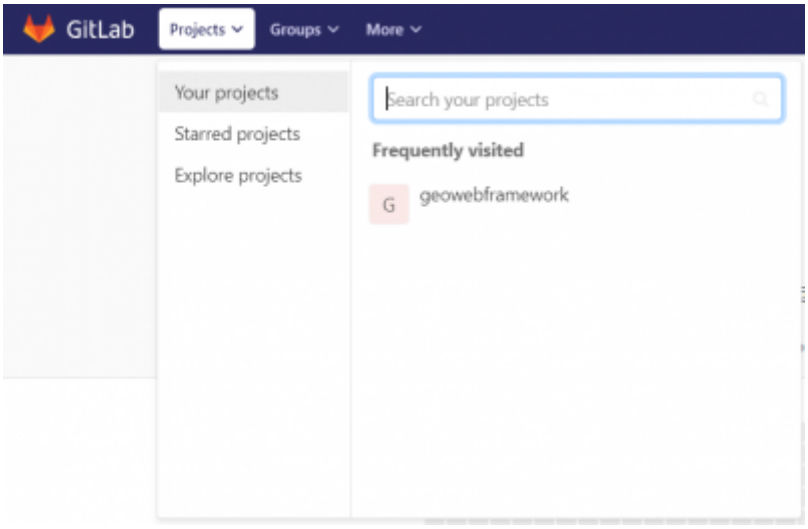
Creazione repository GIT di commessa

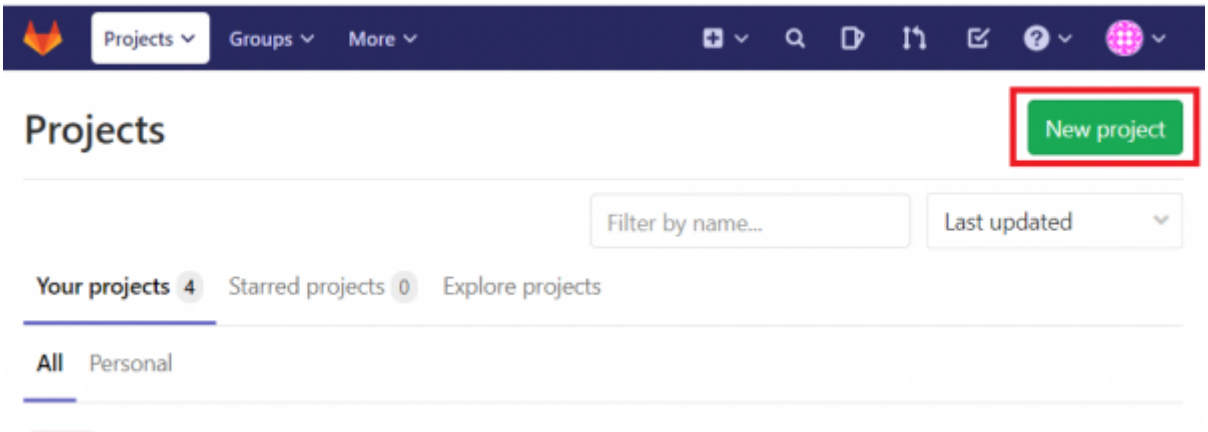
Un repository GIT di commessa può essere visto come una cartella che tiene traccia delle modifiche effettuate nel tempo a tutti i file in essa contenuti.

Prima di procedere alla creazione di un repository, bisogna aver scaricato e installato Git sul proprio pc: la guida ai passaggi necessari per farlo è consultabile nella sezione Installazione di Git.

La prima cosa da fare è la creazione di un nuovo progetto su Gitlab. Per fare questo, basta accedere al sito con il proprio account, aprire la sezione **Projects** dalla toolbar in alto a sinistra e, poi, selezionare **Your projects** dal menu a tendina, come mostrato in figura.

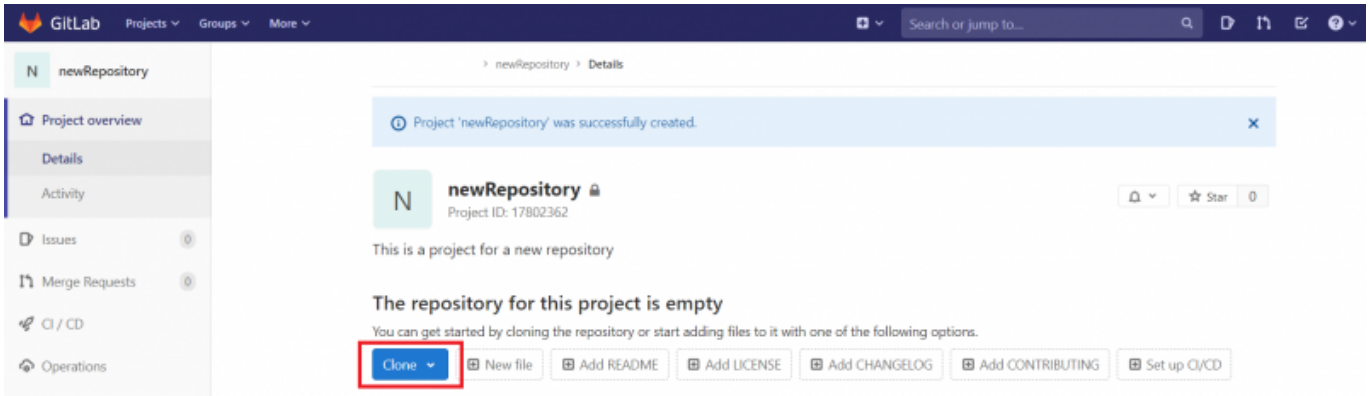


Selezionare il pulsante **New Project** e compilare, nella finestra che si aprirà, i vari campi con le informazioni desiderate, quali nome del progetto, privacy da impostare per il progetto, e così via.



Una volta salvate tutte le informazioni, il progetto (che rappresenta il nuovo repository) viene infine creato; esso è consultabile aprendo la sua pagina di dettaglio su Gitlab (dalla sezione **Projects → Your Projects**).

All'interno della pagina di dettaglio, è sempre presente un pulsante **Clone**: cliccando sul pulsante, si apre un menu a tendina che fornisce la possibilità di copiare il percorso del progetto per clonarlo nel proprio pc tramite protocollo SSH o HTTPS. Solitamente, su sistemi operativi Windows, è preferibile utilizzare il protocollo HTTPS per non incorrere in errori relativi a permessi o sicurezza.



Il processo di clonazione serve a creare, nel pc dove si sta lavorando, un repository locale di Git: ogni modifica effettuata nella cartella del proprio pc non viene automaticamente salvata nel repository locale, ma deve essere esplicitamente aggiunta ad esso e, poi, deve essere riportata sul repository remoto tramite la procedura di *commit and push*. Per clonare il repository nel proprio pc si possono utilizzare due metodi: l'interfaccia grafica di Git per Windows, che risulta più intuitiva ma un poco più laboriosa e, in alternativa, l'interfaccia a riga di comando, che è più rapida, più versatile e più potente ma anche un po' meno intuitiva. Nei prossimi paragrafi verranno presentate entrambe le modalità. Per completezza, si riporta anche il link alla guida ufficiale di Git, in cui si possono trovare le spiegazioni di tutti i comandi disponibili e delle varie opzioni per ogni comando: https://git-scm.com/docs.

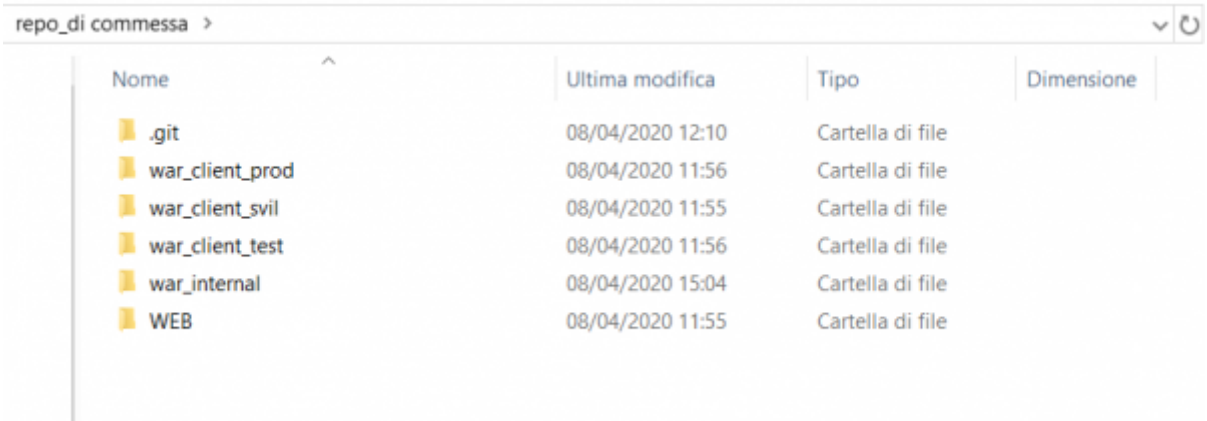
Sequenza operazioni tipiche

Clonazione repository

Vedi Git clone sotto.

Modifica repository tramite personalizzazione struttura

A questo punto, il responsabile di commessa deve decidere la struttura specifica da dare al progetto: si devono individuare gli ambienti necessari, che corrispondono ai diversi WAR, e creare, per ognuno di essi, una sotto-cartella. Un esempio di contenuto standard di un repository di commessa è riportato in figura:



In linea generale, quindi, si può dire che la cartella di repository deve contenere:

- un'unica cartella WEB per i contenuti statici dato che, solitamente, sono gli stessi per tutti gli ambienti;
- una cartella per l'ambiente interno;
- una cartella per ogni ambiente esterno.

Come variante le varie cartelle dei vari ambienti potrebbero ospitare ognuna la propria cartella WEB dei contenuti statici, ed eventuali cartelle conf/ aggiuntive

Eventuale personalizzazione e produzione war

Vedi la guida dedicata.

Salvatggio modifiche effettuate su repository git

Vedi Git commit e Git push sotto.

Clonazione e modifica del repository tramite riga di comando

In questo paragrafo saranno elencati i comandi principali di Git utilizzabili attraverso la Git bash: ciò significa che tali comandi hanno valore solo se digitati all'interno dell'interfaccia a riga di comando di Git, chiamata, appunto, Git bash. Questa interfaccia può essere aperta:

- come qualsiasi programma per Windows, tramite doppio click sulla sua icona;
- in un percorso specifico, cliccando con il tasto destro all'interno di una determinata cartella e scegliendo poi l'opzione **Git bash here**.

Git clone

Dopo aver copiato il percorso di clonazione da Gitlab, andare nella cartella in cui si vuole salvare il repository e digitare il comando:

```
git clone percorso_copiato_da_Gitlab
```

Così facendo, il repository sarà clonato all'interno della cartella. Oltre ai vari file presi da remoto, nel repository locale sarà sempre presente una sotto-cartella dal nome .git che non dovrebbe mai essere eliminata o modificata. Si noti che, in caso vengano richieste le credenziali, vanno inserite quelle con cui ci si è registrati a Gitlab.

Git status

Per controllare quali modifiche sono state effettuate nella cartella locale, basta digitare nella finestra di Git bash il comando:

```
git status
```

Questo comando mostra lo stato della cartella di lavoro: in caso di modifiche non ancora salvate nel repository remoto, verranno indicati quanti e quali file sono stati cambiati, eliminati o aggiunti; se, invece, tutte le modifiche sono state salvate in remoto e la cartella locale è allineata con quella remota, verrà mostrato il messaggio “nothing to commit, working tree clean”.

Git add

Per eseguire l'operazione di *commit and push*, quindi per riportare le modifiche locali anche sul repository remoto, bisogna inizialmente aggiungere i file che si vogliono salvare nell'elenco degli *staged files* tramite il comando “add”; in particolare, per salvare tutti i file bisogna utilizzare il comando

```
git add all
```

S possono anche scegliere i singoli file da aggiungere scrivendo

```
git add nome_del_file
```

Con questo comando, inoltre, è possibile utilizzare anche delle espressioni, in modo da raggruppare una certa tipologia di file ed aggiungere più elementi nello stesso momento. Ad esempio, per aggiungere alla lista degli *staged files* tutti quelli che hanno l'estensione TXT, basta scrivere:

```
git add *.txt
```

Per ulteriori opzioni e utilizzi del comando “git add”, si rimanda alla sezione specifica nella guida ufficiale di Git: https://git-scm.com/docs/git-add.

Git commit

Il passo successivo è l'esecuzione del *commit* vero e proprio dei file, utilizzando, appunto, il comando “git commit”. Per questa operazione, è sempre necessario aggiungere un breve commento così da descrivere le modifiche effettuate; il commento va aggiunto dopo l'opzione “-m”, come riportato qui di seguito:

```
git commit -m "This is a simple comment for a commit"
```

Per conoscere tutte le numerose opzioni del comando “git commit”, è possibile consultare la sezione apposita nella guida ufficiale: https://git-scm.com/docs/git-commit.

Git push

A questo punto, per terminare l'operazione e salvare definitivamente nel repository remoto, si deve

concludere con:

```
git push
```

Anche per il comando “git push” si possono utilizzare diverse ulteriori opzioni, elencate e spiegate nella sezione di riferimento della guida ufficiale di Git: https://git-scm.com/docs/git-push.

Git pull

Se si vuole aggiornare il proprio repository locale con le modifiche più recenti presenti in remoto, si deve utilizzare (all'interno della cartella in cui si trova il repository locale) il comando

```
git pull
```

Questo comando permette di recuperare e incorporare ai file già presenti nella propria cartella, tutte le modifiche fatte sul server remoto. Si noti, comunque, che questo implica che le modifiche avvenute in remoto andranno a sovrascrivere quelle nella cartella di lavoro, allineando perfettamente il repository locale con quello remoto: tutti i file aggiunti o modificati in locale saranno scartati se non presenti o non modificati in remoto.

Altre indicazioni sull'uso di “git pull” possono essere trovate nella guida ufficiale seguendo il link https://git-scm.com/docs/git-pull.

Git merge

Questo comando serve a unificare due o più cronologie di modifica di un repository. Attraverso Git, infatti, uno stesso repository remoto può essere clonato su percorsi diversi assegnando ad ogni percorso un nome differente: in questo modo, partendo da un repository remoto iniziale **master**, si avranno diversi cloni dei quali è possibile tenere traccia singolarmente. Ogni copia locale, infatti, dipende dal **master**, ma può essere alterata in maniera indipendente dalla altre: l'idea è quella di avere un albero di partenza da cui partono più rami (o *branches*) che seguono dei percorsi diversi. In questa configurazione, ogni ramo corrisponde a un ulteriore repository remoto che è, a sua volta, un sotto-repository del **master** iniziale. Di conseguenza, ogni *branch* locale può essere salvato in remoto con la procedura descritta prima (operazione di *commit* e *push*) arrivando ad avere, alla fine, più *branch* diversi tra loro, ma che fanno riferimento allo stesso repository di partenza.

In uno scenario del genere è possibile, con Git, riunire uno o più *branch* incorporando tutte le modifiche.

Prima di eseguire questa operazione, però, è necessario che i *branch* che si vogliono unificare siano stati correttamente salvati, cioè che le modifiche locali siano allineate con quelle remote (si può verificare la condizione con il comando “git status”). Git, infatti, opera considerando che le versioni valide di ogni *branch* siano sempre e solo quelle salvate nei corrispettivi repository remoti: tutte le modifiche non salvate in remoto verranno perse.

Chiaramente, per l'operazione di *merge* si deve partire da un primo *branch* e ci si può assicurare di fare riferimento al *branch* giusto con il comando:

```
git checkout nome_del_branch_di_partenza
```

A questo punto, per unire il primo ramo al secondo, basta utilizzare il comando:

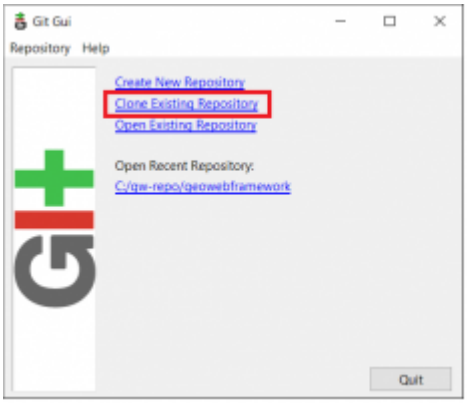
```
git merge nome_del_branch_da_unire
```

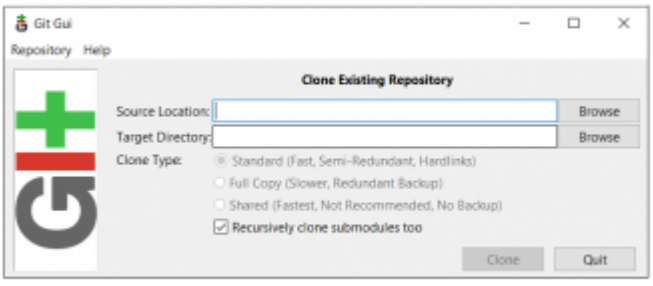
Se i due rami non hanno modificato gli stessi file, l'operazione terminerà così. In caso contrario, invece, Git riporterà un conflitto fra i *branch*, indicando quali file e quali parti di essi sono in contraddizione, e l'operazione di unione non verrà effettuata. Per completare il *merge* si dovrà andare a modificare i file che creano conflitto in modo da renderli uguali.

L'operazione di *merge* è un'operazione centrale di Git, e, per questo, esistono molteplici opzioni e utilizzi del comando “git merge”. Per approfondire è possibile consultare la guida ufficiale al seguente link: https://git-scm.com/docs/git-merge.

Clonazione e modifica del repository tramite interfaccia grafica

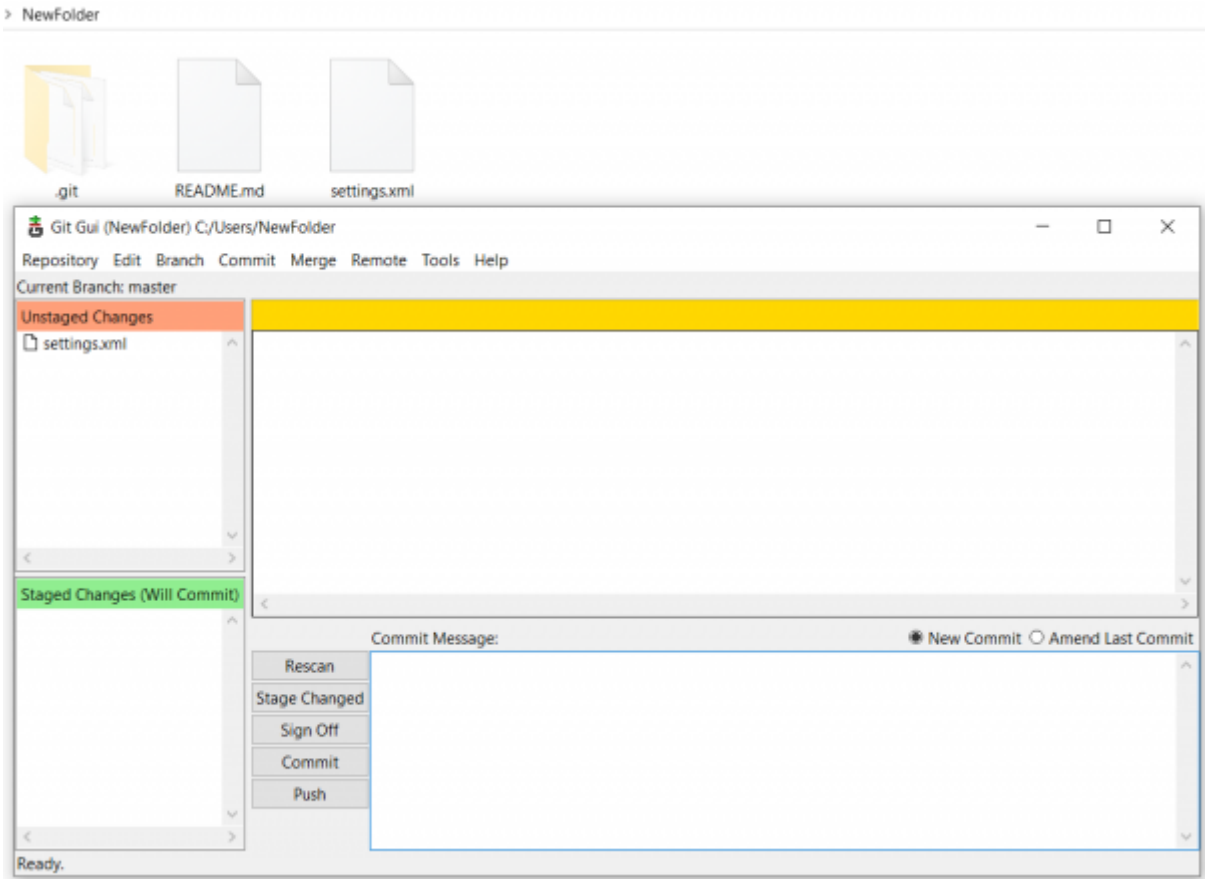
Aprire nel proprio pc il percorso su cui si vuole salvare il progetto, cliccare all'interno della cartella con il tasto destro e selezionare l'opzione **Git GUI here**. Nella finestra che viene aperta, scegliere **Clone existing repository** (come mostrato in figura) e, a questo punto, incollare nel form **Source location** il path precedentemente copiato dal progetto Gitlab; nel form **Target directory**, invece, va inserito il percorso, all'interno del proprio pc, all'interno del quale si vuole salvare il progetto. Procedere cliccando sul pulsante **Quit** e, dopo qualche istante, la clonazione del repository di Git sarà completata: il progetto e tutti gli eventuali file contenuti al suo interno saranno ora presenti anche nel proprio pc. Anche in questo caso, oltre ai vari file, sarà presente nel repository locale una sotto-cartella chiamata .git che contiene varie opzioni di configurazione e che non dovrebbe essere modificata o eliminata. Come nel caso precedente, in caso di richiesta, inserire le proprie credenziali di Gitlab.





Se è necessario modificare, aggiungere o eliminare un file contenuto nella cartella del proprio pc, tali modifiche devono poi essere riportate anche nel repository remoto di Gitlab tramite un'operazione di *commit* che serve ad allineare il repository Git locale e, in seguito, un'operazione di *push* per riportare tutte le modifiche anche nel repository remoto.

Per fare questo, andare nella cartella del proprio pc contenente il repository, cliccare con il tasto destro e, poi, selezionare **Git GUI here** per aprire l'interfaccia grafica di Git. La finestra che verrà aperta sarà simile a quella riportata in figura.

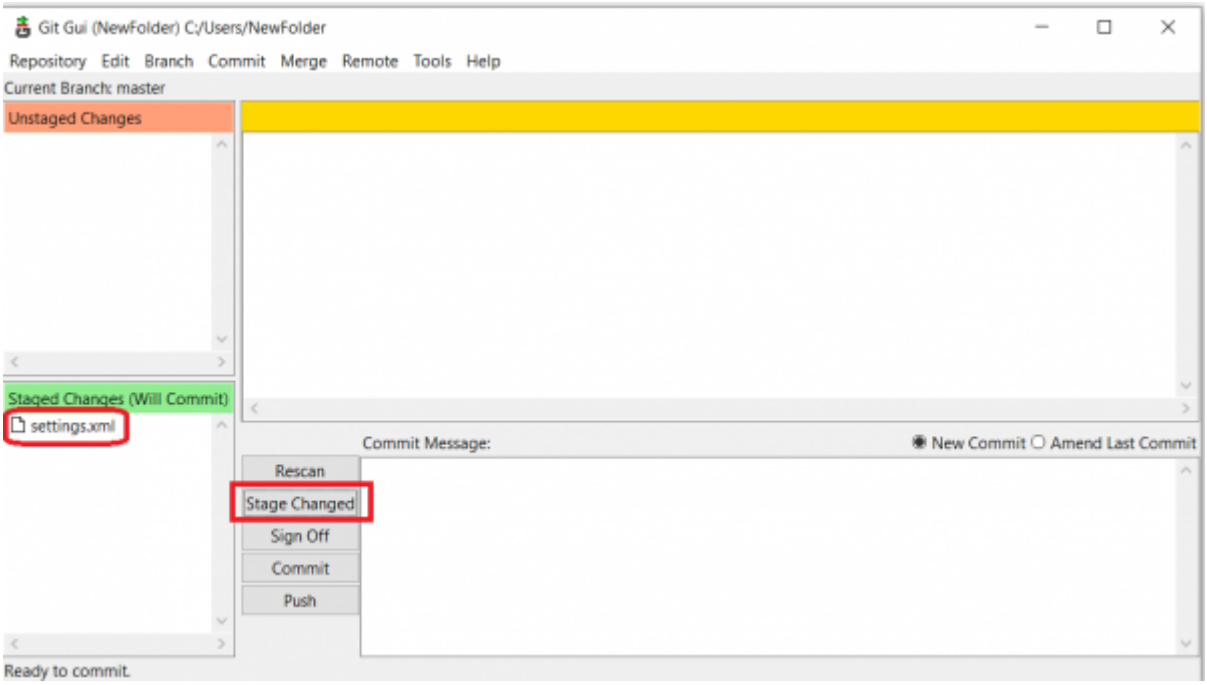


E' possibile vedere, infatti, che la finestra è divisa in quattro diverse sezioni:

- Unstaged Changes, in alto a sinistra: contiene i file aggiunti o modificati che non sono stati salvati sul repository remoto;
- Staged Changes, in basso a sinistra: contiene i file che si vogliono andare a salvare sul repository remoto (inizialmente è vuoto);
- una riquadro in alto a destra dove, una volta selezionato un file, si possono visualizzare le modifiche apportate ad esso;
- una sezione in basso a destra con le varie opzioni per l'operazione di *commit*.

Per salvare le modifiche in remoto, dunque, basta seguire questi passaggi:

1. Se si vogliono salvare tutti i file che sono stati modificati o aggiunti, si deve cliccare il pulsante **Stage changed**, evidenziato nella figura che segue:



In questo modo tutti i file nell'elenco in alto *unstaged files* verranno automaticamente spostati nel riquadro sottostante (**Staged changes**). Se, invece, si vogliono riportare in remoto solo alcuni dei file che sono stati modificati, è possibile selezionarne uno alla volta e poi, dalla toolbar, andare su **Commit → Stage to commit**; così facendo, solo il file selezionato verrà spostato nel riquadro **Staged changes**.

2. Aggiungere nella sezione **Commit message**, in basso a destra, un messaggio testuale per descrivere brevemente le modifiche apportate, in modo che, in seguito, sarà più facile orientarsi fra la successione di versioni del progetto.
3. Cliccare il pulsante **Commit**: tutte le modifiche verranno salvate nel repository locale di Git.
4. Cliccare il pulsante **Push** e poi cliccarlo di nuovo nella finestra che si apre: in questo modo, le modifiche verranno riportate anche nel repository Git remoto.

Casi tipici di utilizzo

Nonostante l'estrema versatilità di uno strumento come Git, è possibile che alcune procedure o alcuni comandi descritti nei paragrafi precedenti siano, in qualche caso, ridondanti o non utili. Per semplificare le operazioni, quindi, qui di seguito sono elencati alcuni casi standard in cui si deve utilizzare Git e i quali necessitano solo di alcuni comandi.

- Il repository di commessa non è condiviso: questo significa che si è i soli a lavorare con esso e, perciò, non è necessario creare ramificazioni complesse. Dopo aver clonato una sola volta il repository nella propria macchina, basterà riportare in remoto, di volta in volta, le modifiche effettuate. Ciò vuol dire che le uniche operazioni che devono essere eseguite tramite Git sono quelle di *commit* e *push*; di conseguenza, gli unici comandi che devono essere usati sono:

```
git add all
```

```
git commit -m "Inserire il messaggio di descrizione della modifica"
```

```
git push
```

- Il repository di commessa è condiviso: più persone devono lavorare allo stesso repository, anche tramite macchine diverse. In questo caso è bene mantenere sempre sia il repository remoto sia tutti i repository locali allineati alle varie modifiche per evitare errori e conflitti. Questo significa che, prima di apportare cambiamenti, si deve controllare che il proprio repository sia aggiornato con quello remoto usando il comando “git pull” (ed eventualmente “git merge”). Solo dopo essersi assicurati che tutto sia correttamente allineato, è possibile apportare le proprie modifiche e poi eseguire l'operazione di *commit* e *push*. Quindi, in questo caso, il comando da utilizzare prima della modifica è:

```
git pull
```

Mentre dopo la modifica:

```
git add all
```

```
git commit -m "Inserire il messaggio di descrizione della modifica"
```

```
git push
```