DA 0 A GIT

Guida tascabile a GIT
Parte 1

R.dev_

Guida tascabile a git

Da zero a git - parte 1

Indice

Autore

Giovanni Riefolo è Front-end Developer e WordPress Specialist. Nel 2016 è co-fondatore di Themecraft Studio, e supporta Aziende e Freelance nello sviluppo di progetti digitali.

Collegati al profilo LinkedIn o segui su Twitter.

Introduzione

Il settore dello sviluppo web è in continua evoluzione e sempre più spesso viene richiesto di aggiornarsi nell'utilizzo di tecnologie necessarie per crescere e dare il meglio di sé nei propri progetti. Themecraft Studio ha progettato una serie di guide con l'intento di aiutare a comprendere velocemente e in termini chiari alcune di queste tecnologie.

Se stai leggendo questa guida, molto probabilmente è perchè hai sentito parlare di Git e vuoi capire cos'è e come puoi utilizzarlo per dare il meglio nei tuoi progetti. Al contrario, se utilizzi già Git per i tuoi progetti, questa guida può tornarti utile con i suoi riferimenti rapidi a comandi e funzioni da utilizzare ogni giorno o in momenti specifici.

Da zero a git è una guida composta in più parti in base al tuo livello di conoscenza, così da coprire quanto più possibile le tue necessità partendo da un uso generico fino a coprire casi d'uso specifici.

Cos'è Git e come funziona.

Git fa parte della famiglia dei sistemi di controllo versione (*VCS - version control system*) ed è utilizzabile sia da riga di comando che attraverso una GUI.

È nato nel 2005 per mano di Linus Torvalds, uno degli sviluppatori principali del kernel Linux. Lo scopo era di supportare lo sviluppo del kernel Linux dopo che il predecessore, BitKeeper, era passato da distribuire licenze gratuite a essere disponibile solo con licenze a pagamento.
Arrivando ai giorni d'oggi, Git ha conosciuto un rapido successo dovuto alla sua velocità, scalabilità ed al supporto a flussi di lavoro non lineari, ed è utilizzato per tenere traccia delle modifiche di qualsiasi file in ambito di sviluppo software.

Essendo un *sistema di* controllo versione distribuito (*DVSC - Distributed version control system*), permette di poter lavorare sul codice sorgente senza la necessità di un server centralizzato, permettendo agli sviluppatori di scaricare una copia integrale del progetto per poter lavorare e condividere le modifiche anche su diverse versioni (o meglio *rami*) di quest'ultimo.

Per comprendere meglio come Git può essere utile nello sviluppo di un progetto, prendiamo un esempio tratto dalla vita reale.

Due programmatori stanno lavorando al progetto di uno script.

Ognuno dei due potrà scaricare sul suo terminale di sviluppo una copia dei file del progetto (il *repository*, o remoto) e apportare le modifiche desiderate.

Uno dei due programmatori apporta delle modifiche che vengono salvate e registrate (sul file *head* ed attraverso un *commit*) per essere inviate, quando è pronto, al repository principale (tramite *push*).

L'altro programmatore scarica la versione aggiornata dello script (tramite *pull*), esegue un commit delle sue modifiche ed un push al repository principale.

Lo script è completo e dal repository principale potranno essere recuperati i file per utilizzare lo script.

Le possibilità che offre Git sono veramente tantissime, possiamo ad esempio includere un repository all'interno di un altro, oppure tornare indietro nel tempo in qualsiasi momento ad una versione diversa del nostro progetto o addirittura creare diverse versioni del nostro repository per lavorare su una nuova caratteristica sperimentale e tenere traccia delle modifiche senza intaccare i file originali.

Installazione

Partiamo dal passo fondamentale, il download e l'installazione. Possiamo scaricarlo per utilizzarlo in riga di comando o affidarci ad un client. Nel caso in cui tu preferisca quest'ultimo ti consiglio di leggere e conservare ugualmente i comandi che troverai più avanti. Talvolta l'uso di una shell rende tutto più efficace e veloce, e soprattutto ci permette di capire meglio cosa succede!

Download per MacOS

[shell] https://git-scm.com/download/mac
[GUI] https://git-scm.com/download/gui/mac

Download per Windows

[shell] https://git-scm.com/download/win
[GUI] https://git-scm.com/download/gui/windows

Download per Linux

[shell] https://git-scm.com/download/linux
[GUI] https://git-scm.com/download/gui/linux

Terminologia

Quando si inizia a conoscere una tecnologia nuova è importante sapere quali sono i termini da utilizzare per farci riferimento. Eccone alcuni, di seguito, tra quelli che sentirai sempre più frequentemente utilizzando Git per i tuoi progetti.

Repository

Il repository altro non è che la copia sincronizzabile della directory principale del progetto.
Servizi come BitBucket o Github mettono a disposizione un hosting per i repository, permettendo di controllare gli accessi e agevolando il lavoro in team. Più avanti in questa guida ti spiegherò come utilizzare i servizi che ti ho citato per la gestione dei tuoi progetti.

Commit

Il `commit` (tradotto letteralmente "consegnare") è l'elenco delle modifiche effettuate ad un progetto, validate e "affidate" a git per essere inviate al repository principale. Git assegna un "nome", ossia una stringa di numeri e lettere, ad ogni singolo commit in modo che sia tracciabile. Questo commit viene registrato in un file di `head` (vedi più avanti).

Push e Pull

Il `push` ed il `pull` (tradotti letteralmente "spingere" e "tirare") sono due comandi di git che servono per inviare e ricevere gli aggiornamenti. Il `push` invierà i tuoi commit al repository principale, il `pull` chiederà a quest'ultimo di poter scaricare le modifiche sul tuo repository locale e unirle ai tuoi file. Il pull è altresì l'unione del comando `fetch` e del comando `merge`.

Fetch

Il `fetch` (tradotto letteralmente "raggiungere, andare a prendere") è il comando che dice a Git di recuperare il contenuto di un repository. Generalmente viene utilizzato il comando `pull`, che combina appunto il `fetch` ("andare a prendere") e il comando `merge` ("unione").

Merge

Il merge (tradotto letteralmente "fusione, unione") è l'unione tra commit di branch differenti.

Remote

I remoti non sono altro che versioni dei repository ospitati in locale oppure online. Lo scopo nell'uso di più remoti è poter collaborare sullo sviluppo del progetto, tenendo però copie separate dei repository. Il remoto di default su Git è indicato con il nome `origin`, ma è possibile crearne di nuovi.
Per vedere i remoti associati al progetto su cui si sta lavorando tramite il comando `git remote`. Con l'opzione `-v` si avranno informazioni complete, potendo vedere da quale remoto si ricevono i commits (`fetch`) e verso quale remoto si inviano (`push`)

Branch

Il branch (tradotto letteralmente “ramo”) in generale indica una versione del tuo repository contenente uno o più commit non presenti nella versione “originale”.
In Git, infatti, abbiamo un “ramo” principale chiamato `master` che raccoglie tutti commit dei file del tuo progetto. Può succedere tuttavia che sia necessario creare un ramo nuovo per testare delle versioni diverse di uno o più file del progetto. In questo caso creeremo un branch, (assegnandogli anche un nome identificativo) che conterrà tutti i commit delle modifiche ai file. Potrai infine scegliere se riunire i rami tramite un merge o abbandonare il ramo creato per tornare al branch originale (oppure a uno nuovo!).

Checkout

Può capitare di dover ripristinare un singolo file o spostarsi su diversi commit su uno o più branch. Per queste operazioni possiamo fare affidamento al comando `git checkout`. Come vedrai in seguito, possiamo usare il comando di checkout per spostarci su un branch diverso da *master* o ripristinare l’intero progetto ad una versione precedente indicandogli l’hash (il “nome”) del commit al quale desideri tornare.

HEAD

Con HEAD si indica il branch di lavoro attuale. Più precisamente, `HEAD` è la referenza all’ultimo commit di un ramo di lavoro di un branch al quale si sta lavorando. Possono esistere più heads ognuno dei quali farà riferimento all’ultimo commit di un branch (visibili in `.git/ref/heads`). È possibile inserire uno specifico commit nella lista degli heads, in quel caso si parla di `HEAD` distaccato (non lo affronteremo in questa guida).

Tag

Può una guida di un sistema di controllo versione non parlare del comando per assegnare il “nome” di versione? Ovviamente no!
Questa funzionalità viene utilizzata tipicamente per contrassegnare un punto, nella storia del progetto, di particolare importanza (Es. la versione 1.0.2 di un patch). Git supporta due tipi di tag:

- **Lightweight tag** - molto semplicemente è il riferimento ad uno specifico commit. Verrà preso come riferimento il checksum del commit e nessun’altra informazione tranne le informazioni dell’autore del repository (nome, email e data creazione). Per assegnare il tag basterà utilizzare `git tag` seguito dal tag, non ci sono opzioni utilizzabili.

```
$ git tag v1.0
```

- **Annotated tag** - i tag con annotazione vengono conservati nel database di Git come veri e proprio “oggetti”, con un loro checksum e con le informazioni dell’autore del tag. Possono inoltre essere criptati e verificati con GnuPG (GPG). Per creare un tag con annotazione bisogna utilizzare il comando git tag seguito dall’opzione -a per assegnare la versione, -m per includere un messaggio.

```
$ git tag -a v1.0 -m “versione iniziale”
```

Conflitto

Arrivati qui sento che è doveroso aprire un piccolo paragrafo con lo scopo di rassicurarti quando ti ritroverai davanti al tuo primo messaggio d'errore. Quando ti troverai davanti ad un conflitto ricorda: il tuo progetto non è distrutto per sempre!

La maggior parte delle volte Git si occupa di risolvere i conflitti senza che nemmeno ce ne rendiamo conto. Quando, ad esempio, devi eseguire un pull dal repository principale, Git scarica gli ultimi commit e si occupa di integrare le modifiche ai file della tua copia di lavoro.
Tuttavia non sempre le cose filano lisce e può succedere che il merge fallisca, richiedendo all'utente l'intervento manuale. Niente paura! Tutto ciò che serve è armarsi di un po' di pazienza, vedere quali sono i file per i quali git non è riuscito a fare automaticamente il merge e applicare manualmente le modifiche proveniente dal branch dal quale stiamo facendo il merge.
All'interno dei file in conflitto, verranno evidenziate da Git stesso quali sono le porzioni di codice contenute in `HEAD` che creano conflitto e qual è il contenuto dell'ultimo commit (identificato dal checksum) che non riesce ad integrare.

Il file presenterà, ad un certo punto, del testo come in questo esempio:

```
<<<<<<< HEAD
<span> codice contenuto in locale </span>
=======
<span> codice del commit 6ef9c56 </span>
>>>>>>> 6ef9c569a908bbd3a2fd296a02ffe5bb41a41bd4
```

> `<<<<<<<`, `=======` e `>>>>>>>` sono dei marker e indicano la versione di codice contenuta in `HEAD` e la versione contenuto nel branch da cui si vuole fare il merge, separati da `=======`

Tutto quello che devi fare è modificare il file scegliendo quale delle due porzioni di codice tenere, cancellando quella che non è più necessaria. Dopodichè potrai eseguire un `commit` con i file modificati e inviarli al repository.

Problema risolto!

Per il momento è tutto! Questi sono i termini fondamentali che ti permetteranno di lavorare fin da subito con Git, organizzando al meglio il tuo lavoro e velocizzando la gestione in team dei tuoi progetti . Nella prossima guida, invece, vedremo da vicino termini e comandi per un utilizzo più complesso di questo sistema.

Non resta che scoprire come utilizzare quanto appreso fino ad ora, quindi cominciamo subito!

Hosting dei repository

Prima di continuare con degli esercizi pratici sull'uso di Git, vorrei che ci fermassimo un momento per dare uno sguardo ai servizi di *hosting* dei repository.

In precedenza, spiegando cos'è Git, ho specificato come questo non richieda un server centralizzato. Inizialmente, infatti, venivano utilizzati sistemi di controllo "centralizzati" (vedi Subversion, CVS o Perforce), dove esisteva una singola copia centralizzata del progetto e tutti gli sviluppatori inviavano i propri commit. Gli sviluppatori potevano vedere le modifiche, scaricare il file aggiornato e integrarlo nel progetto.

I problemi, come potrai immaginare, erano parecchi:

- Due o più sviluppatori potevano avere necessità di integrare modifiche allo stesso file nello stesso giorno, con evidenti problemi di integrazioni;
- Anche se la modifica era di una singola riga, era necessario salvare una copia intera del file;
- Le copie di sviluppo erano salvate unicamente sui terminali degli sviluppatori (!);
- Per ogni copia potevano esserci versioni diverse dei file, rendendo più complesso lo schema del progetto e appesantendo in maniera esagerata la directory principale;
- Per chiedere un confronto di due versioni di uno stesso file ad uno sviluppatore era necessario mandargli entrambe separatamente.

Espressione tipica di uno sviluppatore che utilizza un sistema di controllo centralizzato

Per fortuna i sistemi di controllo versione si sono evoluti. La gestione si è semplificata, per esempio permettendo di creare sul tuo computer un repository e di caricarlo su un server per la collaborazione. Ogni collaboratore può scaricare la versione completa di tutto il progetto e i file vengono aggiornati simultaneamente con le modifiche di ognuno.

Se l'intento è di lavorare in team è dunque necessario uno spazio d'appoggio per la condivisione e la conservazione dei file (e delle informazioni relative). A questo proposito ti suggerisco quindi due tra i più diffusi e utilizzati servizi di hosting.

- Github - il più grande servizio di hosting gratuito per gli sviluppatori software, come si intuisce dal nome nasce per interfacciarsi tramite Git.

- BitBucket - servizio di hosting gratuito che fa parte della famiglia dei servizi Atlassian, permette di interfacciarsi tramite Git o Mercurial.

Entrambi sono servizi molto validi e con molte funzionalità utili, BitBucket differisce dal primo principalmente perché permette la creazione di illimitati repository privati, a differenza di Github con il quale sono a pagamento.

Una volta che hai deciso di creare il tuo account con uno di questi servizi, ti consiglio di seguire le guide offerte dai servizi per configurare il collegamento tramite SSH, così avrai un accesso sicuro e pratico al repository.

Esercizi pratici guidati

Dopo aver visto cos'è Git, da dove installarlo, quali sono i termini a cui far riferimento e dove conservare il tuo codice, è arrivato il momento di testare nella pratica come funziona. Non mi limiterò a darti delle indicazioni sommarie, ma entreremo nel dettaglio vedendo come lavorare con i servizi di hosting citati nel capitolo precedente.

Di seguito un paio di esercizi guidati che ti mostreranno come:

1. **Utilizzare repository online.** Vedremo come creare un repository su Github o su BitBucket, scaricare una copia sul tuo terminale, aggiornare il file `README.md` e inviare le modifiche effettuate.

2. **Utilizzare un repository in locale.** Vedremo come creare un repository sul tuo terminale e inviarlo su Github o BitBucket.

NB. Per svolgere questi esercizi ti servirà il terminale.

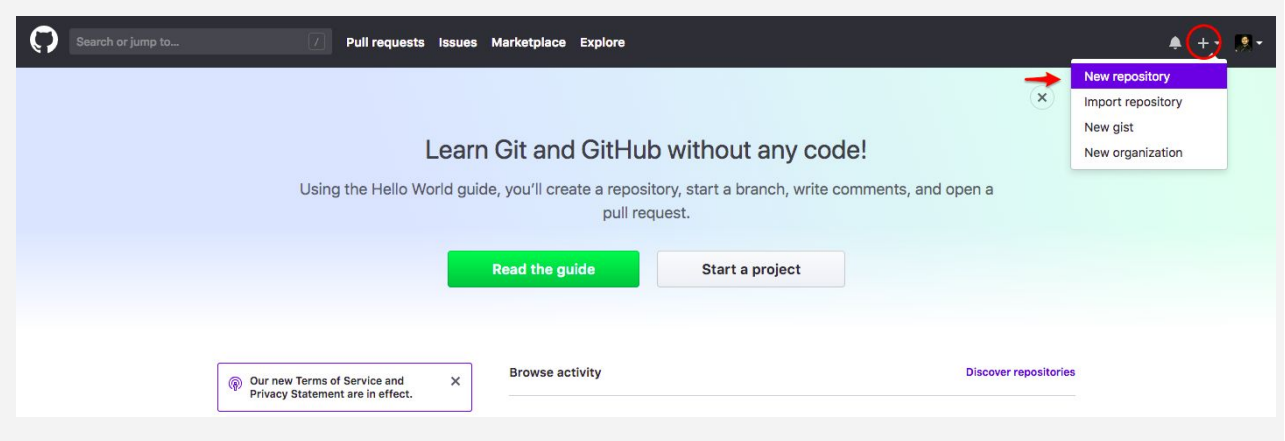
Contenuto degli esercizi

Es. 1 - Utilizzo dei repository online.

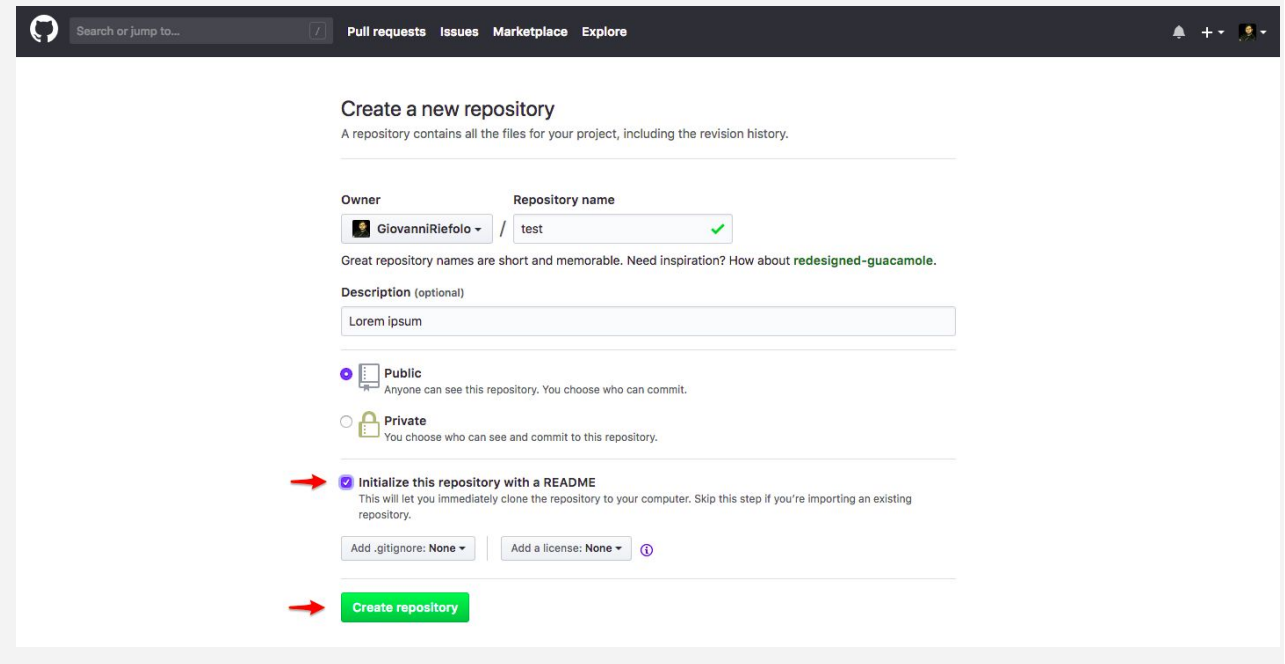
Iniziamo subito con qualche screenshot che può aiutare a capire come creare un repository su Github o su BitBucket. Puoi scegliere uno qualsiasi di questi servizi, questo esercizio guidato prevede entrambe le possibilità.

— Creare un repository su Github

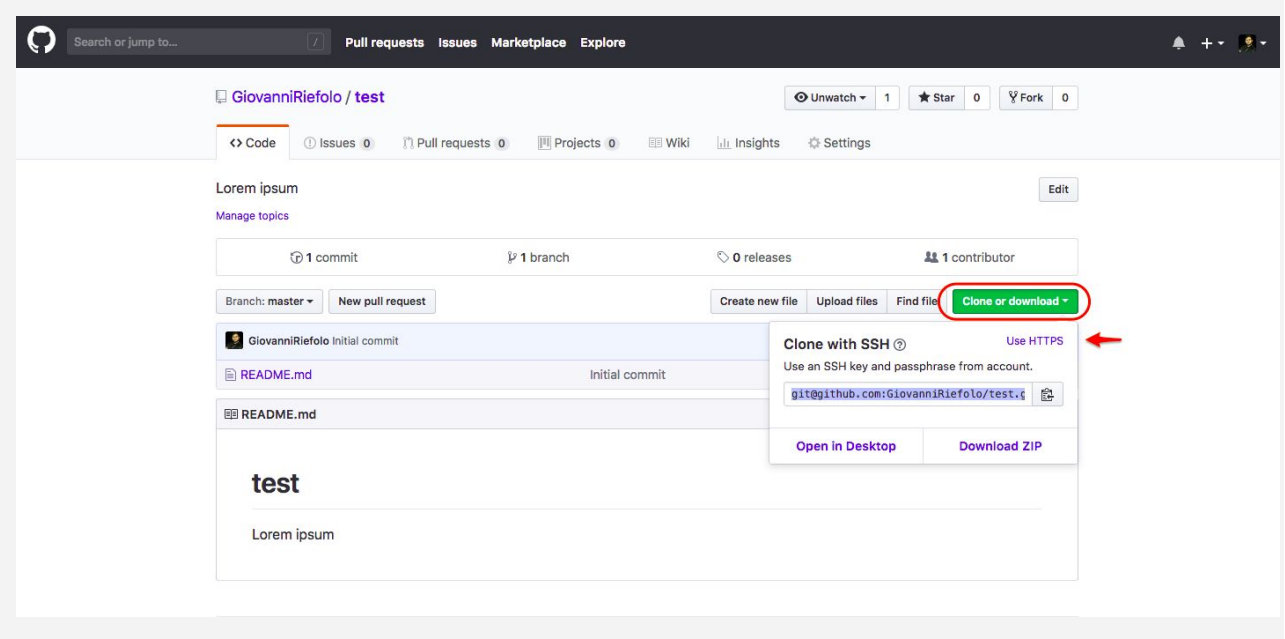
`step 1 -` Clicca su “+” nella barra in alto e seleziona “New repository”.



`step 2 -` Inserisci il nome del tuo repository, seleziona pubblico o privato (in questo caso serve l’account a pagamento), e spunta “initialize this repository with a README” così avremo il nostro primo file da scaricare e modificare!

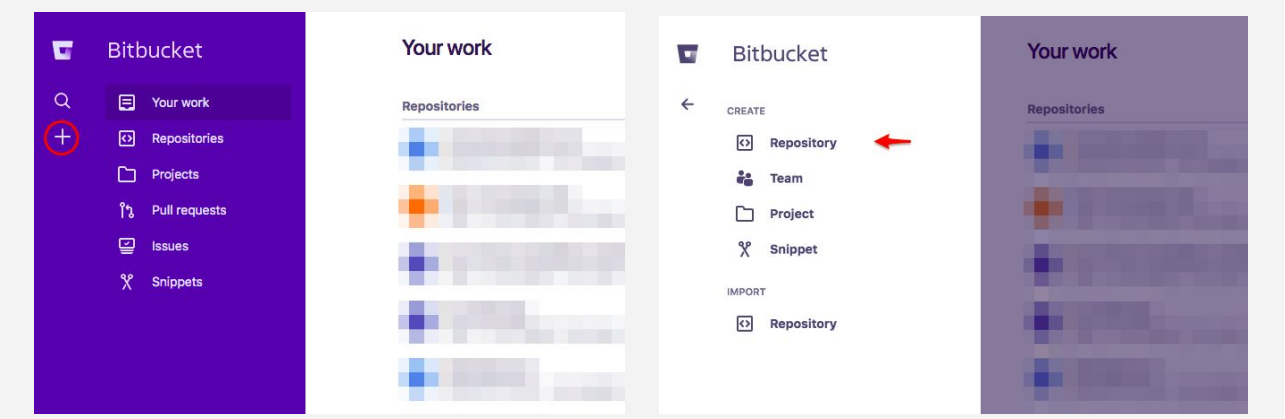


`step 3` - Seleziona “clone or download” per vedere l’URL del repository. Seleziona “Use HTTPS” se non hai configurato l’accesso SSH.



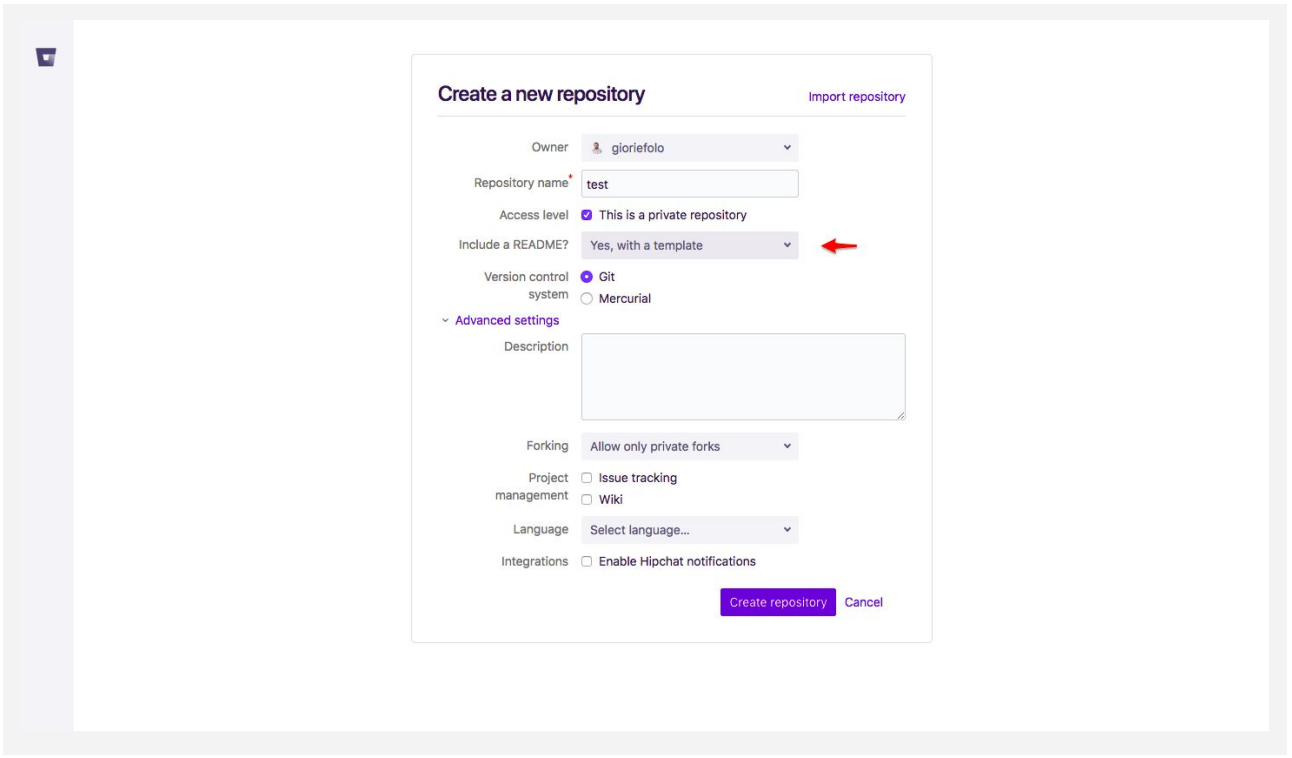
— Creare un repository su BitBucket

`step 1` - Fare clic su “+” nella barra a lato e selezionare “Repository”

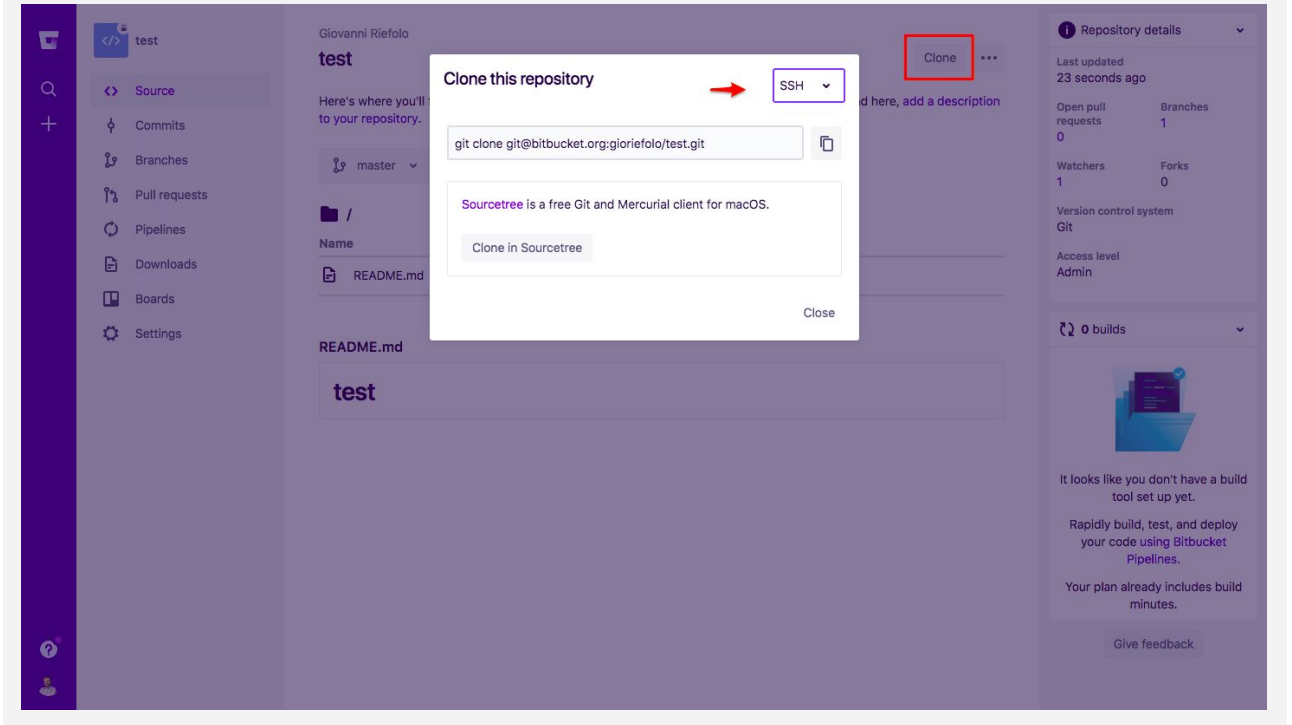


`step 2` - Inserisci il nome del repository e alla voce “include README?” selezionare “Yes with a template” così avremo il nostro primo file da scaricare e modificare.

NB. Dallo screenshot puoi vedere che premendo il tasto “Advanced Settings” ci sono delle informazioni aggiuntive che puoi assegnare al repository e abilitare l’issue tracking e la Wiki dove puoi scrivere delle informazioni sul tuo progetto.



`step 3` - Seleziona “Clone” per visualizzare l’URL del repository. Se non hai configurato BitBucket per la connessione tramite SSH, puoi scegliere la connessione HTTPS.



— Modifica del file README e push su repository

Adesso che abbiamo visto come creare un repository su Github o BitBucket è ora di metter mano al terminale! Riprendiamo dallo step tre di ognuno dei punti precedenti e proseguiamo.

`step 4 -` Che tu abbia macOS o Windows, poco cambia. Crea una cartella dove più preferisci, chiamala `guida-git` e accedi da terminale.

```
$ mkdir guida-git
$ cd guida-git
```

`step 5 -` Dentro la cartella `guida-git` creiamo la cartella `esercizio-1` dove, senza accedervi, scaricheremo il repository

```
$ mkdir esercizio-1
```

`step 6 -` Adesso eseguiamo il comando `git clone` seguito dall'URL del repository (vedi gli step 3 della parte relativa a Github o a BitBucket) e dal nome della directory creata prima

```
//Github
$ git clone git@github.com:GiovanniRiefolo/test.git esercizio-1
```

oppure

```
//Bitbucket
$ git clone git@bitbucket.org:gioriefolo/test.git eserizio-1
```

NB. Come puoi vedere il comando si compone di queste tre parti:

```
git clone
+ url repository
+ directory dove copiare il repository (all'interno di quella attuale)
```

`step 7 -` una volta terminato accediamo alla cartella `esercizio-1`, dentro potremo trovarci il file `README.md`. Apriamo con qualsiasi editor e sostituiamo il contenuto di testo con altro a nostro piacimento. Salviamo le modifiche al file e torniamo sul terminale.

Se vuoi provare un editor da terminale usa `nano` o `vim` per sistemi Linux e MacOs, su Windows (a meno che tu non abbia installato Bash for Windows) l'unico editor è Notepad, ma puoi avviarlo da powershell col comando `notepad`.

`step 8 -` In qualsiasi momento possiamo interrogare git chiedendogli qual è lo stato del progetto. Per farlo basta inserire il seguente comando:

```
$ git status
```

il risultato, nel nostro caso dovrebbe essere questo:

```
$ git status
$ > changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working
directory)

      modified:   README.md
```

Git ci sta informando che ci sono delle modifiche non committate nel file `README.md`. L'output restituisce l'elenco dei file modificati e, a seconda dello stato, evidenzia di verde quelli commitati e di rosso quelli non commitati. Se non ci sono modifiche l'output sarà questo:

```
$ git status
$ > On branch master
    Your branch is up to date with 'origin/master'.

      nothing to commit, working tree clean
```

`step 9 -` Eseguiamo un commit delle modifiche fatte:

```
$ git add --all
```

per aggiungere all'indice le modifiche effettuate

```
$ git commit -m “modificato titolo README”
```

per creare il commit che include le modifiche eseguite al file più un messaggio di descrizione

`step 10 -` inviamo le modifiche al repository remoto

```
$ git push
```

opzionalmente possiamo specificare sia il remoto che il branch, nel caso in cui ci siamo più remoti a cui inviare le modifiche e/o si stia lavorando su una versione alternativa. Di default il remoto è `origin` e il branch è `master`, quindi il comando completo sarà:

```
$ git push origin master
```

Congratulazioni! Se hai seguito tutti gli step senza riscontrare problemi vuol dire che hai appena imparato ad usare Git autonomamente per i tuoi progetti.

Es. 2 - Utilizzare un repository in locale.

In questo esercizio faremo l'inverso del primo. Partiamo quindi creando un repository locale, aggiungeremo dei file e in seguito ci collegheremo ad un repository esterno per inviare i nostri commit.

N.B. Una volta installato, git è disponibile sul nostro computer in qualsiasi posizione.

— Creare un repository locale

`step 1 -` Apriamo il terminale, accediamo alla cartella guida-git. Se non l'abbiamo ancora creata nell'esercizio precedente, utilizziamo il comando mkdir (sia su Linux/MacOS che su Windows).

```
$ cd guida-git
```

`step 2 -` Creiamo la cartella esercizio-2 e accediamo all'interno

```
$ mkdir esercizio-2
$ cd esercizio-2
```

`step 3 -` Adesso inizializziamo un repository nuovo che si troverà dentro la cartella `esercizio-2`.

```
$ git init
```

Una volta lanciato il comando, git crea un nuovo repository nella directory caricando il file `HEAD`, il file di configurazione e altri componenti che gli serviranno per il corretto funzionamento.

Possiamo verificare che git abbia caricato correttamente gli *asset* necessari guardando le cartelle nascoste. Eseguendo il comando comando `ls -a` dovremmo trovare questo

```
$ .    ..    .git
```

`.git` è la cartella nascosta che git ha appena creato. Se accediamo al suo interno troveremo:

```
$ .    HEAD    description info    refs
  ..   config  hooks       objects
```

quelle in grassetto sono ulteriori sottocartelle contenenti altri *asset* di git.

`step 3 -` Creiamo un file `README.md` all'interno della cartella `esercizio-2` e modifichiamo il contenuto. Se vuoi provare un editor da terminale usa `nano` o `vim` per sistemi Linux e MacOs, su Windows (a meno che tu non abbia installato Bash for Windows) l'unico editor è Notepad, ma puoi avviarlo da powershell col comando `notepad`.

`step 4 -` Se interroghiamo git sullo stato del repository dopo aver creato e salvato il nuovo file README.md questo sarà il risultato:

```
$ git status
On branch master

No commits yet

Untracked files:
      (use "git add <file>..." to include in what will be
      committed)

      README.md

nothing added to commit but untracked files present (use "git add"
to track)
```

Git ci sta informando che non sono ancora stati creati dei commits e che il file README.md è nuovo e non è ancora stato incluso in nessun commit. Creiamo dunque il nostro primo commit contenente il file REAMDE.md con le relative modifiche.

```
$ git add --all
$ git commit -m “creato file README”
```

Questa volta, se interrogato, git ci risponderà diversamente:

```
$ git status
On branch master
Nothing to commit, working tree clean
```

Tutto a posto, il file è stato aggiunto al commit e siamo pronti per il prossimo passaggio!

`step 5 -` Adesso dobbiamo caricare il nostro repository online. Per iniziare va’ creato un repository online a cui collegare il nostro locale. Possiamo scegliere tra vari servizi, in questa guida trattiamo Bitbucket e Github. Se non hai ancora letto l’esercizio 1, ti consiglio di dare uno sguardo alla parte guidata per la creazione di un repository su uno di questi due servizi.

— Creare un repository su Github
— Creare un repository su Bitbucket

Una volta creato un nuovo repository online andiamo a collegarlo a quello in locale. Per fare questo associeremo un *remoto* al nostro repository. I remoti sono i server collegati al tuo repository identificati da un nome. Quello di default è **origin**. Se interroghiamo git su quale remoto è attualmente collegato non risponderà in quanto l’elenco dei remoti è vuoto

```
$ git remote -v
$
```

Adesso colleghiamo il repository sul server (Github o Bitbucket) e diamogli il nome “origin”.

```
$ git remote add origin git@github.com:GiovanniRiefolo/test.git
```

Se interrogato adesso git risponderà

```
$ git remote -v
origin    git@github.com:GiovanniRiefolo/test.git  (fetch)
origin    git@github.com:GiovanniRiefolo/test.git  (push)
```

La risposta ci dice che esiste un remoto identificato con il nome origin disponibile all’URL git@github.com:GiovanniRiefolo/test.git. La riga è ripetuta perché indica sia il server per il `fetch` (richiesta) che per il **push** (invio). Se aggiungiamo più remoti git li elencherà tutti. Non è possibile avere remoti con nomi uguali.

`step 6 -` Ultimo step! Adesso che abbiamo collegato il repository sul server inviamo i commit:

```
$ git push origin master
```

Questo comando invia i file al remoto e da’ qualche informazione in più:

- specifica il nome del remoto a cui inviare i commit
- specifica il nome del branch su cui i commit vanno inviati

Congratulazioni! Se hai seguito tutti gli step senza riscontrare problemi vuol dire che hai appena imparato ad usare Git autonomamente per i tuoi progetti.

Se hai completato tutti gli esercizi vuol dire che le basi di git non hanno più segreti per te e sei pronto per affrontare sfide ancora più grandi!

Il prossimo capitolo è composto principalmente dall’elenco di comandi utili che abbiamo visto nella guida e negli esercizi, seguiti dalla descrizione veloce. In questo modo potrai avere sempre sottomano un riferimento per qualsiasi evenienza!

Happy coding!

Comandi ed opzioni

Di seguito ti mostrerò i comandi più utilizzati con qualche opzione che può tornarti utile in qualche caso.

`git init`	Questo comando crea un repository di Git vuoto.
`git status`	Questo comando mostra lo stato del repository indicando i file modificati e se sono stati aggiunti in `index` oppure no.
`git add <filename>`	Questo comando aggiunge in `index` il file indicato.
`git add --all`	Questo comando aggiunge all'`index` tutti i file modificati.
`git commit -m "messaggio"`	Questo comando crea un commit con i file aggiunti all'`index`, l'opzione permette di aggiungere un messaggio al commit. Le modifiche si troveranno quindi nell'`HEAD`.
`git commit -am "messaggio"`	Questo comando riassume `git add --all` e `git commit -m`. **Attenzione:** `git commit -am` non include i nuovi file creati, ma solo le modifiche apportati ai file già presenti.
`git push`	Questo comando permette di inviare al repository tutti i commit contenuti nell'`HEAD` locale.
`git push <remote> <branch>`	Questo comando permette di inviare al repository indicato in <remote> tutti i commit contenuti nell'`HEAD` per il branch <branch>. Generalmente il remoto principale viene chiamato `origin` e il branch iniziale `master`, quindi il comando esteso sarà `git push origin master`.
`git pull`	Questo comando permette di scaricare tutti i file dal server collegato, provando a unirli con i file del repository su cui si sta lavorando.
`git pull <remote> <branch>`	Questo comando permette di permette di scaricare tutti i file dal server remoto collegato `<remote>` sul branch indicato `<branch>` all'interno del repository su cui si sta lavorando. Generalmente il remoto principale viene chiamato `origin` e il branch iniziale `master`, quindi il comando esteso sarà `git pull origin master`
`git remote add <nome> <URL>`	Questo comando permette di associare un server remoto al repository, definendo il nome (origin è quello di default) e l'URL.
`git remote remove <nome>`	Questo comando rimuove uno specifico remoto dall'elenco di quelli associati.
`git remote -v`	Questo comando mostra l'elenco dei remoti associati al repository
`git remote show <nome>`	Questo comando mostra maggiori informazioni su uno specifico remoto quali URL, branch collegati, configurazione push/pull
`git tag v1.0`	Questo comando permette di creare un tag di versione *lightweight* associato ad un commit.

`git tag -a v1.0 -m “versione iniziale”`	Questo comando permette di creare un tag di versione *annotated* con un proprio checksum, le informazioni del creatore del tag e il messaggio associato. L’opzione `-a` viene utilizzato per assegnare il nome della versione, l’opzione `-m` viene utilizzato per scrivere il messaggio (se non inserito Git farà apparire un editor di testo per scrivere il messaggio).
`git config --global color.ui auto`	Questo comando abilita globalmente l’uso dei colori di git nella bash. E’ possibile configurare ulteriormente l’uso dei colori editando `~/.gitconfig`.
`git clone git@github.com:User/repository directory/ <directory>`	Questo comando permette di clonare un repository dall’URL selezionata dentro alla directory scelta `<directory>`.

Ringraziamenti

Ettore Del Negro
Full Stack Developer

Documentazione esterna

Documentazione ufficiale di Git (inglese)
https://git-scm.com/doc

Guida alla sintassi per il file README
https://help.github.com/en/articles/basic-writing-and-formatting-syntax