

Università degli studi di Perugia



Candidato:
Domenico Pepino

Relatore:
Luca Grilli

Dipartimento di Ingegneria
Corso di Laurea Triennale in Ingegneria Informatica ed Elettronica

IL SISTEMA DI CONTROLLO DI VERSIONE DISTRIBUITO GIT

Sommario

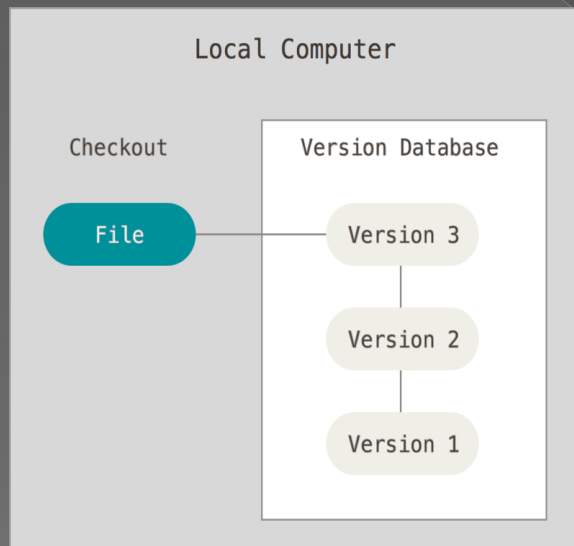
- Sistemi di controllo di versione
- Git
- GitHub e alternative

Obiettivo

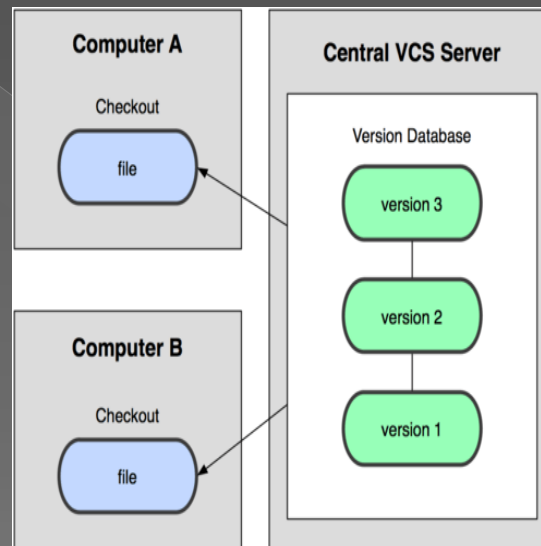
Mostrare come i Sistemi di controllo di versione, Git soprattutto, permettono di tenere traccia di tutte le modifiche di un progetto e di evitare l'insorgere dei conflitti delle modifiche di diversi sviluppatori sullo stesso codice.

Sistemi di Controllo di Versione

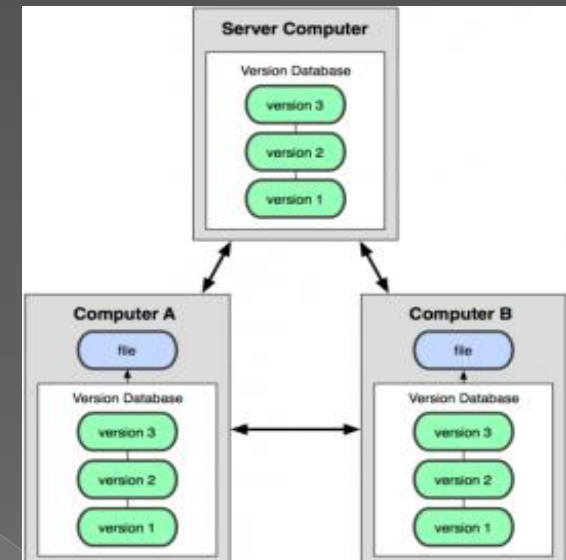
Locale



Centralizzato



Distribuito



- Database semplice
- Informazioni Locali

- Permette la collaborazione tra sviluppatori
- Informazioni sul server
- Miglior protocolli: CVS, SUBVERSION

- I client controllano una copia completa della repository
- Evita la problematica della rottura del server

Il Sistema di Controllo di Versione Distribuito GIT

Nasce nel 2005, dopo la revoca dell'uso gratuito di BitKeeper, con i seguenti obiettivi prestazionali:

- Velocità
- Design Semplice
- Ottimo supporto allo sviluppo non-lineare
- Gestione del controllo di versione totalmente distribuito
- Capacità di gestire grandi progetti

È possibile installarlo su qualsiasi sistema operativo, tramite pannello di controllo o interfaccia grafica, ad esempio per i sistemi linux:

```
$ yum install curl-devel expat-devel gettext-devel \ openssl-devel zlib-devel
```

Gestione del Controllo

Ogni volta che si opera una modifica in un file o un salvataggio dello stato, Git fa un'immagine di tutti i file modificati in quel momento, creando una serie di istantanee.

Controlla tramite checksum ogni operazione.

Utilizza il checksum denominato SHA-1.

SHA-1 è una funzione di hash crittografica che accetta un input e produce un valore di hash a 20 byte, denominato come digest del messaggio.

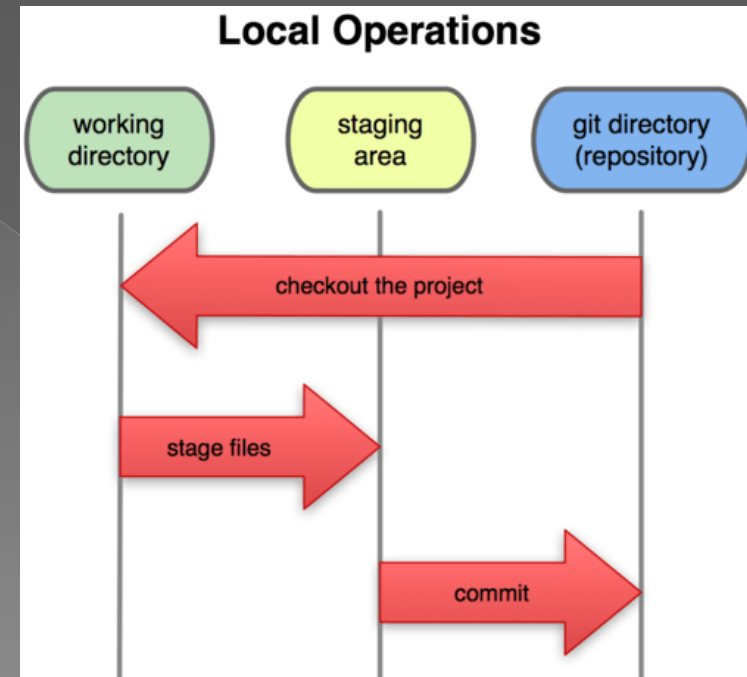
Divisione per stati dei file e del piano di lavoro

All'interno di Git i file possono entrare in tre stati

- Committed
- Modified
- In Staged

Questi tre strati dividono il piano di lavoro in:

- Directory di Git
- Working directory
- Area di stage



Comandi per la creazione e gestione di un progetto

- `git clone`, per importare una repository esistente
- `git init`, per creare una repository
- `git add`, per poter aggiungere un file in area di stage o nella directory di git
- `git status`, per conoscere lo stato dei file (tracciato o non tracciato e unmodified o modified)
- `git diff`, per conoscere le modifiche di un file presente in due aree diverse del progetto
- `git commit`, per ufficializzare un file nel progetto
- `git checkout --` , per eliminare tutte le modifiche
- `git rm`, per rimuovere un file.
- `git mv nomefile nomenuovofile`, per rinominare un file (Git non traccia questa operazione)
- `git log`, per controllare la cronologia delle commit

Etichette

Permettono di contrassegnare i file e mantenere una migliore gestione della cronologia.

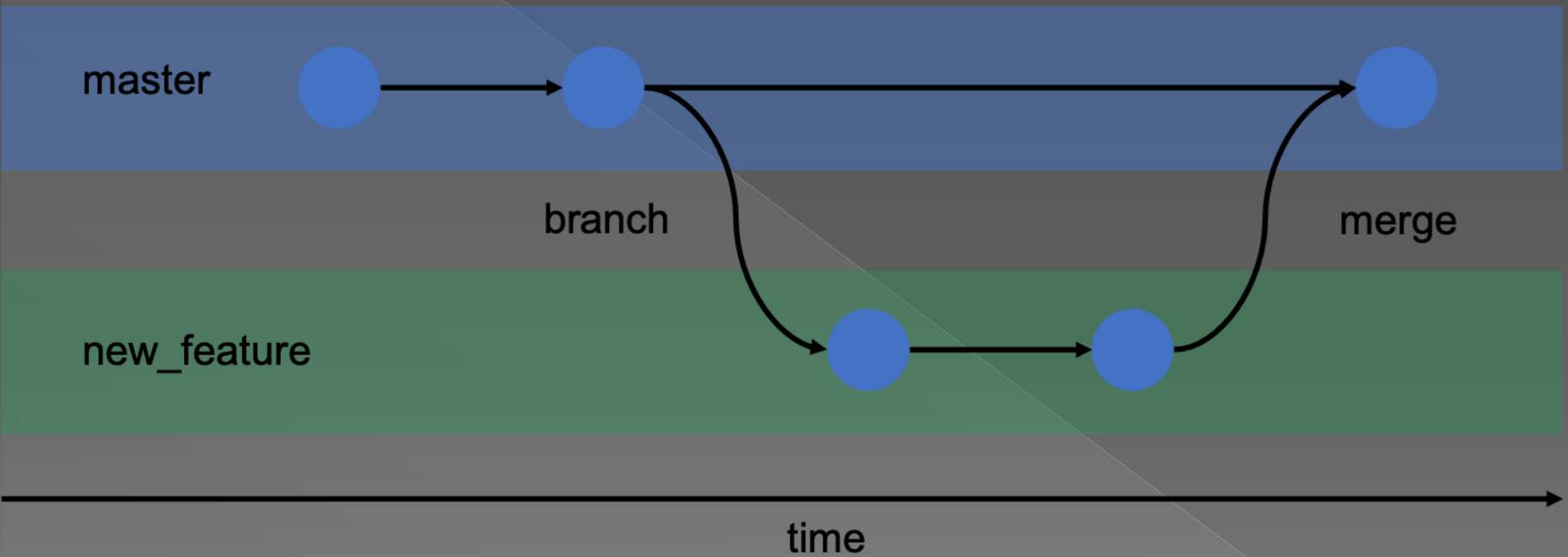
Esistono due tipi di etichette:

- Semplici, rappresentano il riferimento ad un commit specifico
- Annotate, oggetti complessi, ne viene calcolato il checksum, possono essere firmate con GPG

Comandi utili:

- `git tag`, per creare un'etichetta (`-a` per creare le etichette Annotate)
- `git show` per ottenere tutte le informazioni aggiuntive (nome, email e data del creatore)
- `git tag`, per creare un elenco di tutte le etichette create

Branching



Branching : Comandi

- git branch, permette la creazione dei rami o di elencare tutti i rami creati
- git checkout, seleziona il ramo indicato
- git merge, per unire un ramo al progetto originale
- git branch – merged, per visualizzare tutti i rami uniti
- git branch –no-merged, per visualizzare solo i rami non uniti
- git rebase, unisce un ramo al progetto originale

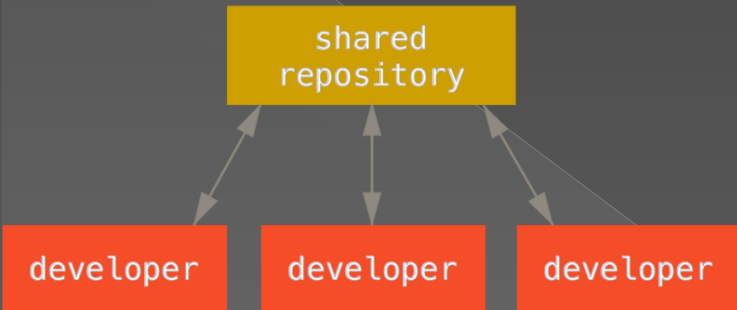
Differenza tra rebasing e merging

Effettuando l'operazione di rebasing, il ramo viene aggiunto alla fine del progetto originale.

Non si ha sovrapposizione di rami

Cronologia più comprensibile

Workflows

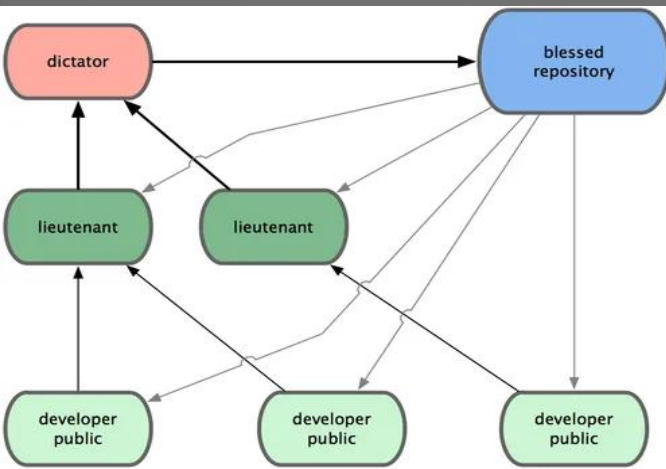
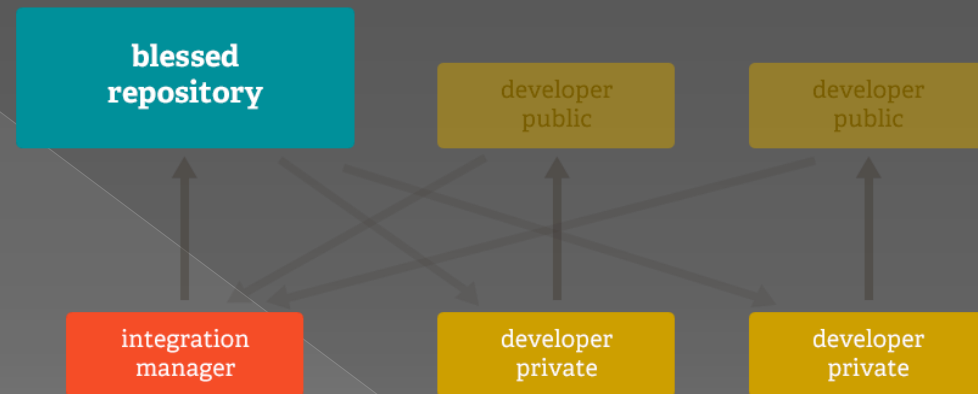


Workflows centralizzato

- Utilizzato per progetti piccoli con pochi sviluppatori
- Tutti gli sviluppatori possono accedere e modificare il progetto originale

Workflows con un manager

- Il manager è l'unico che può effettuare modifiche sul progetto universale
- gli sviluppatori inviano il proprio lavoro al manager



Workflows con un dittatore e tenenti:

- Utilizzata su grandi progetti
- I tenenti sono responsabili solo di alcuni sottoteam
- il dittatore è l'unico che può effettuare modifiche sul progetto originale

Protocolli per la condivisione di un progetto

Un aspetto importante per la condivisione di un progetto è il tipo di protocollo scelto per comunicare con la repository remota.

In Git esistono quattro principale protocolli:

- Locale
- Secure Shell
- Git
- HTTP

GitHub

- Offre la possibilità di utilizzare sia Git che Subversion
- Consente di creare in modo illimitato le repository pubbliche, con pagamento mensile per le private
- Permette di creare team che funzionano come account normali
- Permette di configurare velocemente la propria repository con i protocolli HTTPS e SSH
- Permette l'accesso in sola lettura tramite il link di condivisione

Servizi di hosting alternativi a GitHub

	GitHub	Bitbucket	SourceForge	GitLab
Costo	Gratuito	Gratuito	Gratuito	Gratuito
Repository private	A pagamento	Illimitate e gratuite	Illimitate e gratuite	Illimitate e gratuite
Repository pubbliche	Illimitate e gratuite	Illimitate e gratuite	Illimitate e gratuite	Illimitate e gratuite
Limite di archiviazione	1 GB per repository	2 GB per repository	Nessuno	Nessuno
Utenti	Illimitati	5 per gli account gratuiti	Non permette la collaborazione	Illimitati
Tipi di sistemi di controllo di versione	Git SVN	Git, HG	Git , SVN, HG	Nessuno
Wiki	Disponibile	Disponibile	Disponibile	Disponibile

Conclusioni

Abbiamo visto come i sistemi di controllo di versione aiutino a soddisfare tutte le esigenze degli sviluppatori, permettendo la gestione della cronologia dei lavori attraverso un'opportuna gestione di versioning. Git mette a disposizione un efficiente controllo di versione e permette, attraverso il branching, di agevolare notevolmente il lavoro dei programmatori permettendo loro di duplicare il progetto principale e di modificarlo senza sporcare il codice. Si è visto inoltre che, utilizzando i sistemi di controllo di versione, un aspetto molto importante è la gestione del team di sviluppo e l'utilizzo dei protocolli per la lettura e scrittura, utilizzati per ovviare i problemi dei conflitti e per la sicurezza del progetto stesso.

Grazie per l'attenzione