Git Cheat-Sheet: Prontuario operativo e buone pratiche di versioning

Git Cheat-Sheet: Prontuario Operativo

Git è uno standard de facto per il controllo di versione distribuito, ma la ricchezza delle sue opzioni a riga di comando può risultare complessa in scenari critici o durante la risoluzione di conflitti complessi.

Questa guida raccoglie i pattern operativi, i comandi essenziali e le procedure di ripristino più comuni utilizzate nella gestione quotidiana dei repository.

1. Gestione dei Branch e Workflow Quotidiano

Il flusso di lavoro si articola tipicamente su due rami principali: `dev` (sviluppo, integrazione e test) e `main` (stabile, rilasciato in produzione).

I comandi base per muoversi:

```
# Mostra tutti i branch locali e indica su quale ci troviamo
git branch

# Crea un nuovo branch a partire da quello attuale e ci si sposta dentro
git checkout -b nome-nuovo-branch

# Passa a un branch già esistente (es. torna su dev)
git checkout dev
```

Allineamento manuale da dev a main (senza lo script `merge.sh`):

Se vogliamo fare l'unione dei rami a mano, la sequenza corretta è questa:

```
# 1. Salviamo tutto sul nostro branch di sviluppo
git checkout dev
git add .
git commit -m "Descrizione modifiche"
git push origin dev

# 2. Spostiamoci su main e allineiamo col server
git checkout main
git pull origin main

# 3. Uniamo dev dentro main
git merge dev

# 4. Spingiamo in produzione
git push origin main

# 5. Torniamo su dev e riallineiamo con le ultime modifiche di main
git checkout dev
git merge main
```

---

2. Gestire le Pull Request (PR) in Locale

Se un contributore esterno apre una Pull Request (PR) verso il nostro repository, non dobbiamo mai fare il merge alla cieca. Conviene scaricarla in locale su un branch temporaneo per testarla con `zola serve` e capire cosa sia stato modificato.

Forgejo (e Gitea/GitHub) mappa le PR con un riferimento speciale: `pull/[ID_PR]/head`.

```
# 1. Scarichiamo la PR in locale creando un branch temporaneo (es. pr-12 per la PR #12)
git fetch origin pull/12/head:pr-12

# 2. Spostiamoci sul branch temporaneo appena creato
```

```
git checkout pr-12

# 3. Testiamo il sito localmente (le modifiche sono attive qui!)
zola serve
```

Pulizia post-test:

Una volta verificata la bontà della PR:

```
# Torniamo sul nostro branch di sviluppo
git checkout dev

# Cancelliamo il branch temporaneo della PR che non serve più
git branch -D pr-12
```

A questo punto possiamo effettuare il merge cliccando sul pulsante verde dell'interfaccia web di Forgejo. Ricordiamoci poi di eseguire un `git pull origin dev` per scaricare le modifiche approvate nella nostra copia locale.

---

3. Risolvere i Conflitti di Merge

Il conflitto si crea quando noi e un altro utente (o una PR) abbiamo modificato **le stesse identiche righe dello stesso file**. Git non sa quale versione tenere e si ferma chiedendo il nostro intervento.

Come procedere:

Se durante un merge (ad esempio `git merge dev` o `git merge pr-12`) riceviamo un errore di tipo `CONFLICT (content): Merge conflict in content/articolo.md`:

1. **Apriamo il file incriminato col nostro editor di testo.**
2. **Cerchiamo i marcatori di conflitto** inseriti da Git:

```
<<<<<<< HEAD
Questo è il codice presente nel branch in cui ci troviamo attualmente (il
destinatario).
=======
Questo è il codice presente nel branch che stiamo cercando di unire (la
```

```
sorgente).
>>>>>>> dev
```

3. **Risolviamo il conflitto manualmente**: cancelliamo i marcatori (`<<<<<<<`, `=======`, `>>>>>>>`) e modifichiamo il testo lasciando solo la versione corretta (o unendole con criterio).
4. **Registriamo la risoluzione**:

```
# Aggiungiamo il file risolto all'area di stage
git add content/articolo.md

# Concludiamo il merge con un commit
git commit -m "Risolto conflitto di merge"
```

4. Pronto Soccorso Git (Quando le cose vanno a rotoli)

Ecco le classiche situazioni di panico in cui ci si ritrova almeno una volta.

"Abbiamo sbagliato il messaggio dell'ultimo commit" (Non ancora pushato)

Se abbiamo fatto un commit ma ci siamo accorti di aver commesso un errore nel testo del messaggio:

```
git commit --amend -m "Messaggio corretto e pulito"
```

"Abbiamo dimenticato di aggiungere un file nell'ultimo commit" (Non ancora pushato)

Se abbiamo fatto il commit ma ci siamo accorti che mancava un file o una correzione al volo:

```
git add file-dimenticato.md
git commit --amend --no-edit
```

Questo comando inserisce la modifica direttamente nell'ultimo commit, senza cambiare il messaggio.

"Vogliamo annullare l'ultimo commit ma TENERE le modifiche"

Se abbiamo fatto un commit ma vogliamo "scomporlo" per modificare i file o fare commit separati:

```
git reset --soft HEAD~1
```

Il commit sparisce, ma i nostri file rimangono modificati nell'area di stage.

"Vogliamo annullare l'ultimo commit e BUTTARE VIA tutte le modifiche" (Punto di non ritorno!)

Se vogliamo cancellare l'ultimo commit e ripristinare i file a come erano in precedenza:

```
git reset --hard HEAD~1
```

Attenzione: questo comando cancella definitivamente le modifiche non committate.

"Vogliamo annullare tutte le modifiche locali non salvate"

Se abbiamo modificato dei file, ci fanno onco e vogliamo tornare all'ultimo commit pulito:

```
# Ripristiniamo i file tracciati modificati
git restore .

# Rimuoviamo eventuali file o cartelle nuove non tracciate
git clean -fd
```

---

5. La Macchina del Tempo: `git reflog`

Se abbiamo eseguito un `git reset --hard` nel posto sbagliato, o abbiamo cancellato un branch credendo che non servisse più, potremmo pensare di aver perso tutto. Fortunatamente Git non cancella nulla immediatamente.

Il **reflog** è il registro interno di Git che traccia ogni singolo spostamento dell'HEAD (la testina di lettura).

1. **Visualizziamo la cronologia dei nostri passi:**

   ```
   git reflog
   ```

   Vedremo una lista simile a questa:

   ```
   ab12cd3 HEAD@{0}: reset: moving to HEAD~1
   ef45gh6 HEAD@{1}: commit: Aggiunto articolo sulla crittografia
   ij78kl9 HEAD@{2}: checkout: moving from dev to main
   ```

2. **Troviamo il punto in cui tutto funzionava**: ad esempio `HEAD@{1}` (prima del reset errato).
3. **Torniamo indietro nel tempo:**

   ```
   git reset --hard HEAD@{1}
   ```

   I file torneranno allo stato esatto in cui si trovavano in quel momento.

Fluxer Icon

Canale di discussione e supporto

Canale di discussione e supporto. Per confrontarti su configurazioni o altro entra su fluxer.dreadful.work (https://fluxer.dreadful.work/invite/dreadful).