Comandi in Shell

Andrea Savastano

December 28, 2024

Contents

1 Commandi git

Introduzione

Questa sezione presenta un elenco di comandi git per gestire la propria repository in locale e per interagire con la repository in remoto sul server GitHub.

Si utilizza la shell `GitBash` perche è più simile a quella di `Linux`, in alternativa si può usare `PowerShell` o `cmd` di `Windows` che invece richiedono i comandi nativi di `Windows`.

E' possibile avere una repository git su un qualsiasi path (sul Desktop).
Una cartella si definisce repository git in locale al dispositivo quando è presente la sottocartella nascosta `.git`, che consente di usare i relativi comandi.
La repository in locale è diversa da quella che si ha su un server remoto ad esempio su GitHub, ed è possibile collegarle in modo che le modifiche (aggiunte o rimozioni di file) effettuate nella repository locale vengano caricate sul server, ossia sulla rispettiva repository remota.

1.1 Inizializzazione della repository

Se si desidera collegare la repository locale ad una remota, allora assicurarsi di aver creato prima la repository in remoto su GitHub (`https://github.com/new`) e di averne prelevato l'`"URL-remoto"`, che può essere HTTPS o SSH.

1.1.1 git clone

```
1 git clone "URL-remoto"
```

Clona la repository remota specificata dall'URL nel percorso in cui è stato eseguito il comando, creando una nuova cartella con lo stesso nome del remoto e aggiornata con gli stessi dati. Crea quindi una repository locale (`.git`), collegandola direttamente a quella remota.
In particolare si crea un branch `main` in locale che è collegato a quello remoto `remotes/origin/main`.
E' un comando molto più comodo e veloce per la creazione e il collegamento della repository, ma in alternativa si possono usare i comandi `git init` e `git remote`.

1.1.2 git init

```
1 git init
```

Inizializza la cartella in cui si esegue il comando come una nuova repository in locale (`.git`).
E' possibile lavorare solo in locale (tramite comandi come `git add`, `git commit`, ...), in quanto la repository locale non è stata collegata a nessuna repository in un server remoto.

1.1.3 git remote

```
1 git remote add origin "URL-remoto"
```

Collega la repository locale creata con `git init` con una remota già esistente.
In seguito a questo passaggio si ha che la repository in locale è collegata ad una nuova in remoto.

```
1 git remote set-url origin "URL-remoto"
```

Collega la repository locale ad un altro repository remoto, modificando l'URL del remoto esistente. Questo può servire quando si vuole collegare il repository locale a quello remoto con l'URL SSH, sostituendolo a quello HTTPS, o viceversa.
Oppure può servire per sostituire l'URL vecchio del repository remoto con uno nuovo, perchè è stato eventualmente cambiato.

```
git remote -v
```

Mostra l'elenco dei repository remoti associati al repository locale, insieme agli URL per `fetch` e `push`.

```
# output:
origin  https://github.com/savaava/ShellCommands.git (fetch)
origin  https://github.com/savaava/ShellCommands.git (push)
```

Il nome dei repository remoti è di default `origin` e viene mostrato il repository remoto, tramite il suo URL, da cui si prelevano le commit (fetch) e quello su cui si caricano le commit (push).

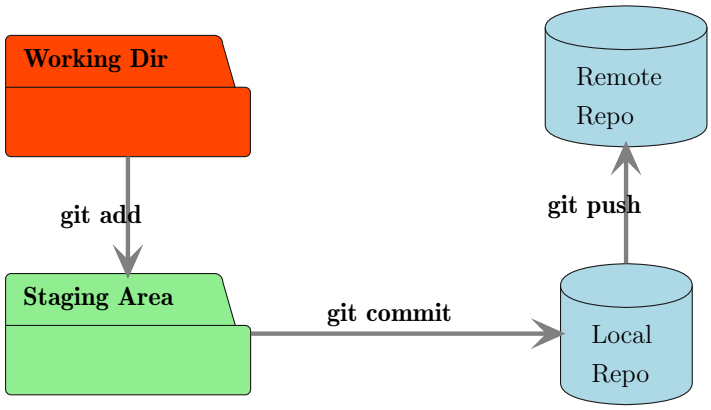
1.2 Aggiunta/caricamento

Quando si aggiorna la repository locale, quindi la working directory, aggiungendo/eliminando/modificando un file, è possibile effettuare una commit, la quale carica le modifiche in locale. Per aggiornare anche la repository remota a cui è collegata quella locale si deve effettuare una push della commit.

Pertanto, dopo aver aggiornato la repository locale, i passaggi da seguire sono 3:

1. `git add`
2. `git commit`
3. `git push`

I comandi si approfondiscono ai paragrafi successivi.



E' importante monitorare sempre lo stato della repository con `git status`, per tenere traccia dei cambiamenti in corso d'opera.

1.2.1 git status

```
git status
```

Mostra lo stato della repository locale, specificando i file che sono stati aggiunti, modificati o eliminati. Il comando in esame fornisce informazioni anche sui file che la repo sta monitorando:

- I file **Untracked** non sono ancora monitorati da Git (non aggiunti con `git add`), quindi sono i file aggiunti e non ancora caricati nella repo per la prima volta;
- I file **Tracked** sono già sotto il controllo di Git, quindi sono i file già caricati precedentemente nella repository, e possono essere **Unmodified**, **Modified** o **Deleted**.

I file Untracked, Modified e Deleted sono colorati in rosso nel terminale quando non sono stati ancora aggiunti alla staging area con `git add`. Una volta aggiunti saranno verdi.

```
1  # Output quando non sono state apportate modifiche:
2  On branch main
3  Your branch is up to date with 'origin/main'.
4
5  nothing to commit, working tree clean
```

In questo caso la repository in locale è sul branch `main` come suggerisce la riga 2, ed è collegato al branch remoto `origin/main` e sono entrambi aggiornati con gli stessi dati come suggerisce la riga 3. La riga 5 suggerisce che non vi sono modifiche aggiunte alla staging area per cui non c'è nulla da committare.

```
1  # Output quando e' stato creato, modificato ed eliminato un file:
2  On branch main
3  Your branch is up to date with 'origin/main'.
4
5  Changes not staged for commit:
6   deleted:    file.txt
7   modified:   file2.txt
8
9  Untracked files:
10  fileProva.sh
11
12 no changes added to commit
```

1.2.2 git add

```
1  git add <file>...
2  git add .
```

Aggiunge alla staging area i file specificati, che sono stati creati, modificati o eliminati.
Quindi il comando aggiunge alla staging area le modifiche apportate alla repository rispetto alla versione precedente.
E' efficace utilizzare direttamente `git add .` perchè aggiunge velocemente alla staging area tutte le modifiche apportate alla working directory, quindi tutti i file aggiunti (Untracked), tutti i file modificati (Modified) e tutti i file eliminati (Deleted).

Rispetto all'esempio sopra proposto che prevede tre modifiche alla repo, eseguiamo `git add .` per aggiunge tali modifiche alla staging area e analizziamo lo stato della repo tramite `git status`:

```
1  On branch main
2  Your branch is up to date with 'origin/main'.
3
4  Changes to be committed:
5   deleted:    file.txt
6   modified:   file2.txt
7   new file:   fileProva.sh
```

Si noti che ora le tre modifiche sono nella staging area e pronte per essere committate.

1.2.3 git commit

```
git commit -m "message"
```

Aggiunge le modifiche presenti nella staging area nella repository git locale e non in remoto.
In questo modo la repo locale e quella remota non sono più aggiornate con gli stessi dati e si può effettuare una push della/delle commit in sospeso.
E' necessario inserire sempre una stringa non vuota come messaggio. Questa può risultare utile per tracciare le modifiche in corso d'opera apportate alla repository e per questo è utile scrivere un messaggio rappresentativo della commit.

```
git commit -am "message"
```

Aggiungendo l'opzione `-a` è possibile saltare la staging area e aggiungere le modifiche direttamente alla repository locale come commit. Tuttavia non è in grado di aggiungere alla commit i file Untracked, i quali devono necessariamente passare per la staging area (`git add .`).

```
# Stato della repo quando e' stata effettuata una sola commit:
On branch main
Your branch is ahead of 'origin/main' by 1 commit.

nothing to commit, working tree clean
```

1.2.4 git push

```
git push
```

Carica le commit sulla repository remota a cui è collegata quella locale.
Di default la repository locale è sul branch `main` e `git push` carica le commit sul branch remoto `origin/main`.

1.2.5 git fetch

```
git fetch
```

Preleva le eventuali modifiche della repository remota e le carica in locale, senza aggiornare la working directory.

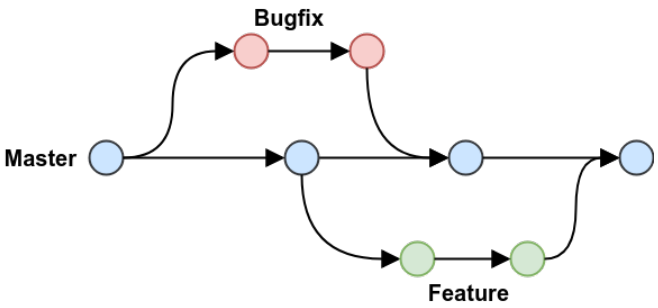
1.2.6 git pull

```
git pull
```

Preleva le eventuali modifiche della repository remota e le carica in locale, aggiornando la working directory, a differenza di `git fetch`.

1.3 Branch

I branch (rami) sono una delle potenzialità più rilevanti per Git perchè consentono ai contributori del progetto di lavorare in parallelo sullo stesso progetto che si trova sempre in remoto.
Un branch diverso da quello `main` (`Master` nell'immagine) è una versione del progetto diversa rispetto a quella originale perchè si caricano modifiche sul branch ausiliario e non più sul `main`.
Sarà possibile eliminare il branch e perdere eventuali modifiche, oppure effettuare una merge (fusione) con quello `main` e ricongiungersi al percorso originale.



Il branch rosso `Bugfix` e quello verde `Feature` sono branch ausiliari che si ricongiungono col `main`, dopo aver apportato modifiche per cui sono stati creati.

1.3.1 git branch

```
git branch "nome-branch"
```

```
git branch -a
```

```
git push --set-upstream origin "nome-branch"
```

```
git branch -d "nome-branch"
```

```
git push --delete origin "nome-branch"
```

1.3.2 git checkout

```
git checkout "nome-branch"
```

```
git checkout -b "nome-branch"
```

1.3.3 git diff

```
git diff "nome-branch"
```

1.3.4 git merge

```
git checkout main;
git merge "nome-branch"
```

1.4 Stash

1.5 Altri

1.6 Ripristinare commit

1.7 Configurazione

2 Comandi ssh

Introduzione

Questa sezione spiega come gestire una connessione SSH tra il dispositivo client e quello server utilizzando OpenSSH (Open Secure Shell), che consente di eseguire operazioni remote su altri computer attraverso una rete in modo sicuro sulla porta 22.

Si utilizza `PowerShell` di `Windows` e `GitBash` eseguiti come amministratore, in alternativa è possibile utilizzare `cmd` sempre come amministratore.

2.1 Inizializzazione dei dispositivi

2.2 Apertura/Chiusura della connessione

2.3 Operazioni remote

3 Comandi bash