Utilizzo base di Gerrit

Installare hook commit-msg per generare Change-ID e remote

```
gitdir=$(git rev-parse --git-dir)
curl -Lo ${gitdir}/hooks/commit-msg \
  https://gerrit-review.googlesource.com/tools/hooks/commit-msg
chmod +x ${gitdir}/hooks/commit-msg

git remote add gerrit ssh://username@gerrit-host:29418/project-name.git
```

Inviare modifiche a Gerrit:

```
git add .
git commit -m "Feature: implementazione X"
git push gerrit HEAD:refs/for/master
```

Aggiornare una modifica esistente:

```
git add .
git commit --amend
git push gerrit HEAD:refs/for/master
```

NoteDb: Il sistema di storage

NoteDB è il sistema di archiviazione dati introdotto in Gerrit 2.15 (2017) che ha sostituito il precedente database relazionale.

- Archiviazione dei metadati direttamente in Git
  - **Changes**: dati sulle modifiche proposte
  - **Comments**: commenti sulle revisioni
  - **Accounts**: informazioni sugli utenti
  - **Groups**: appartenenze ai gruppi
  - **Projects**: configurazione progetti

```
refs/changes/XX/YYYY/meta
refs/changes/XX/YYYY/drafts
refs/changes/XX/YYYY/robot-comments
refs/users/AB/user-UUID/
```

- Sostituto del database SQL (pre-3.x)
- Vantaggi:
  - Storia completa di review e commenti
  - Backup e replica più semplici
  - Transazioni atomiche con il codice
  - Replicazione efficiente

Vantaggi di NoteDB

Coerenza dei dati

- Repository Git e metadati restano **sempre allineati**
- Eliminazione dei problemi di sincronizzazione tra DB e repository
- Transazioni atomiche Git per modifiche

Backup semplificato

- Backup completo tramite normale clonazione Git
- Nessun sistema di backup separato per DB
- Disaster recovery più semplice

Migliore audit trail

- Storia completa e immutabile di tutte le modifiche
- Tracciabilità totale delle operazioni
- Verifica facilitata di chi ha fatto cosa e quando

Maggiore scalabilità

- Migliore supporto per installazioni distribuite
- Elimina il DB come potenziale collo di bottiglia
- Facilita lo scaling orizzontale

Svantaggi di NoteDB

Prestazioni in alcuni scenari

- Query complesse possono essere più lente
- L'accesso random ai dati richiede più risorse
- La ricerca full-text è più complessa

Transizione complessa

- La migrazione da DB relazionale può essere difficile
- Alcune query devono essere riprogettate
- Curva di apprendimento per gli amministratori

Complessità nel debugging

- Struttura dei dati meno intuitiva rispetto alle tabelle SQL
- Richiede comprensione di Git refs e oggetti
- Strumenti di debugging meno maturi

Storage inefficiente

- Archiviazione di metadati in Git può richiedere più spazio
- Complessità nella garbage collection
- Maggior consumo di inodes nel filesystem

Il namespace refs/for

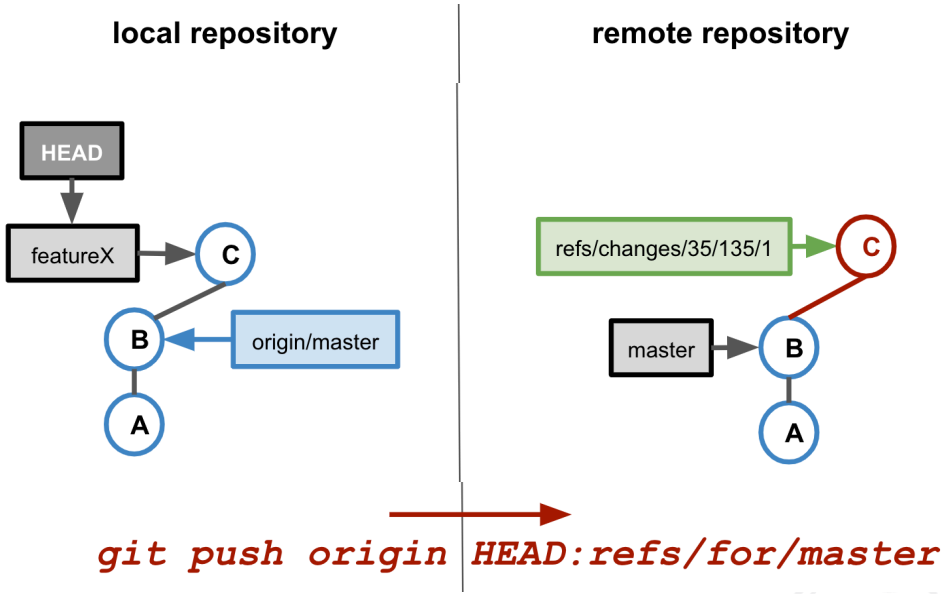
- Namespace speciale per il processo di review
- Formato: `refs/for/<branch-name>[/<topic>]`
- Non è un riferimento Git reale ma un endpoint
- Il server Gerrit:
  1. Riceve il push
  2. Valida il contenuto
  3. Crea o aggiorna un Change
  4. Genera riferimenti interni

```
# Esempio di push per review
git push origin HEAD:refs/for/master

# Con topic
git push origin HEAD:refs/for/master/feature-x

# Per draft
git push origin HEAD:refs/drafts/master
```

Push for code review



```
wget https://gerrit-releases.storage.googleapis.com/gerrit-3.11.2.war
java -jar gerrit-3.11.2.war init -d /tmp/gerrit
# accetta default tranne
# [auth]       type = DEVELOPMENT_BECOME_ANY_ACCOUNT
# browser
# usa admin
# in preferenze utente creare HTTP password
# crea progetto prova
git clone http://localhost:8080/prova
cd prova
gitdir=$(git rev-parse --git-dir)
curl -Lo ${gitdir}/hooks/commit-msg https://gerrit-review.googlesource.com/tools/hooks/commit-msg
chmod +x ${gitdir}/hooks/commit-msg
cd prova
touch A
git add A
git commit
git show   // vedi changeID
#crea change request
git push origin HEAD:refs/for/master
git ls-remote  origin
git fetch origin refs/changes/01/1/meta
git show FETCH_HEAD
git allcommit | xargs git hist
#aggiungi commento votazione chiusura
git fetch origin refs/changes/01/1/meta
git show FETCH_HEAD
git allcommit | xargs git hist
```

Critiche a semantic versioning

https://tom.preston-werner.com/2022/05/23/major-version-numbers-are-not-sacred

https://antfu.me/posts/epoch-semver

UNIVERSITÀ DEGLI STUDI
DI MILANO

Dependency Hell … ovunque

Dipendenze: necessarie?

- pericolo: sindrome NIH
- ma attenti anche a non esagerare:
  - risposta stackoverflow che vi risolve un semplice problema facendovi importare un intero framework
- evidenziarle e chiarirle
  - al livello giusto

… nelle specifiche …

- Possono essere:
  - legacy dependency
  - necessarie per potersi integrare a sistemi esistenti
  - richieste dal cliente

… nella progettazione …

- ad esempio in UML
  - class diagram
  - component diagram
  - deployment diagram

https://www.oracle.com/it/corporate/features/understanding-java-9-modules.html

… nella implementazione …

- linguaggi (ad esempio Java)
  - `import` delle classi
  - `module-info.java`
    - requires
    - exports
- build automation (ad esempio in Gradle)

… nel testing …

- dipendenza da librerie specifiche
  - junit, mockito
- dipendenza da risorse varie (esempi, files)

… durante il deploy

- pacchettizzazioni (apt-get, pip, homebrew)
- dinuovo build automation (plugin e task appositi)
- docker

http://git-scm.com/book/it/v2/Git-Tools-Submodules

… e nel versioning …

dipendiamo da una libreria esterna:

- ma se vogliamo/dobbiamo fare una patch?
- se la stiamo sviluppando in parallelo?
- o semplicemente non è pacchettizzata

Git fornisce un suo meccanismo per gestire questi casi:

- **submodules**

Git submodules

- Utile nel caso si debba lavorare sul codice della dipendenza insieme al progetto
- la dipendenza è un repository git a sé stante, in una sottodirectory
  - lo aggiungo con `git submodule add repo dirname`
    - Clona il sottomodulo nella directory corrente e, per impostazione predefinita, esegue il checkout del branch main.
    - Aggiunge il percorso di clonazione del sottomodulo al file ".gitmodules" e aggiunge questo file all'indice, pronto per essere committato.
    - Aggiunge l'ID del commit corrente del sottomodulo all'indice, pronto per essere committato.
  - lo "scarico" con `git submodule update`
    - Clona il repository del sottomodulo e fa il checkout del commit specificato nel superprogetto.
    - Se il sottomodulo è già stato clonato, aggiorna il sottomodulo al commit specificato nel superprogetto.
  - lo aggiorno con `git submodule update --remote`
    - Aggiorna il sottomodulo al commit più recente del branch specificato nel file ".gitmodules".

https://git.wiki.kernel.org/index.php/GitSubmoduleTutorial

Piccola demo submodules

Fortemente ispirata (anche se leggermente semplificata) a quella del link riportato e a cui vi rimando (oltre a chiaramente il video)