Indice

Introduzione

Mio personale manuale di Git sviluppato per avere una base stabile e rapida dove salvare le mia conoscenze

Capitolo 1

Primi passi con Git

Innanzi tutto cerchiamo di definire cosa è Git e che tipo di applicazione è. Git rientra tra i programmi di versioning cioè sistemo che registrano, nel tempo, i cambiamenti ad un file o ad una serie di file, così da poter richiamare una specifica versione in un secondo momento. Per gli esempi di questo libro verrà usato il codice sorgente di un software come file sotto controllo di versione, anche se in realtà gli esempi si possono eseguire quasi con ogni tipo di file sul computer. Nello specifico Git è un sistema di gestione del software distribuito.

1.1 Stati del file

Git ha 3 possibili stati dei file del progetto :

- **MODIFIED** : file è stato modificato rispetto alla versione salvata ma non è stato selezionato per essere salvato nelle snapshot
- **STAGED** : file modificato e inserito nella lista dei file da mettere nella versione snapshot
- **COMMITTED** : file salvato nel database locale

Quindi in generale, guardando la soluzione da un altra prospettiva, possiamo dire che i file che modifichiamo, quelli che fanno parte del nostro progetto, della nostra aria di lavoro, sono le versioni modificate di quelli che ha in possesso il database di Git. Quando effettuiamo uno stage non facciamo altro che inserire i nomi dei file in un indice che rappresenta i file che faranno parte della snapshot. Mentre il commit non è altro che una raccolta di questi file e come se facessimo un istantanea, salviamo il loro contenuto in un database gestito da Git in locale sul nostro PC, questi file sono correlati da una serie di metadati che ne permettono

il recupero e la certezza che essi non possano essere modificati senza che Git non se ne accorga. Git mette a disposizione sia la classica riga di comando che delle interfacce grafiche ma per avere a disposizione tutto il set di funzionalità si consiglia il CLI.

1.2 Istallazione

Per quanto riguarda Windows e Mac basta scaricare l'applicativo ed istallarlo tramite le procedure guidate, mentre per linux in funzione della distribuzione i comandi possono essere i seguenti:

```
1    apt-get install git
2    yum install git
```

1.3 Impostare l'ambiente

E possibile configurare Git tramite il tool da riga di comando :

```
1    git config
```

Mentre i file che tengono tracia delle configurazioni sono :

- **/etc/gitconfig** : contiene i valori impostati dei singoli utenti nel sistema e dei loro repository se passiamo l'opzione *–system* al comando *git config* possiamo leggere e mofificare questo file
- **/.git/config /.config/git/config** : contiene i dati specifici per l'utente, per modificare questo file possiamo passare al comendo *git config* l'opzione *–global*
- **config** : è un file specifico per la repository del progetto specifico

In generale si prioritarizzano le impostazioni da quelle più vicine al progetto a quelle globali, quindi .gitconfig ha priorità superiore ad /etc/gitconfig.

1.4 Definire l'identità

Dopo aver installato git bisogna iniziare con i settaggi; il primo è quello relativo all'identità globale.

```
1    git config --global user.name "John Doe"
2    git config --global user.email johndoe@example.com
```

Questo settaggio è importate poiché queste sono le credenziali che Git utilizza per i commit. Queste inoltre saranno le credenziali che verra utilizzate di default per ogni operazione che si vuole effettuare. nel caso si vogliano sovrascrivere alcune di queste credenziali per specifici progetti basta ripetere il comando nella cartella principale del progetto senza l'attributo *–global* per modificare solo le opzioni locali.

1.5 Definire l'editor

Dopo aver impostato l'identità bisogna scegliere l'editor predefinito tramite il comando :

```
1       git config --global core.editor nome_editor
```

l'editor viene invocato da Git per mostrare delle informazioni come ad esempio i merge o per altre comunicazioni relative al codice.

1.6 Verificare le impostazioni

Per verificare le impostazioni inserite basta crivere nel CLI :

```
1     git config --list
```

Ottenendo come output :

```
1     user.name=John Doe
2     user.email=johndoe@example.com
3     color.status=auto
4     color.branch=auto
5     color.interactive=auto
6     color.diff=auto
7     ...
```

è possibile anche nostrare uno solo dei campi inserendo una key:

```
1     git config user.name
```

1.7 Comando help

Durante l'uso di Git ci sono 3 modi per poter chiedere informazioni al CLI :

```
1         git help <verb>
2         git <verb> --help
3         man git-<verb>
```

è possibile quindi richiamare o il manuale di qualsiasi comando in questi 3 modi ed inoltre questi comandi sono globali e possono essere richiati da qualsiasi finestra del CLI

Capitolo 2

Le basi di Git

In questo capitolo vedremo i comandi base per iniziare ad utilizzare Git. Saremo in grado di; inizializzare e configurare un *repository*, gestire il tracciamento di un file, lo stage e il commit, come ignorare alcuni file specifici o famiglie di file tramite pattern specifici, come tornare ad una versione precedente in maniera semplice e veloce, visualizzare lo storico dei commit e come scaricare o caricare i file in un *repository* remoto.

2.1 Inizializzare un repository

Esistono due modi per inizializzare un progetto e sono, o importare un progetto già esistente e caricarlo in Git oppure importare un repository già esistente da un server.

2.1.1 Inizializzare un repository da un progetto già esistente

Se vuoi iniziare a tenere traccia di un progetto già esistente bisogna aprire il CLI, navigare fino alla cartella principale del progetto e lanciare il comando :

```
1        git init
```

Questo creerà una sottocartella nascosta ***.git*** che conterrà i file di configurazione del nostro repository per il progetto. A questo punto abbiamo solo lo "scheletro" del nostro repository e ancora nessun file è controllato, nella sua evoluzione, da Git. Per permettere a Git di seguire l'evoluzione del tuo progetto devi aggiungere i file che vuoi inserire nel repository, tramite il comando :

```
1        git add *.c
```

```
2          git add LICENSE
3          git commit -m "inizializzo il progetto"
```

Cerchiamo di capire cosa è avvenuto quando abbiamo lanciato questi comandi; con il primo abbiamo aggiunto nell'area di stage tutti i file contenuti nella cartella principale del progetto che abbiano come estensione .c cioè file di codice C. Con il secondo comando abbiamo inserito nello stage il file *LICENSE* ed in fine con il comando *commit* abbiamo creato una *snapshot* dei file che viene salvata nel DB di Git.

2.1.2 Clonare un repository già esistente

Se vogliamo lavorare su di un progetto che è già stato creato e che risiede su di un server basta usare il comando :

```
1      git clone
```

Cosi facendo si scaricano quasi tutti i dati contenuti nel server, compresi tutte le modifiche fatte ai file, così facendo se il progetto sul server dovesse essere danneggiato tutti gli sviluppatori in possesso di un clone hanno una copia che può essere utilizzata per ripristinare i dati, come se avessimo un backup distribuito. Il comando clone si utilizza nel seguente modo :

```
1      git clone [URL]
```

In questo modo creerai una cartella con i file del progetto nella cartella dove lancerai il comando. Scaricando tutti gli ultimi file e impostando una cartella **.git**. I file scaricati saranno già pronti all'uso e alla modifica del codice. Possiamo inserire il progetto in una nostra cartella nella posizione attuale, aggiungendo un altro parametro cioè il nome dopo l'URL:

```
1      git clone [URL] nome_cartella_progetto
```

Il comando è lo stesso del precedente con l'aggiunta di un opzione. si possono utilizzare molti protocolli di comunicazione come ad esempio HTTPS, SSH, git://, ecc. Più in generale sono tutti formati da : ***user@server:path/to/repo.git***. Quando si scarica un progetto e si vogliono apportare alcune modifiche, i file che sono contenuti sono già tracciati, quindi git rivela le modifiche che hai apportato mentre se crei altri file questi sono nello stato non tracciato, quindi Git non vedrà se sono stati modificati o meno.

2.1.3 Registrare i cambiamenti nei repository

Dopo aver effettuato il clone avrai una copia dell'intero repository in locale, a questo punto puoi iniziare a modificare i file ed effettuare delle **commit** per tenere aggiornate tutte le modifiche effettuate. In generale i file del tuo progetto possono trovarsi in due stati: tracciati e non tracciati. I file tracciati sono file della quale git ne è al corrente e questi possono essere in tre condizioni : non modificati, modificati e in staged. I file non tracciati sono tutto il resto, in pratica tutti i file che sono nella directory di progetto ma che non erano nell'ultima snapshot e che non si trovano nell'area di staging. Quando si effettua un clone tutti i file sono tacciati e non modificati. Quando modifichi i file vengono visti da git come modifica e quindi bisogna effettuare una commit per aggiornare le modifiche. Questa immagine rappresenta gli stati nella quale si può trovare un file in git.

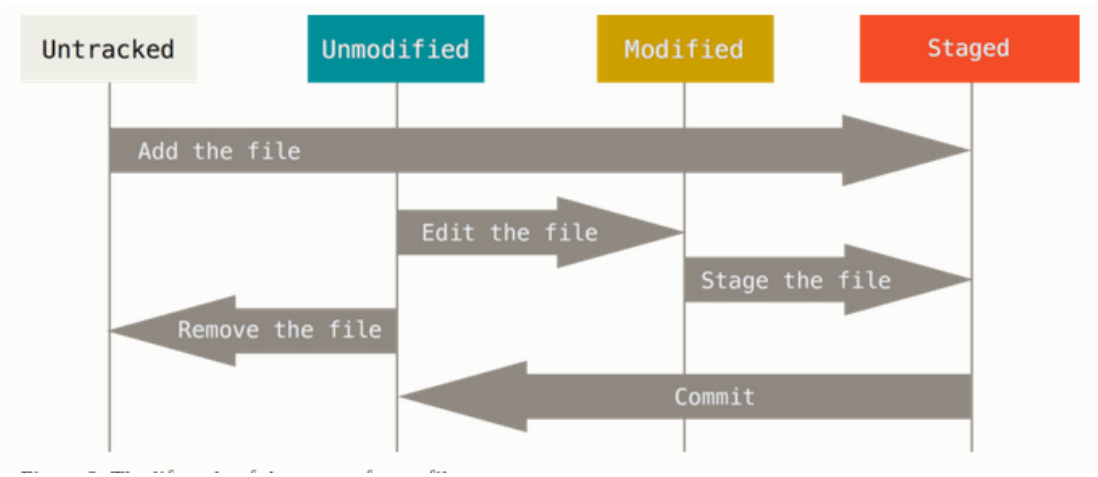

Figura 2.1: stati di un file in git

Controlare lo stato di un file

Lo strumento principale per conprendere lo stato dei file è il comando :

```
1    git status
```

un tipico status è il segunete :

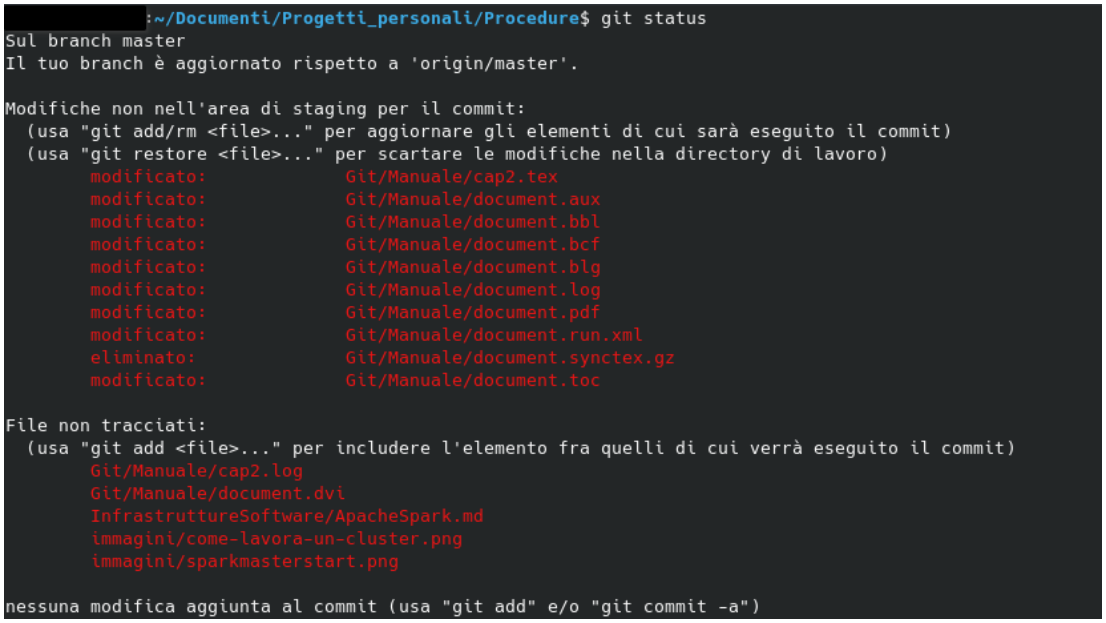

Figura 2.2: esempio di git status per la modifica di questo manuale

Questo output sta ad indicare che ci sono file che sono stati modificati nello specifico il file cap2.tex, quindi tracciati e poi possiamo notare che ci sono dei file che non sono stati tracciati, che quindi, sono stati creati prima dell'ultimo commit. Un ultima informazione che puoi trarre è il ramo di sviluppo nella quale ti trovi attualmente. I file non tracciati non verranno aggiunti alle snapshot finche non vengono esplicitamente aggiunti

Tracciamento di nuovi file

Per esplicitare che si vogliono tener traccia di file attualmente non tracciati possimo farlo con il comando :

```
1    git add <nome_file>
```

In alternativa possiamo utilizzare il comando :

```
1    git add .
```

per integrare tutti i file non tracciati all'interno del prossimo commit. Il comando **git add** accetta percorsi di file o directory, nel caso della directory aggiungerà tutti i file in essa contenuti.