Git

staging area
`git init`
`.git`
`git status`
`git add "....."`
`commit`
`commit -m`
`git log`
`git checkout`
`git reset "....."`

branch: ramificazione del nostro branch principale chiamato main oppure master

`git checkout -b <.....>`:comando per creare un nuovo branch

`git branch`:per vedere la lista dei branch

`git switch`:mi sposto tra i vari branch

`git merge commit`:gestisce i casi di conflitti e crea un nuovo commit

`git squash`:unisce i commit creando un unico commit

`git rebase`:gli ultimi commit hanno la priorità sugli altri commit
il rebase cambia la storia del nostro progetto

`git branch -m`:comando per modificare il nome del branch(prima di modificare il nome del branch spostarsi sul branch da modificare)

`git branch -D`:comando per eliminare il branch(prima di eliminare il branch bisogna spostarsi su un altro branch)

git kraken git lens

`git diff`: questo comando fa vedere le modifiche effettuate al file

rebase interactive

`<h color="orange">git rebase -i</h>`: Questo comando avvia un rebase interattivo, permettendo di modificare, riordinare o combinare i commit in modo flessibile.

`reword` : modifica il messaggio del commit

`git rebase --abort`: Questo comando viene utilizzato per annullare un'operazione di rebase in corso e riportare il repository allo stato in cui si trovava prima dell'inizio del rebase. È utile quando si incontrano conflitti o si decide di non voler procedere con il rebase.

Punti chiave su `git rebase --abort`:

- Interrompe completamente il processo di rebase
- Ripristina HEAD alla sua posizione originale
- Tutte le modifiche effettuate durante il rebase vengono scartate
- È sicuro da usare se non si è certi di voler continuare un rebase complesso

Il Git Conventional Commit è una convenzione per la scrittura dei messaggi di commit che aiuta a creare una cronologia di commit esplicita e strutturata. Ecco alcuni punti chiave:

- Struttura: `&lt;type&gt;(&lt;scope&gt;): &lt;description&gt;`
- Tipi comuni: `feat` (nuova funzionalità), `fix` (correzione bug), `docs` (documentazione), `style` (formattazione), `refactor`, `test`, `chore` (manutenzione)
- Scope: indica la parte del codice interessata (opzionale)
- Description: breve descrizione delle modifiche

Vantaggi dell'utilizzo dei Conventional Commits:

- Generazione automatica di changelog
- Determinazione automatica del versionamento semantico
- Comunicazione chiara della natura dei cambiamenti ai colleghi
- Facilita l'analisi e la comprensione della storia del progetto

Esempio di un commit convenzionale:

```
feat(auth): add login functionality for users
```

`git fixup`: Questo comando è utilizzato per creare un commit di correzione che verrà automaticamente combinato con il commit precedente durante un rebase interattivo. È utile per apportare piccole modifiche o correzioni a commit recenti.

`git commit –amend` : serve a commitare il lavoro al ultimo commit senza generarne uno aggiuntivo

`git commit --fixup` : Questo comando crea un commit di correzione che sarà automaticamente combinato con il commit specificato durante un rebase interattivo. È utile per apportare modifiche o correzioni a commit specifici nella storia del repository.

Punti chiave su `git commit --fixup`:

- Crea un commit con un prefisso speciale che indica che è un fixup
- Durante un rebase interattivo, questi commit vengono automaticamente spostati e combinati con il commit originale
- Aiuta a mantenere una storia pulita del repository
- Può essere usato in combinazione con `git rebase -i --autosquash` per un processo di pulizia automatizzato

😎 RIASSUNTO

`git cherry-pick`: Questo comando permette di applicare commit specifici da un branch all'altro.

`git stash`: Questo comando salva temporaneamente le modifiche non committate, permettendo di cambiare branch o eseguire altre operazioni senza committare lavoro incompleto.

`git stash push -m<……..>`

`git stash list`

`git stash apply`

`git stash -help`: per vedere la documentazione interna del comando(si puo usare per qualsiasi comando)