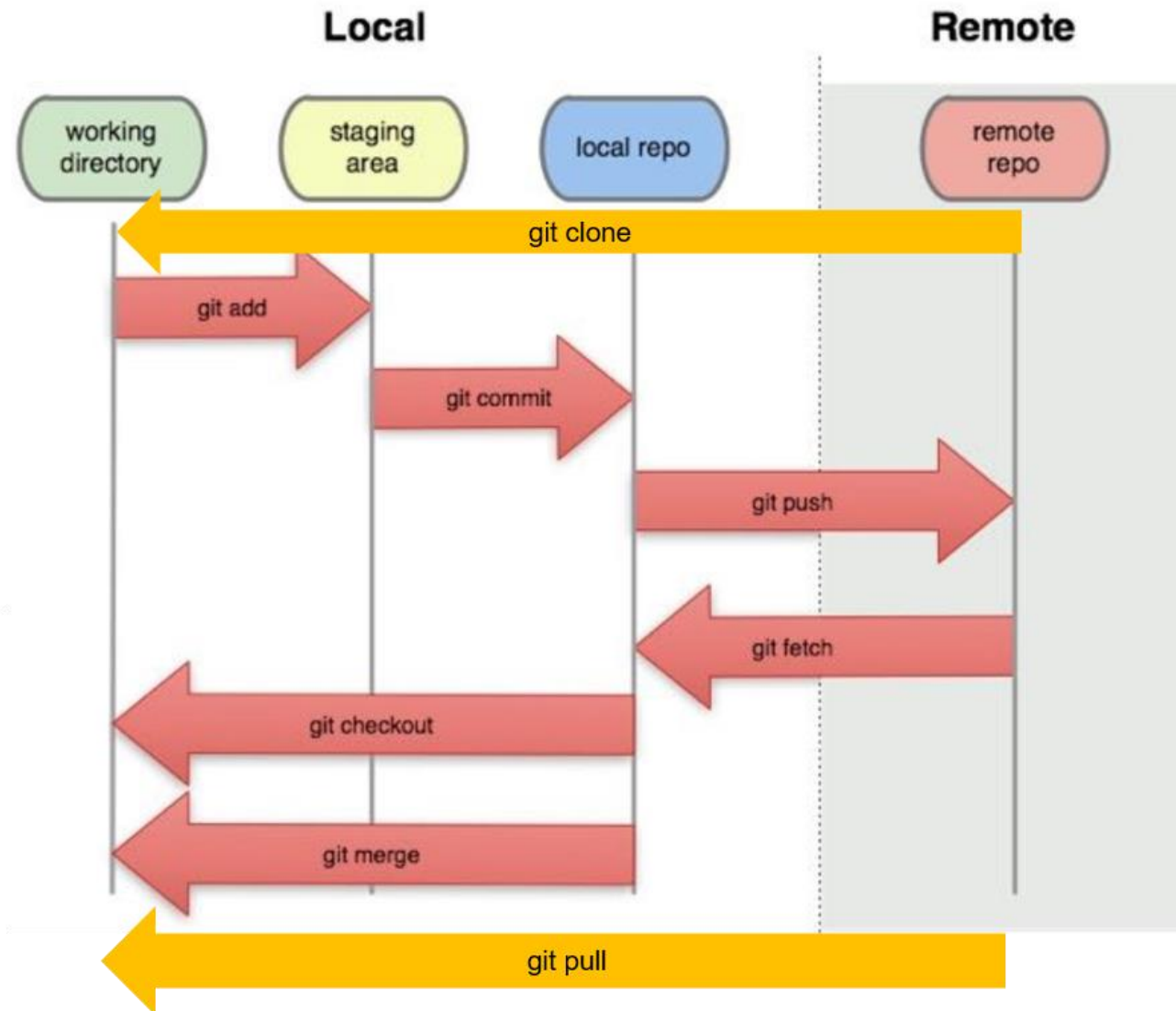
\$ git **clone** (*fetch & initial checkout*): clona un repository completo in una nuova working directory

\$ git **add**: aggiunge le modifiche alla staging area

\$ git **commit**: commit delle modifiche dalla staging area al repository locale

\$ git **push**: carica i commit locali su un repository remoto

\$ git **pull** (*fetch & merge*): scarica e unifica i commit remoti nella propria working directory



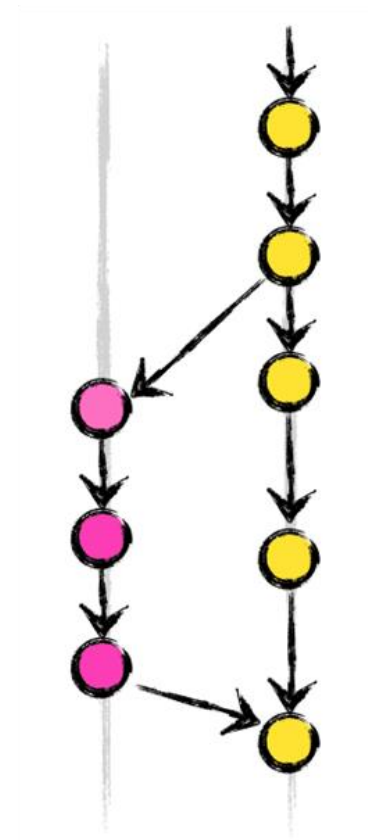


# Branching e merging



- È il modo per lavorare contemporaneamente a più versioni di un repository
- Un *branch* è una linea di sviluppo indipendente
  - Per default, il repository ha un unico branch chiamato «master»
  - Un branch differente è usato per lavorare su modifiche ed estensioni prima di fare il *merge* (fusione) sul master
- Per ogni issue X (feature, bug, ecc):
  - Crea un nuovo branch X dedicato
  - Implementa X
  - Effettua il merge dei cambiamenti di X in master

branch X      master

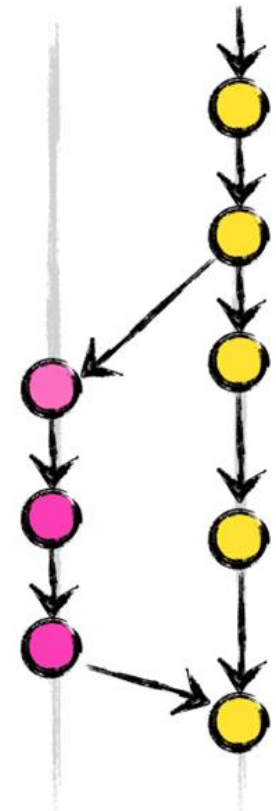


# Comandi git per il branching



- **\$ git branch**
  - Mostra i branch esistenti ed evidenzia quello corrente
- **\$ git branch *branch-name***
  - Crea un nuovo branch a partire da quello corrente
- **\$ git checkout *branch-name***
  - Cambia il branch corrente e aggiorna la working directory
- **\$ git merge *branch-name***
  - Fonde il branch specificato, con tutte le modifiche (commit) effettuate su di esso, nel branch corrente
- **\$ git branch -d *branch-name***
  - Cancella il branch specificato

branch X      master





# Esercitazione branching e merging



# GitHub





## Built for developers

GitHub is a development platform inspired by the way you work. From **open source** to **business**, you can host and review code, manage projects, and build software alongside millions of other developers.

Username

Email

Password

Use at least one letter, one numeral, and seven characters.


Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy policy](#). We'll occasionally send you account related emails.

GitHub for teams




## A better way to work together

GitHub brings teams together to work through problems, move ideas forward, and learn from each other along the way.



Search GitHub

[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Read the guide

Start a project

lanubile

Browse activity

Discover repositories

louieQ created a repository `softeng-inf-uniba/prova-di-assegnazione-di-gruppo-ghost-1` 19 hours ago

`softeng-inf-uniba/prova-di-assegnazione-di-gruppo-ghost-1`  
prova-di-assegnazione-di-gruppo-ghost-1 created by GitHub Classroom  
Updated Mar 13

louieQ created a repository `softeng-inf-uniba/prova-di-assegnazione-di-gruppo-ghost` 19 hours ago

`softeng-inf-uniba/prova-di-assegnazione-di-gruppo-ghost`  
prova-di-assegnazione-di-gruppo-ghost created by GitHub Classroom  
Updated Mar 13

bateman created a repository `softeng-inf-uniba/prova-di-assegnazione-di-gruppo-teamprova2` 19 hours ago

`softeng-inf-uniba/prova-di-assegnazione-di-gruppo-teamprova2`  
prova-di-assegnazione-di-gruppo-teamprova2 created by GitHub Classroom  
Updated Mar 13

danielagir created a repository `softeng-inf-uniba/prova-di-assegnazione-di-gruppo-teamprova1-1` 19 hours ago

`softeng-inf-uniba/prova-di-assegnazione-di-gruppo-teamprova1-1`

Saved replies keyboard shortcuts

Now saved replies have keyboard shortcuts to make them even easier to use.

View new broadcasts

Repositories you contribute to

`softeng-inf-uniba/base1718` 1 ★

`softeng-inf-uniba/ciao mondo` 0 ★

`softeng-inf-uniba/guidadocente1...` 0 ★

`collab-uniba/scrība` 1 ★

Your repositories

New repository

Find a repository...

All Public Private Sources Forks

`aman-srivastava/SNA45Slack`

`Campus-Advisors/module-0-welcome-lan...`

`rotenpotatoes-rails-intro`

`Spoon-Knife`

`hw-ruby-intro`

`hello-world`

`hello-git`

Your teams

Find a team...

`collab-uniba/collabers`

`collab-uniba/scrība-team`

`collab-uniba/team-di-prova`

Prof. Filippo Lanubile

# Repository su GitHub



collab-uniba / scriba

Unwatch 6

Star 1

Fork 2

Code

Issues 37

Pull requests 0

Projects 1

Wiki

Insights

Settings

No description, website, or topics provided.

Edit

Add topics

93 commits

2 branches

0 releases

5 contributors

MIT

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download



lanubile Update README.md

Latest commit c6fe49e 5 minutes ago

Dispatcher

Primo esemplare di server che riceve richieste da un client chiamato ...

2 years ago

Docs

Add files via upload

8 months ago

Scriba

Separato il trascrittore dal client serverSide

2 years ago

client

Merge branch 'master' into issue#98

6 months ago

goProxy

Fixed Dockerfile

7 months ago

server

Update package-lock.json

8 days ago

.gitignore

Initial commit

2 years ago

.travis.yml

Fix issue #75: server automated test added

7 months ago

LICENSE

Initial commit

2 years ago

README.md

Update README.md

5 minutes ago

docker-compose.yml

added proxy on docker-compose

7 months ago

README.md

scriba

- collab-uniba / scriba

Unwatch

6

Star

1

Fork

2

<> Code

Issues 37

Pull requests 0

Projects 1

Wiki

Insights

Settings

Label issues and pull requests for new contributors

Dismiss

Now, GitHub will help potential first-time contributors discover issues labeled with **help wanted** or **good first issue**

Go to Labels

Filters

is:issue is:open

Labels

Milestones

New issue

37 Open

52 Closed

Author

Labels

Projects

Milestones

Assignee

Sort

Ripristino dipendenze Ionic

URGENT

#107 opened on 14 Dec 2017 by PepponeFX

Chiunque può registrarsi

Security

enhancement

#99 opened on 1 Sep 2017 by lanubile

Sperimentare MS Speech Recognition API

Spike

enhancement

#90 opened on 28 Aug 2017 by lanubile

Separare interfaccia Speaker da interfaccia partecipante

enhancement

help wanted

#74 opened on 20 Jul 2017 by PepponeFX

Verificare servizi di traduzione

Spike

#73 opened on 15 Dec 2016 by PepponeFX

3

Provare servizi STT di BlueMix

Spike

#72 opened on 7 Dec 2016 by PepponeFX

5

Aprire il progetto su BlueMix

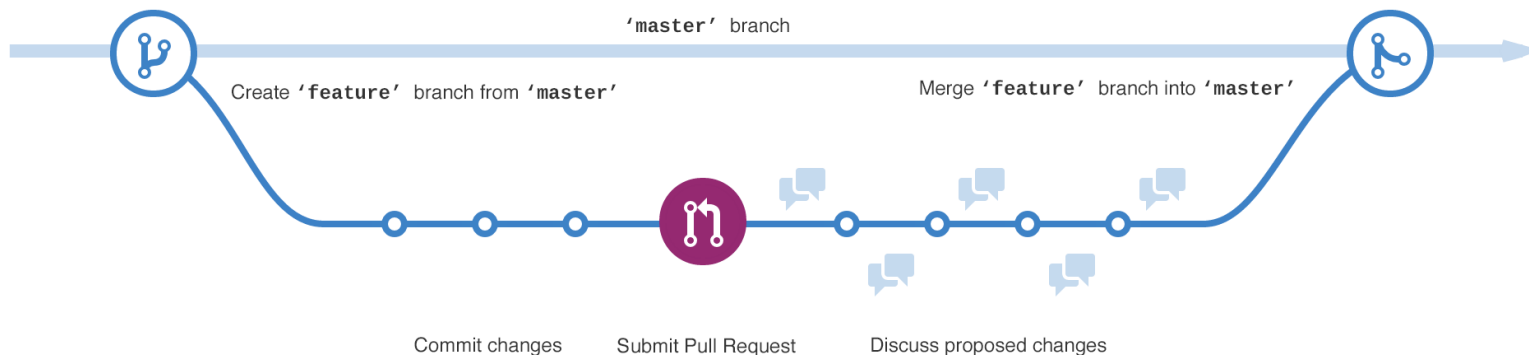
Spike

1

# GitHub Flow: Branching e Pull Request



1. Crea un branch
  - per lavorare su qualcosa di nuovo
2. Aggiungi i commit
  - Lavora sui file, committando sul repository locale e aggiornando il branch sul repository remoto (in GitHub)
3. Apri una pull request
  - dal tuo branch con i cambiamenti proposti
4. Discuti e valuta
  - fai partire una discussione per suggerire cambiamenti
5. Merge e deploy
  - della pull request una volta approvata



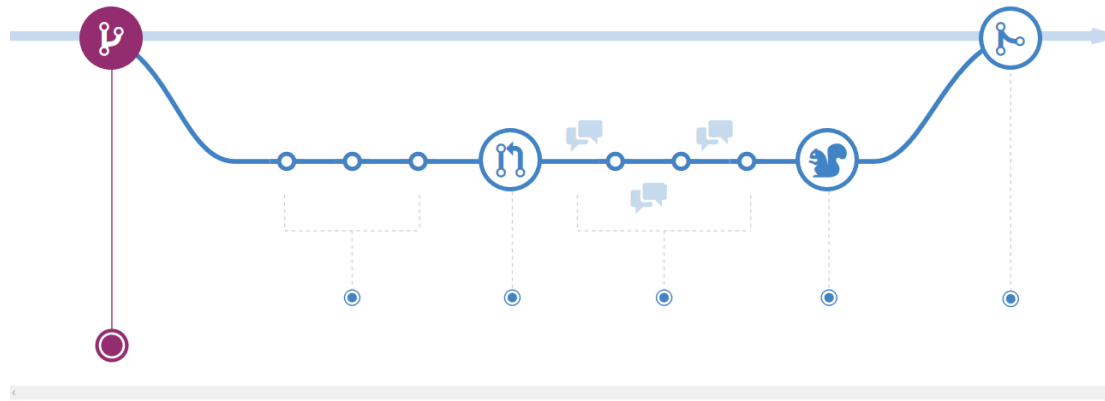
<http://scottchacon.com/2011/08/31/github-flow.html>

Prof. Filippo Lanubile



# 1. Crea un branch

- Crea o scegli l'issue su cui lavorare (feature da aggiungere o bug da risolvere)
  - Segna l'issue come «assigned»
- Crea il branch sul repository locale con il numero dell'issue o il titolo
  - `git branch issue#n` oppure `git branch titoloissue`

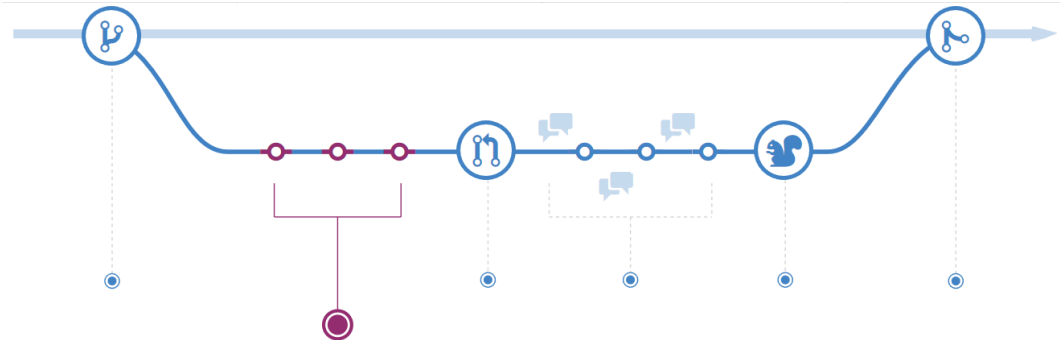






## 2. Aggiungi i commit

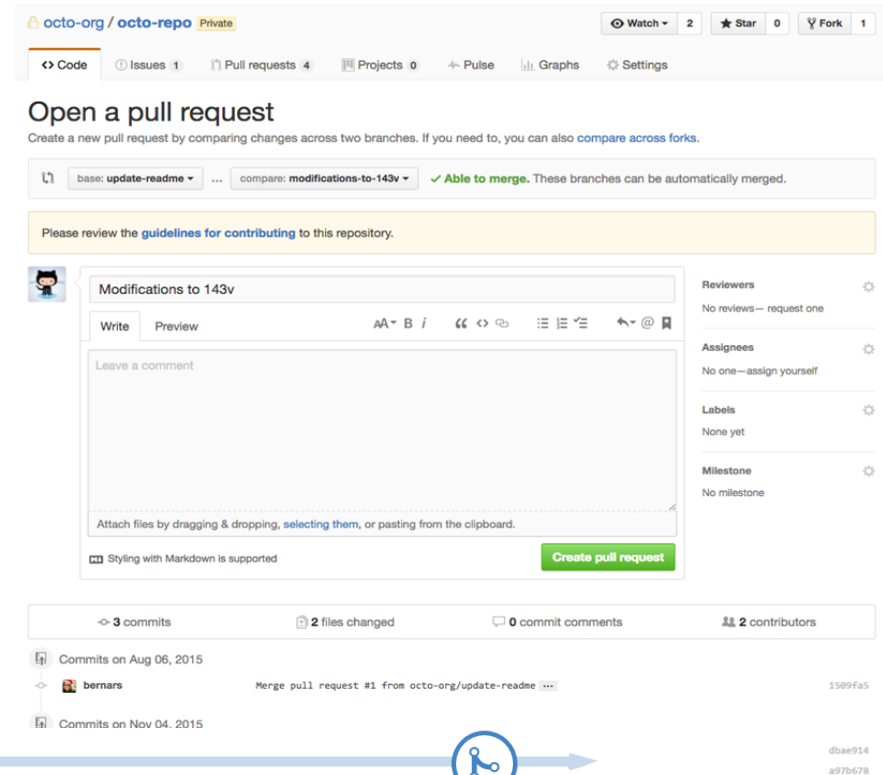
- Spostati sul nuovo branch
  - `$ git checkout issue#n`
- Crea, edita, cancella, i file necessari per implementare il lavoro assegnato
- Esegui i commit delle modifiche sul nuovo branch
  - `$ git add ...`
  - `$ git commit -m "..."`
- Aggiorna con regolarità il branch sul server origin in GitHub (la prima volta viene creato)
  - `$ git push origin issue#n`





# 3. Apri una pull request

- Richiesta di code review sui commit
  - @username
- Riferimento per la chiusura automatica dell'issue:
  - Closes #123



# Scrivere pull request significative

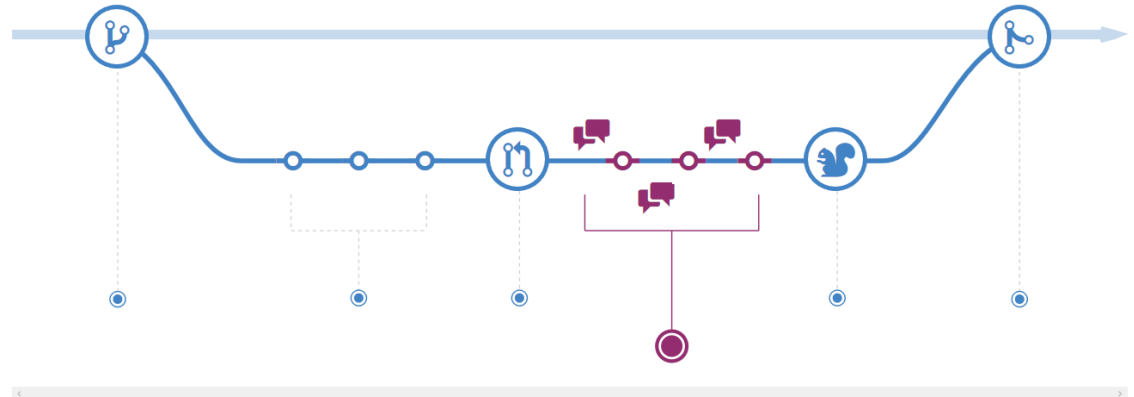
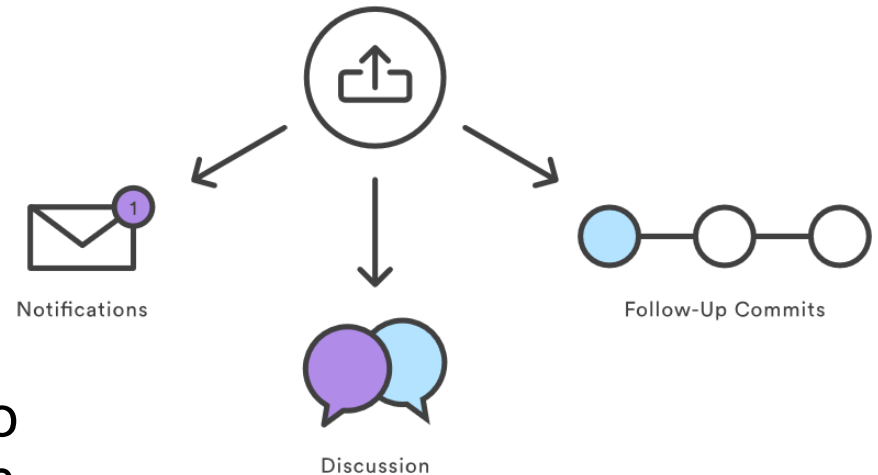


- Titolo: conciso ed esplicativo
- Descrizione:
  - il riferimento all'issue: «Closes #123»
  - a che serve
  - come testare le modifiche
  - note, avvertimenti, link a risorse utili
  - se necessario, inserire anche immagini
  - per menzionare qualcuno: «@username»



## 4. Discuti e valuta

- Chiunque può partecipare alla discussione
  - Molti progetti richiedono l'approvazione di almeno un componente del team
- Possibile aggiungere ulteriori commit prima del merge delle modifiche



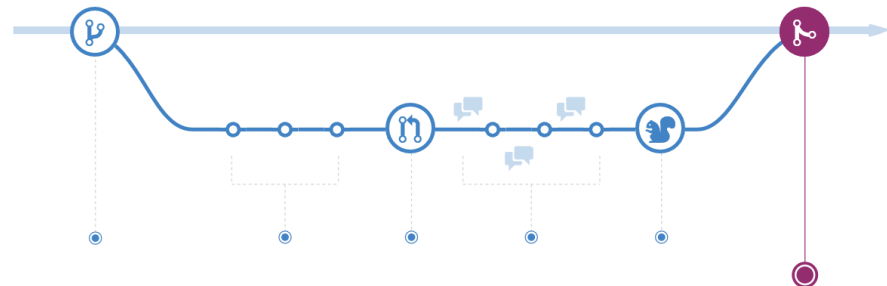
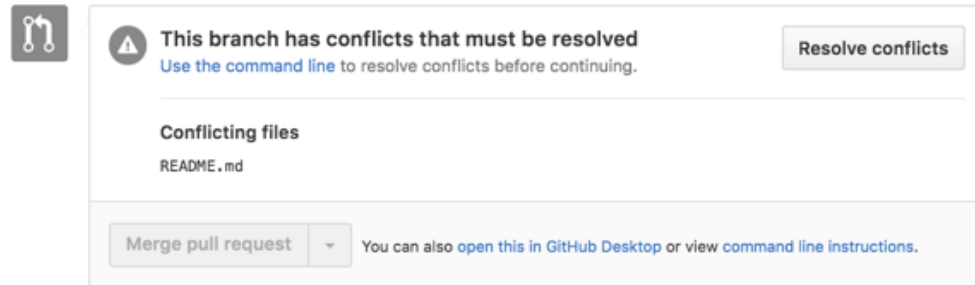
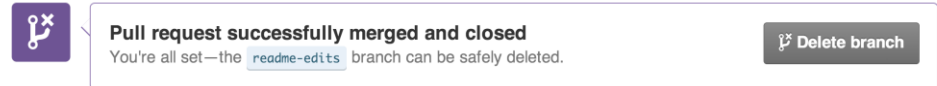
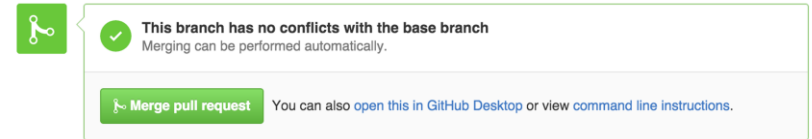


# 5. Merge e deploy

- Se va tutto bene, cancella il branch
  - sul repository remoto: via web
  - sul repository locale:
    - git checkout master
    - git pull
    - git branch -d issue#n
- Se ci sono conflitti con il branch master bisogna risolverli
  - git pull per aggiornare il repository locale
  - Modifica e commit
  - Push dei cambiamenti sul branch del repository remoto
  - Riprova il merge

<https://help.github.com/articles/about-merge-conflicts/>

- Il master branch è pronto per essere «deployato»
  - Manualmente o **automaticamente**



# Esercitazione GitHub Flow



# Best practices



- Aggiorna la working directory all'inizio di una sessione di lavoro
  - git fetch / git pull
- Aggiorna il repository remoto con i cambiamenti locali alla fine di una sessione di lavoro
  - git push
- Testare le modifiche prima di fare un commit
  - A maggior ragione prima di una pull request
- Commit frequenti, piccoli con modifiche logicamente collegate
  - Non mischiare correzioni e nuove feature
- Scrivere commenti significativi in un commit
  - Idem per una pull request
- Non modificare la storia pubblica delle versioni
- Usa i branch
- Rispetta il workflow adottato
- Non usare il controllo di versione per file binari che possono essere dinamicamente costruiti dal codice
  - .gitignore