Primi esercizi Git+Bash

[**1. Inizializzazione di un repository e primo commit 1**](#_vtyxqkac0k28)

[**2. HEAD manipulation + merge 1**](#_ogo1n981dvv0)

[**3. Merging example… 1**](#_ewwuqd9mibe8)

[**4. Repo interattivo 2**](#_3258wjmhzxx5)

[**5. Repo Git con branch e commit su file multipli 2**](#_wb2mgbpg5dqw)

[**6. Repo Git con branch e commit su file multipli e merge 3**](#_b1o0lo8257ny)

### Inizializzazione di un repository e primo commit

Obiettivo: Creare uno script che inizializzi un repository Git, aggiunga file e faccia il primo commit.

* 1. Scrivi uno script Bash che:
     1. Crei una nuova directory chiamata repo\_esercizio.
     2. Inizializzi un repository Git all'interno della directory.
     3. Crei un file README.md con un testo a tua scelta.
     4. Aggiunga il contenuto di README.md all’index.
     5. Esegua il commit del file con il messaggio "First commit".
        1. **Implementa anche la versione dove il messaggio di commit viene**
           1. **Inserito dall’utente**
           2. **Inserito tramite l’editor di testo Nano**

### HEAD manipulation + merge

* 1. Scrivi uno script bash che
  2. Inizializza un repository in $HOME/Desktop chiamato HeadManipulationRepo
  3. ci accede
  4. Crea tre file (A,B,C) e ci inserisce una riga.
  5. Per ogni file eseguite un commit
  6. Crea un branch *devTemp* e in questo branch apportate delle modifiche a fileB.
  7. Mostrate la cronologia dei commit per osservare le divergenze
     1. Provate il comando **git log --oneline --graph --all**
        1. graph evidenzia le diramazioni nel repo
        2. all mostra i commit su tutti branch, il default è solo sul corrente
     2. Prova un merge tra i due branch. Eseguite il merge da main
     3. Mostrate la cronologia dei commit post-merge per osservare la situazione
     4. Controlla lo stato del repo per eventuali conflitti

### Merging example…

* 1. Crea una script che:
     1. Inizializza un nuovo repo in un cartella chiamata MergeRepo
     2. per 3 volte:
        1. Aggiunge all’index un nuovo file (file1, file2, file3) con una riga di intestazione
        2. Esegue un commit
     3. Crei due branch featureA e featureB
     4. Modifichi la stessa riga in file2.txt in entrambi i branch
     5. Visualizzi lo stato del repo
     6. Esegua il merge delle modifiche di featureA nel main
        1. Che succede? Spiega
        2. Durante il merge del branch featureA nel branch main, il processo di fusione si conclude con successo poiché non si verifica alcun conflitto.

Questo accade perché nel commit di branch featureA il file 2 è stato modificato mentre nell’ultimo commit di branch main, prima che avviene la diramazione, il file 2 non è ancora stato modificato.

Poiché non c’è sovrapposizione nelle modifiche alle stesse righe del file, Git è in grado di applicare senza problemi le modifiche provenienti dal branch featureA al branch main.

* + 1. Esegua ora il merge di featureB su main
       1. Cosa succede? 💣
       2. Durante il merge di featureB nel branch featureA, il processo di fusione non si conclude con successo ma avviene un conflitto.

Questo si verifica perché nel commit di featureA il file 2 è stato modificato e stessa cosa nel commit di featureB.

Poiché c’è sovrapposizione nelle modifiche provenienti dal branch featureA e featureB questo causa inevitabilmente un errore che git lascia da risolvere a chi ha eseguito il merge.

### Repo interattivo

* 1. Per questo esercizio non scrivere uno script ma usa la shell in modalità interattiva.
  2. Crea un repo git su desktop chiamato repoInterattivo
     1. Cosa succede se si chiama un comando come git log prima di entrare in un repo git?
     2. Quando si chiama un comando git log prima di entrare in una repo git succede che esce un messaggio di errore dove dice che non è una repo git
  3. Al suo interno crea 5 file txt vuoti: file1.txt,...., file5.txt
  4. Visualizza lo stato del repository e commenta ciò che vedi.
     1. Facendo git status ho visto 5 file precedentemente creati. Git mi dice che questi 5 file non sono ancora stati tracciati e quindi non sono ancora stati aggiunti all’index questo significa che non saranno presenti al prossimo commit. Dobbiamo notare che nella situazione attuale non è ancora possibile fare un commit perché non c’è nulla nell’index
  5. visualizza lo storico dei commit creati finora.
     1. Commenta ciò che Git presenta in console.
     2. Facendo git log –oneline visualizzo un errore dove dice “che il tuo branch non ha ancora commits”. Questo accade perché come spiega git, non si è ancora effettuato commits
  6. Inserisci una riga di testo a piacere su file1.txt
     1. prova a usare nano invece che echo…
  7. Visualizza lo stato del repository e commenta ciò che vedi.
     1. Facendo git status ho visto 5 file precedentemente creati. Git mi dice che questi 5 file non sono ancora stati tracciati e quindi non sono ancora stati aggiunti all’index questo significa che non saranno presenti al prossimo commit. Dobbiamo notare che nella situazione attuale non è ancora possibile fare un commit perché non c’è nulla nell’index. Inoltre anche se abbiamo modificato file1.txt, nel git status non vediamo la voce che indica che file1.txt è stato modificato questo perché precedentemente fil1.txt non è stato aggiunto all’index quindi qualsiasi modifica da utente, affinchè il file non viene aggiunto all’index, esso non viene visualizzato come index.
  8. Aggiungi all’area di staging file1.txt
  9. Visualizza lo stato del repository e commenta ciò che vedi.
     1. Facendo git status vedo per prima cosa che il file1.txt è stato aggiunto all’index ovvero all’area di staging, inoltre lo visualizzo di colore verde e osservo che git mi fornisce un comando per poterlo togliere dall’index, tracciamento. File1.txt a questo punto siccome si trova nell’area di staging è possibile committarlo. Osservo anche 4 file txt che sono in colo rosso e questo sta a indicare che non sono ancora stati tracciati quindi non saranno presenti al prossimo commit
  10. Modifica file1.txt e file2.txt
  11. Visualizza lo stato del repository e commenta ciò che vedi.
      1. Noti qualcosa di particolare?
      2. si osservo qualcosa di particolare facendo ora git status. Per prima cosa c’è il file1.txt che è nello staging ma vedo anche che git indica che non sono state tracciate le modifiche effettuate a file1.txt dopo che esso è stato aggiunto a file1.txt; quindi al prossimo commit file1.txt viene committato mentre le modifiche effettuate al file no perché non sono state aggiunte all’index. Infine continuo a vedere i 4 file che ancora non sono stati tracciati da git. Appunto per le modifiche effettuate a file 2 è valida sempre la spiegazione di sopra ovvero che finche il file non viene aggiunto all’index tutte le modifiche effettuate a esso non sono visualizzate come modifiche
  12. Esegui il commit C1 contenente solo le modifiche a file1 e file2
  13. visualizza lo storico dei commit
  14. Visualizza lo stato del repository e commenta ciò che vedi.
      1. Ora non visualizzo più che il file1.txt è stato tracciato, le modifiche di file1.txt e il file2.txt non sono state tracciati. Tuttavia continuo a vedere che file 3, 4 e 5 non sono ancora stati stracciati.
  15. Crea un branch F1
  16. modifica file3.txt ed esegui un commit su main, non su F1
  17. Sposta HEAD su F1
      1. visualizza il contenuto della cartella e commenta.
      2. Ci sono tutti i file?
      3. Osservo che non ci sono tutti i file ma manca il file 3 questo perché non è stato aggiunto a questo commit ma è stato aggiunto al commit successivo. Inoltre vedo ancora il file 4 in entrame le posizioni perché esso non è ancora stato tracciato e quindo lo vedo ovunque mi sposto
  18. Sposta file4.txt dentro ad un cartella chiamata **Saves**
      1. git mv o mv? Provate prima git mv e poi mv. Spiegate il comportamento.
      2. Ho osservato che spostando il file con git mv mi dava errore mentre spostando il file con mv lo faceva tranquillamente. Questo perché con git mv postiamo solo file che sono già stati aggiunti all’index o sono stati committati mentre con mv spostiamo tutti i file che non sono ancora stati tracciati
  19. sposta file1.txt dentro a Saves. Ricorda che file1.txt è sotto controllo versione
      1. uso git mv o mv?
      2. Uso git mv appunto perché git mv viene usato per spostare solo file che sono già stati aggiunti all’index o sono stati committati
  20. rimuovi file2.txt solo dall’index ma tienilo nell WD
  21. Visualizza lo stato del repository e commenta ciò che vedi
      1. Visualizzo il file 1 che si trova nell’index ma è stato rinominato questo perché quando sposto un file già committato ritorna allo stato di staging e inoltre il comando git mv o mv oltre a muovere il file, lo rinominano, ecco la ragione per la quale lo vediamo come rinominato ma anche spostato. Mentre il file 2 era già stato committato ma siccome abbiamo deciso di rimuoverlo dal commit e tenerlo solo nel computer questo comporta che il file ritorna allo stato di non tracciato. Infine come sempre vedo il file 5 che non è ancora stato tracciato.

### Repo Git con branch e commit su file multipli

**Posizionamento**: Lo script deve posizionarsi nella directory $HOME/Desktop.

**Inizializzazione del Repository**: Crea una nuova directory chiamata branchRepo, entra nella directory e inizializza un repository Git.

**Creazione File Iniziali**: All’interno della directory branchRepo, crea i file file1.txt, file2.txt e file3.txt.

**Commit iniziale su Main (C1)**:

* Scrivi il testo Prima modifica per C1 in file1.txt.
* Esegui un commit con il messaggio C1: Prima modifica su file1.txt.

**Branch F1 e Commit (C2)**:

* Crea un branch F1 partendo dal commit C1.
* Aggiungi il testo Modifica su F1 per C2 in file2.txt e crea un commit con il messaggio C2: Modifica su file2.txt in F1.

**Ritorno su Main e Commit (C3)**:

* Torna su main e aggiungi il testo Modifica su main per C3 in file3.txt.
* Crea un commit con il messaggio C3: Modifica su file3.txt su main.

**Branch F2 e Commit (C4)**:

* Crea un branch F2 partendo dal commit C3.
* Aggiungi il testo Modifica su F2 per C4 in file1.txt e crea un commit con il messaggio C4: Modifica su file1.txt in F2.

**Ritorno su Main e Commit (C5)**:

* Torna su main e aggiungi il testo Modifica su main per C5 in file2.txt.
* Crea un commit con il messaggio C5: Modifica su file2.txt su main.

**Creazione di Branch Multipli con Commit su File Alternati**:

* Crea tre nuovi branch chiamati HF3, HF4, HF5 partendo da main.
* Su ciascun branch, esegui un numero di commit pari al numero del branch (3 commit su HF3, 4 commit su HF4, 5 commit su HF5).
* Per ogni commit su questi branch:
  + Alterna i file tra file1.txt, file2.txt, e file3.txt.
  + Per esempio, per HF3, esegui Commit 1, Commit 2, e Commit 3, modificando a turno i tre file.
    - L’operatore % potrebbe aiutare…
  + Usa il messaggio Commit X su <branch\_name> in <file\_name> (ad esempio Commit 1 su HF3 in file1.txt).

**Conclusione**: Riporta il branch corrente a main e stampa “REPO $nomeRepo CREATO” e visualizza la storia di tutti i commit su tutti i branch.

### Repo Git con branch e commit su file multipli e merge

* 1. Scrivi uno script Bash che esegua le seguenti operazioni per simulare un progetto Git

**Posizionamento**: Lo script deve posizionarsi nella directory $HOME/Desktop.

**Inizializzazione del Repository**: Crea una nuova directory chiamata mergeRepo, entra nella directory e inizializza un repository Git.

**Creazione File Iniziali**: All’interno della directory mergeRepo, crea i file fileA.txt, fileB.txt e fileC.txt.

**Commit iniziale su main (C1)**:

* Scrivi “Prima modifica su fileA.txt per C1” all'interno di fileA.txt.
* Aggiungi il file al repository ed esegui un commit con il messaggio C1: Prima modifica su fileA.txt.
* Usa git log --oneline per ottenere un riepilogo della cronologia fino a C1.

**Creazione e Commit su Feature1 (C2)**:

* Crea un branch Feature1 partendo da main.
* Scrivi “Modifica su Feature1 per C2” all'interno di fileB.txt.
* Esegui un commit con il messaggio C2: Modifica su fileB.txt in Feature1.
* Usa git log --graph --oneline per visualizzare la struttura del repository.

**Creazione e Commit su Feature2 (C3)**:

* Torna su main e crea un branch Feature2.
* Scrivi Modifica su Feature2 per C3 all'interno di fileC.txt.
* Esegui un commit con il messaggio C3: Modifica su fileC.txt in Feature2.
* Usa git log --graph --oneline --decorate per verificare la struttura e i branch attivi.

**Merge di Feature1 su main**:

* Torna su main.
* Esegui il merge del branch Feature1 su main.
* Usa git log --oneline --graph --decorate --all per visualizzare la struttura dopo il merge di Feature1.
* Usa git status per confermare lo stato del repository.

**Creazione di un Branch Hotfix e Commit**:

* Crea un branch Hotfix partendo dall'ultimo commit di main.
* Aggiungi Correzione per Hotfix in fileA.txt.
* Esegui un commit con il messaggio Hotfix: Correzione su fileA.txt.

**Merge di Hotfix su main**:

* Torna su main.
* Esegui il merge del branch Hotfix su main.
* Usa git log --oneline --graph --decorate --all per visualizzare la struttura dopo il merge di Hotfix.
* Usa git status per confermare il completamento del merge.

**Merge Finale di Feature2 su main con Risoluzione Manuale dei Conflitti**:

* Torna su main.
* Esegui un merge di Feature2 su main. In caso di conflitti in fileC.txt, il merge si interromperà e verrà visualizzato un messaggio di conflitto.
* **Risolvi il conflitto manualmente**: Apri fileC.txt con l’editor nano, risolvi il conflitto manualmente e salva il file.
* Aggiungi il file risolto e crea un commit con il messaggio Risoluzione conflitti per Feature2.

**Monitoraggio Finale**:

* Esegui git log --graph --oneline --decorate --all per visualizzare la struttura finale completa del repository.
* Stampa Script completato per indicare la fine dello script.