

Progettazione, Algoritmi e Computabilità

Versioning

Michele Beretta

michele.beretta@unibg.it



Controllo di versione

Controllo di versione

La gestione delle versioni consente di *monitorare* le modifiche fatte a file, soprattutto *sorgenti*. In particolare:

- Che modifiche sono state fatte? Quando? Da chi?
- Che codice c'era alla versione X.Y?

Permette di navigare nella storia e tornare a versioni precedenti.

Nati negli anni 70, i tool si sono evoluti da quelli originali (non più usati).

Tool usati

I più diffusi in ordine di importanza sono:

- **Git** (distribuito)
- Subversion/`svn` (client-server)
- Mercurial (distribuito)

Logica alla base del controllo di versione

Il controllo di versione permette lo **sviluppo collaborativo**:

- Ogni sviluppatore lavora su una *copia locale* del codice
- Terminate le modifiche, la copia locale può essere sincronizzata con una remota
- Necessità di risolvere eventuali conflitti (esempio con Git)

Git

Git



Git è un VCS (Version Control System)

- Gratuito
- Open source
- Distribuito

Creato nel 2005 da Linus Torvalds per lo sviluppo del kernel Linux.

Git – installazione

- *Linux*: se non già installato, usate il vostro package manager (`pacman -S git`, `apt install git`, etc)
- *macOS*: preinstallato con Xcode, ma consigliato comunque di usare Homebrew (`brew install git`) perché più recente
- *Windows*: usare l'installer [qui](#).

Per Windows è necessario successivamente inserire Git nel PATH oppure usare la shell che viene installata.

Git – concetti base

- Un progetto è detto *repository* (o *repo*)
- Un repo è un insieme di file e cartelle gestite da Git
- Un repo è principalmente *in locale* e può essere sincronizzato con dei server git (repo *remote*)
- Ogni progetto ha uno o più *branch*, versioni diverse dello stesso codice
- Esiste sempre almeno un branch, detto *master* o *main*, che rappresenta la versione «principale» dei file
- Quando si vogliono «salvare» delle modifiche si fanno dei *commit* su un branch

Git – concetti base

- Un *commit* è (quasi) **per sempre**: se fate un commit di un file e poi lo eliminate, quel file resta nella storia
- Non sono ammessi commit vuoti o senza commento

Git – perché

- Veloce
- Facilità di merge dei conflitti
- Branching economico
- Semplicità di rollback

Git – edit loop

1. (Opzionale) Sync con un repo remoto (`git pull`)
2. Modifica ai file
3. Aggiunta dei file all'area di staging (`git add`)
4. Commit (`git commit -m "message"`)
5. (Opzionale) Push su un repo remoto (`git push`)

Si possono salvare le modifiche temporaneamente in locale con il comando `git stash`.

Git - comandi da terminale

Comando	Uso
git init	Crea un repo nella cartella locale
git clone <url>	Clona un repo remoto in locale
git checkout <branch>	Cambia il branch locale
git checkout -b <branch>	Crea un branch locale
git pull	Aggiorna il branch locale corrente
git push	Aggiorna il branch remoto corrente
git add <files>	Aggiunge dei file all'area di staging
git commit -m <msg>	Commit dei file nell'area di staging
git stash	Stash locale dei file
git log	Log di tutti i commit
git diff	Mostra le modifiche in pending
git merge <branch>	Merge di un branch nel corrente

Git – merge e risoluzione dei conflitti

Git permette di fare il *merge* fra versioni diverse del codice. Lo fa in automatico quando si sincronizza il repo locale con quello remoto.

Se Git è in grado di risolvere in automatico i conflitti, allora lo fa.

Altrimenti, Git crea delle sezioni nei file affetti con le due possibili soluzioni come nell'esempio

```
<<<<<<<
Contenuto corrente/vecchio del file
=====
Modifiche nuove
>>>>>>>
```

Una volta modificato il file come si desidera, basta salvarlo e fare il commit. È possibile usare `git mergetool` per avere un'interfaccia più carina.

Git – cosa includere in un commit

Sì:

- Codici sorgenti
- File di testo di configurazione e documentazione
- File di input per codice (e.g., immagini/pdf per LaTeX)

No:

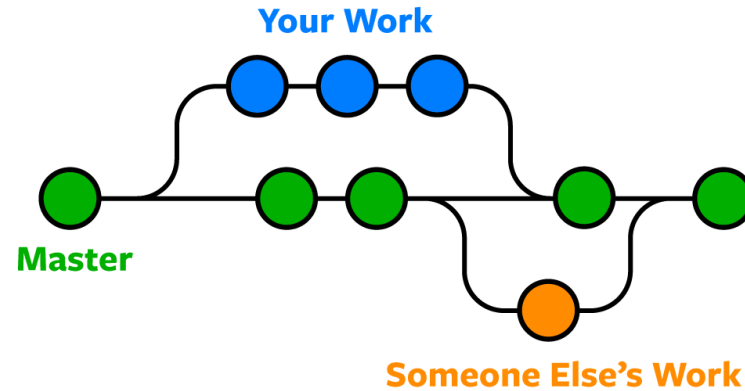
- File binari (.docx, .xlsx, .rtf, .pdf, etc.)
- Compilati (.class, .o, etc.)

ASSOLUTAMENTE NO:

- Password
- Chiavi SSH

Si possono ignorare file tramite `.gitignore`.

Git - branch



Tramite i branch è possibile lavorare su «versioni diverse» dello stesso progetto, in modo da non intralciarsi a vicenda.

I branch sono usati tipicamente per lavorare su fix e features.

GitHub

GitHub



GitHub è una piattaforma **cloud** che permette lo sviluppo di software tramite Git.

Esistono varie alternative, come GitLab, ma GitHub è tra le più usate.

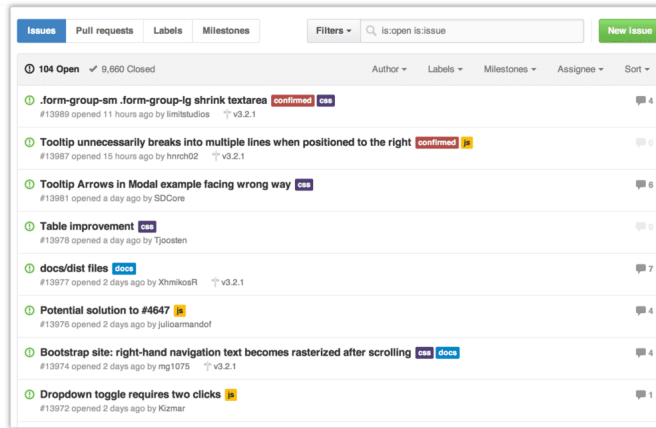
GitHub – cosa offre in più

GitHub offre alcune funzionalità in più rispetto a Git:

- Issues
- Pull Requests
- Projects
- Wiki
- Azioni

E tanto altro

Issues

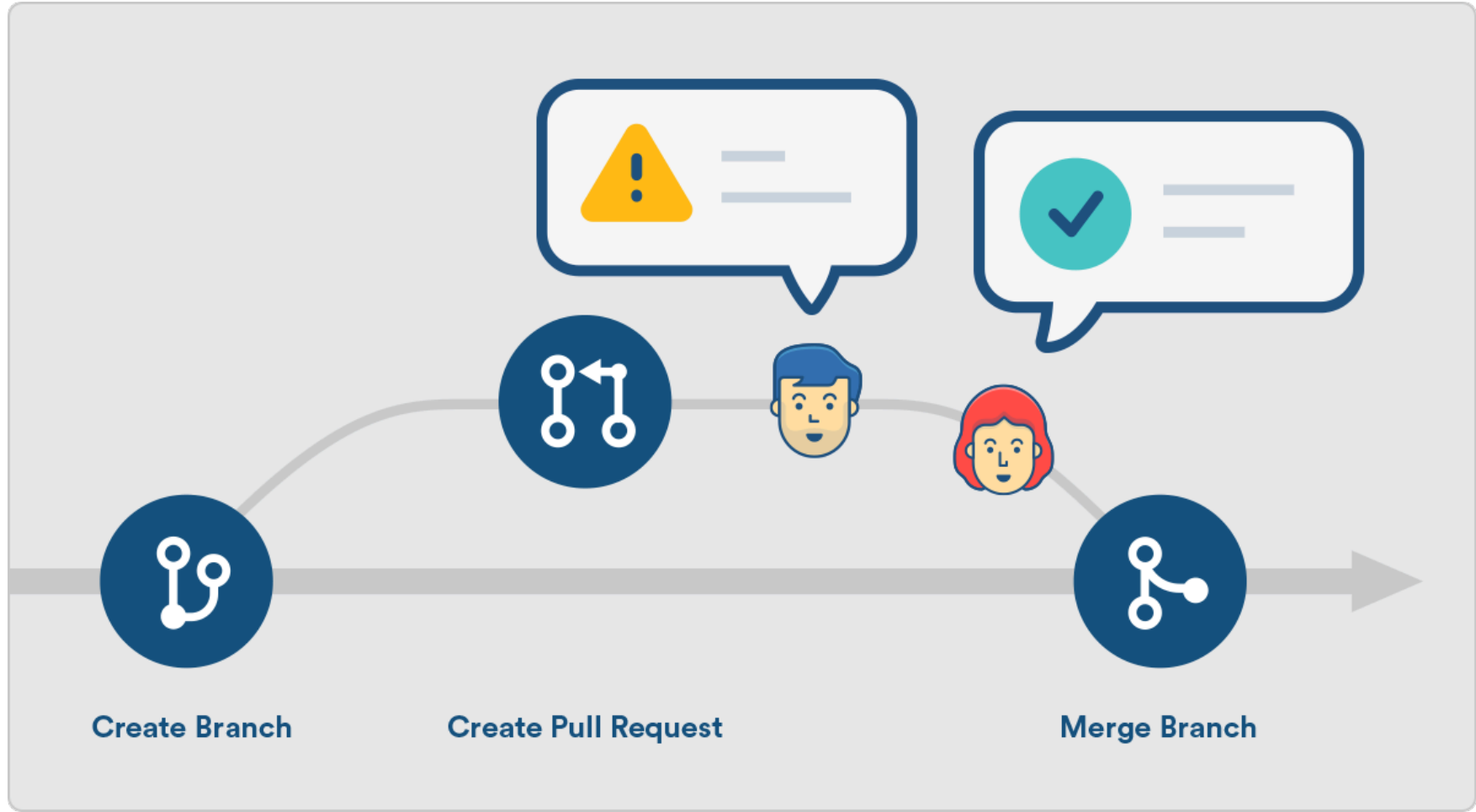


Una *issue* è una discussione che tiene traccia di un problema nel codice. Possono essere aperte da chiunque, e una volta risolto il problema vengono chiuse.

Pull Requests

Tramite le pull-requests viene fatto il merge di un codice di un branch su di un altro branch. È possibile farlo anche da delle fork di un progetto.

Per ogni pull request è possibile definire uno o più revisori che si devono occupare di revisionare la pull request ed approvarla (o eventualmente richiedere modifiche necessarie affinché sia approvabile).



Prova pratica

Esercizio 1. Registrarsi a GitHub¹ se non già fatto. Creare un repo privato, clonarlo, ed eseguire il push di un file di testo a scelta.

Esercizio 2 (Riprende esercizio 1). Modificare il file di testo a piacere e caricare le modifiche.

Esercizio 3. Creare un repo locale a partire da un progetto Java a scelta, e caricare il codice già esistente su un repo remoto (usando `.gitignore`). Successivamente, creare dei commenti in JavaDoc, generare la documentazione in automatico tramite JAutoDoc e caricare le modifiche.

¹0 altri servizi