GIT, GitHub e Software Version Control

F. Durastante (fabio.durastante@unipi.it)
S. Steffè (steffe@cs.dm.unipi.it)

1-2 Dicembre

Laboratorio di Introduzione alla Matematica Computazionale – A.A. 2022/2023

Software Version Control

Software Version Control

Nell'**ingegneria del software**, il controllo di versione (in inglese: Software Version Control) è una classe di sistemi responsabili della gestione delle modifiche a programmi per computer, documenti, siti web di grandi dimensioni o altre raccolte di informazioni.

Software Version Control

Nell'**ingegneria del software**, il controllo di versione (in inglese: Software Version Control) è una classe di sistemi responsabili della gestione delle modifiche a programmi per computer, documenti, siti web di grandi dimensioni o altre raccolte di informazioni.

Noi faremo uso di git che è un sistema di controllo della versione distribuito *gratuito* e **open source** progettato per gestire qualsiasi cosa, dai progetti piccoli a quelli molto grandi, con velocità ed efficienza.

Software Version Control

Nell'**ingegneria del software**, il controllo di versione (in inglese: Software Version Control) è una classe di sistemi responsabili della gestione delle modifiche a programmi per computer, documenti, siti web di grandi dimensioni o altre raccolte di informazioni.

Noi faremo uso di git che è un sistema di controllo della versione distribuito *gratuito* e **open source** progettato per gestire qualsiasi cosa, dai progetti piccoli a quelli molto grandi, con velocità ed efficienza.

In una modalità leggermente diversa dalle altre lezioni, questa sarà in forma di *tutorial* in cui cercheremo di fare le differenti operazioni passo-passo.

Git

Server Git

In **primo luogo** abbiamo bisogno di un server che gestisca un servizio **git**.

Nel migliore dei mondi possibili, avremmo un server sotto il nostro controllo assoluto in cui questo servizio è installato.

Quello che faremo invece sarà di usare un servizio terzo, gratuito che ci offra questo servizio:

`https://github.com/`

GitHub, Inc. è un provider di hosting Internet per lo sviluppo di software e il controllo della versione tramite Git. Offre le funzionalità di gestione del codice sorgente di Git, oltre ad alcune altre funzionalità proprietarie. Ha sede in California, ed è una **filiale di Microsoft dal 2018**.

Server Git

In **primo luogo** abbiamo bisogno di un server che gestisca un servizio **git**.

Nel migliore dei mondi possibili, avremmo un server sotto il nostro controllo assoluto in cui questo servizio è installato.

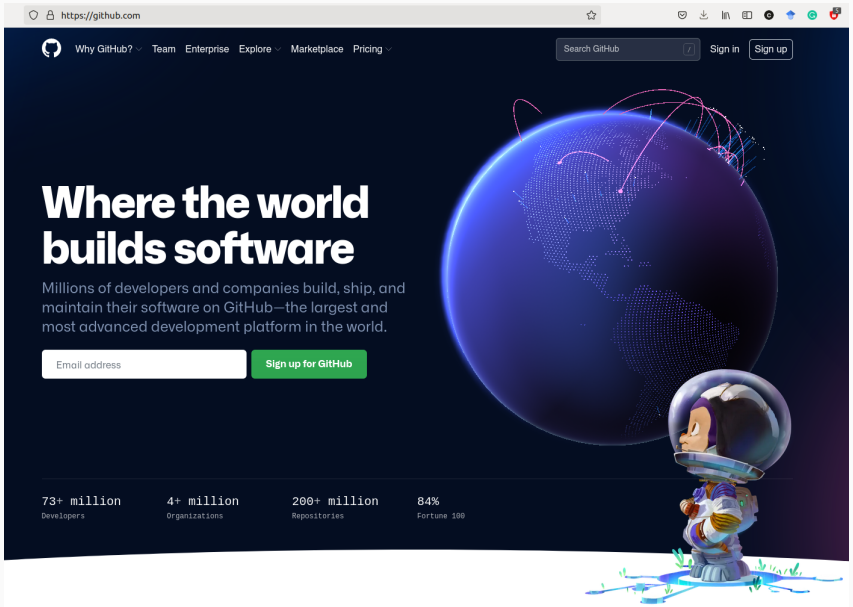
Quello che faremo invece sarà di usare un servizio terzo, gratuito che ci offra questo servizio:

`https://github.com/`

Esistono delle **alternative** come: about.gitlab.com/ o gitea.io che si possono installare anche su un vostro server! O altri prodotti commerciali come Bitbucket: bitbucket.org/.

Registrarsi su GitHub

La **prima operazione** da compiere è quella di registrarsi su GitHub.

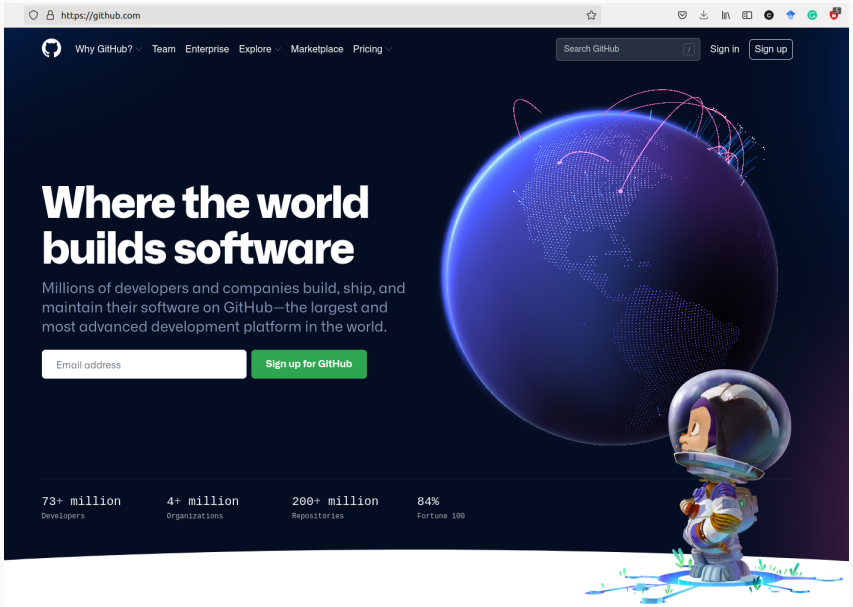


1. Inserite un indirizzo E-Mail (e.g, `n.cognome@unipi.it`) e fate click sul bottone:

   Sign up for GitHub,

Registrarsi su GitHub

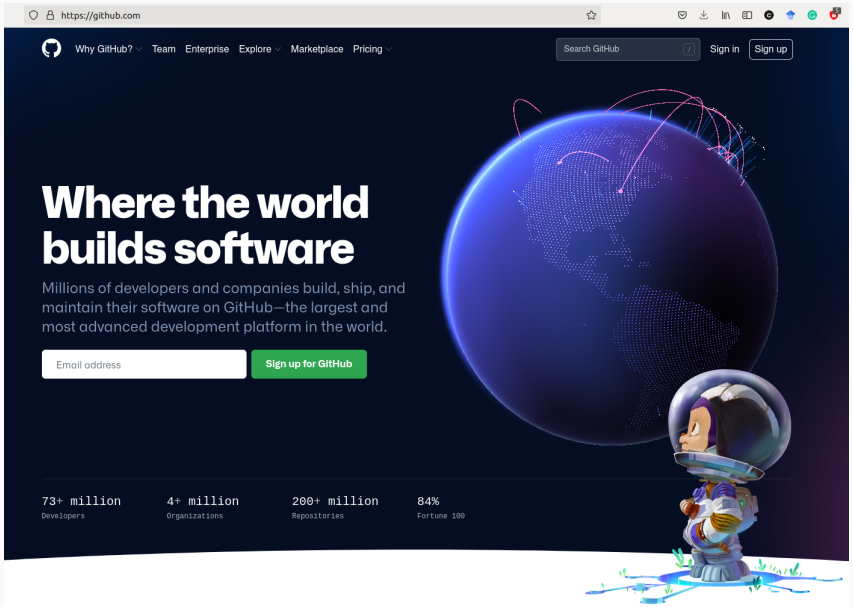
La **prima operazione** da compiere è quella di registrarsi su GitHub.



1. Inserite un indirizzo E-Mail (e.g, `n.cognome@unipi.it`) e fate click sul bottone: **Sign up for GitHub**,
2. Seguite le istruzioni che vi chiederanno una **password** – inseritene una diversa – da quelle delle credenziali di ateneo,

Registrarsi su GitHub

La **prima operazione** da compiere è quella di registrarsi su GitHub.



1. Inserite un indirizzo E-Mail (e.g, `n.cognome@unipi.it`) e fate click sul bottone: Sign up for GitHub,
2. Seguite le istruzioni che vi chiederanno una **password** – inseritene una diversa – da quelle delle credenziali di ateneo,
3. Inserite uno **username**, è importante che sia facile, sarà poi la radice delle vostre **pagine web**. Ad esempio, `fdurastante` per fdurastante.github.io.

Registrarsi su GitHub

La **prima operazione** da compiere è quella di registrarsi su GitHub.

1. Inserite un indirizzo E-Mail (e.g, `n.cognome@unipi.it`) e fate click sul bottone: 

,
2. Seguite le istruzioni che vi chiederanno una **password** – inseritene una diversa – da quelle delle credenziali di ateneo,
3. Inserite uno **username**, è importante che sia facile, sarà poi la radice delle vostre **pagine web**. Ad esempio, `fdurastante` per fdurastante.github.io.
4. Segnate `n` per le mail pubblicitarie,

Registrarsi su GitHub

La **prima operazione** da compiere è quella di registrarsi su GitHub.

1. Inserite un indirizzo E-Mail (e.g, `n.cognome@unipi.it`) e fate click sul bottone: 

,
2. Seguite le istruzioni che vi chiederanno una **password** – inseritene una diversa – da quelle delle credenziali di ateneo,
3. Inserite uno **username**, è importante che sia facile, sarà poi la radice delle vostre **pagine web**. Ad esempio, `fdurastante` per fdurastante.github.io.
4. Segnate `n` per le mail pubblicitarie,
5. Confermate e risolvete il problema per farvi riconoscere come esseri umani.

Registrarsi su GitHub

La **prima operazione** da compiere è quella di registrarsi su GitHub.

1. Inserite un indirizzo E-Mail (e.g, `n.cognome@unipi.it`) e fate click sul bottone: 

,
2. Seguite le istruzioni che vi chiederanno una **password** – inseritene una diversa – da quelle delle credenziali di ateneo,
3. Inserite uno **username**, è importante che sia facile, sarà poi la radice delle vostre **pagine web**. Ad esempio, `fdurastante` per fdurastante.github.io.
4. Segnate `n` per le mail pubblicitarie,
5. Confermate e risolvete il problema per farvi riconoscere come esseri umani.
6. Inserite il codice di verifica che vi è stato mandato via mail e scegliete l'opzione `free` del servizio.

Registrarsi su GitHub

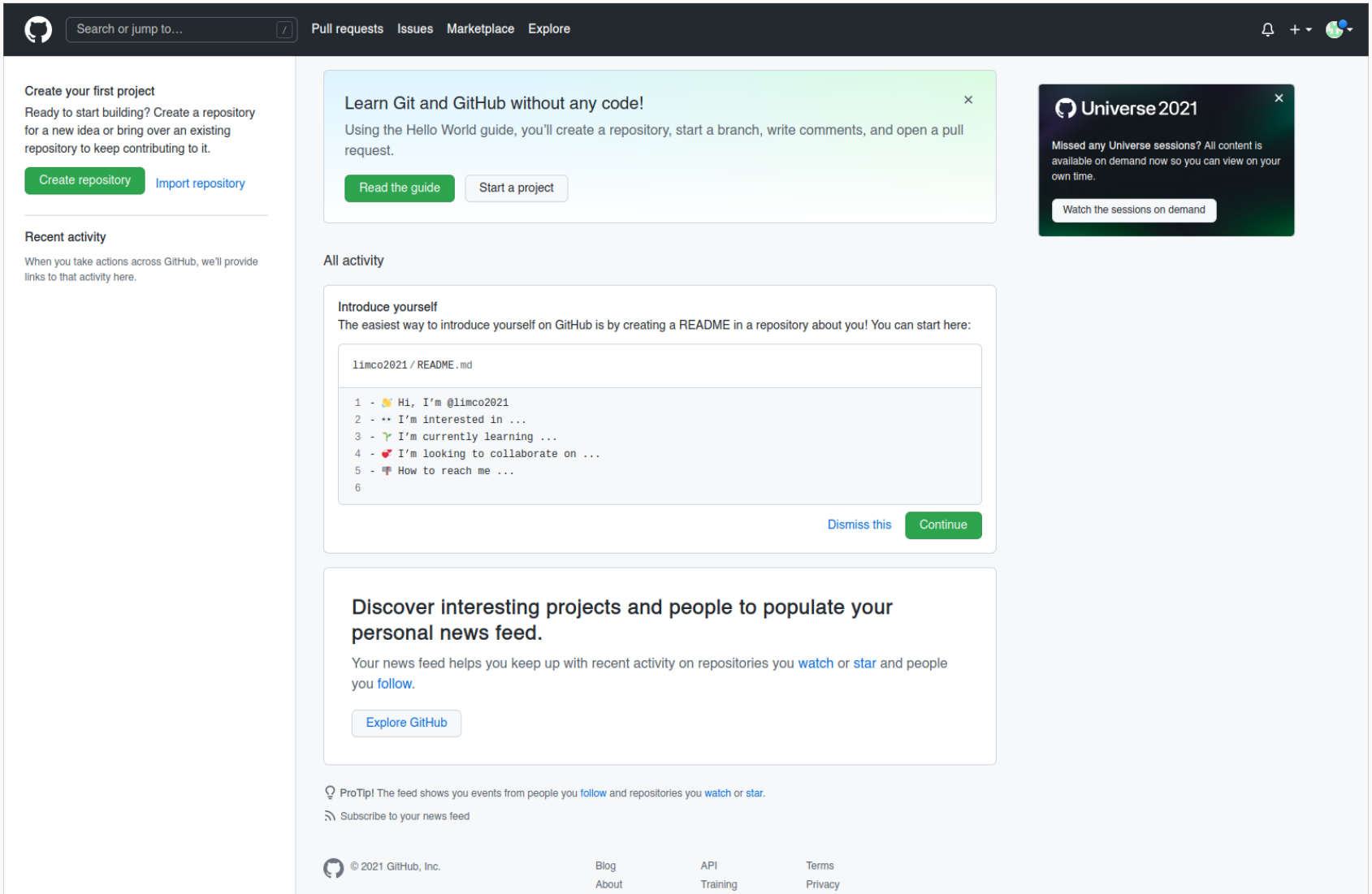
La **prima operazione** da compiere è quella di registrarsi su GitHub.

1. Inserite un indirizzo E-Mail (e.g, `n.cognome@unipi.it`) e fate click sul bottone: 

,
2. Seguite le istruzioni che vi chiederanno una **password** – inseritene una diversa – da quelle delle credenziali di ateneo,
3. Inserite uno **username**, è importante che sia facile, sarà poi la radice delle vostre **pagine web**. Ad esempio, `fdurastante` per fdurastante.github.io.
4. Segnate `n` per le mail pubblicitarie,
5. Confermate e risolvete il problema per farvi riconoscere come esseri umani.
6. Inserite il codice di verifica che vi è stato mandato via mail e scegliete l'opzione `free` del servizio.

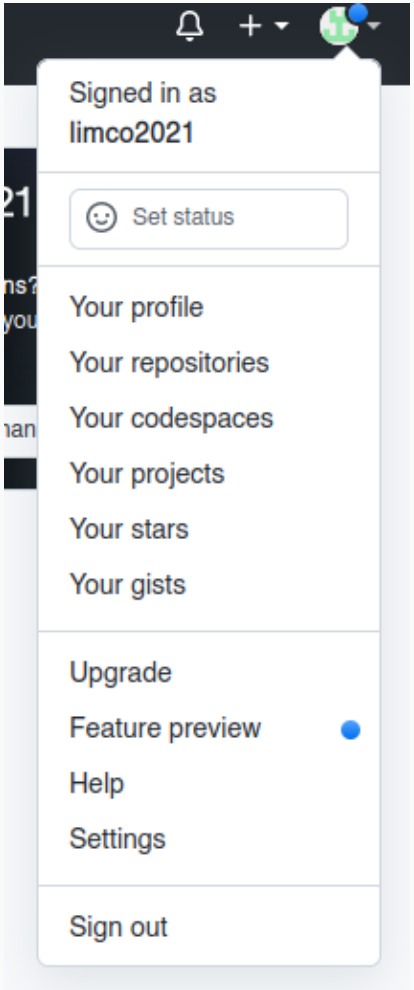
Siete ora **loggati** sul vostro account GitHub.

Registrarsi su GitHub



Caricare una chiave SSH

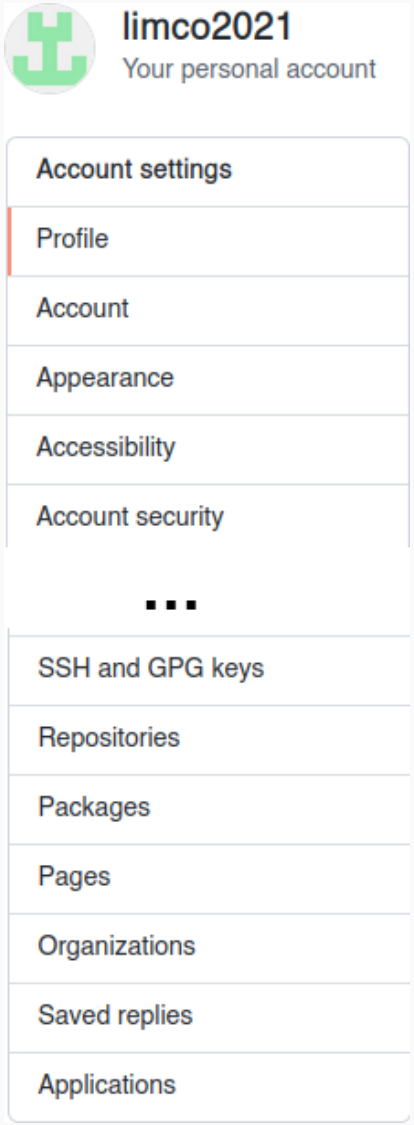
Per **scambiare file in modo sicuro** è necessario comunicare a GitHub una `chiave ssh`.



1. Dal menù in alto a destra si faccia click per aprire il menù a tendina e si selezioni `Settings`.

Caricare una chiave SSH

Per **scambiare file in modo sicuro** è necessario comunicare a GitHub una `chiave ssh`.



1. Dal menù in alto a destra si faccia click per aprire il menù a tendina e si selezioni `Settings`.
2. Nella nuova pagina che si aprirà dal menù di sinistra si selezioni `SSH and GPG Keys`

Caricare una chiave SSH

Per **scambiare file in modo sicuro** è necessario comunicare a GitHub una `chiave ssh`.

1. Dal menù in alto a destra si faccia click per aprire il menù a tendina e si selezioni `Settings`.
2. Nella nuova pagina che si aprirà dal menù di sinistra si selezioni `SSH and GPG Keys`
3. Possiamo caricare la chiave facendo click sul bottone: 

Generare una chiave SSH

Loggarsi via `ssh` su una macchina `mathsgalore`. Poi eseguire il comando: `ssh-keygen -t ed25519 -C "your_email@example.com"` con la mail con cui ci si è registrati a GitHub. Dopo aver generato la chiave che, se lasciata con il nome standard sarà, `id_ed25519`, si esegue `ssh-add ~/.ssh/id_ed25519`.

Caricare una chiave SSH

Per **scambiare file in modo sicuro** è necessario comunicare a GitHub una `chiave ssh`.

1. Dal menù in alto a destra si faccia click per aprire il menù a tendina e si selezioni `Settings`.
2. Nella nuova pagina che si aprirà dal menù di sinistra si selezioni `SSH and GPG Keys`
3. Possiamo caricare la chiave facendo click sul bottone: 

4. Copiamo l'**intero contenuto** del file `~/.ssh/id_ed25519.pub` e facciamo click su 

.

Caricare una chiave SSH

Per **scambiare file in modo sicuro** è necessario comunicare a GitHub una `chiave ssh`.

1. Dal menù in alto a destra si faccia click per aprire il menù a tendina e si selezioni `Settings`.
2. Nella nuova pagina che si aprirà dal menù di sinistra si selezioni `SSH and GPG Keys`
3. Possiamo caricare la chiave facendo click sul bottone: 

4. Copiamo l'**intero contenuto** del file `~/.ssh/id_ed25519.pub` e facciamo click su 

.

Adesso possiamo **scambiare file** con il nostro servizio Git in **maniera sicura**.

Il nostro primo repository

Un **repository** viene solitamente utilizzato per organizzare un **singolo progetto**. I *repository* possono contenere cartelle e file, immagini, video, fogli di calcolo e set di dati, ed in generale tutto ciò di cui il vostro progetto ha bisogno.

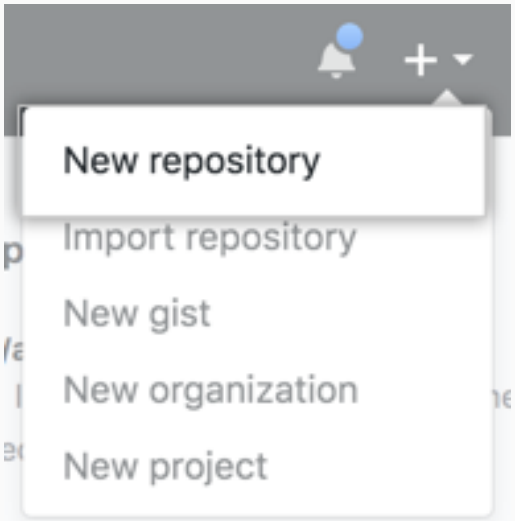
È buona norma per repository includere un file `README`, questo contiene **informazioni sul progetto**. GitHub semplifica l'aggiunta di uno nello stesso momento in cui create il vostro nuovo repository. Offre anche altre opzioni comuni come un file di licenza.

Se avete deciso di non usare GitHub...
Potete comunque seguire il resto della lezione "poggiando" i vostri *repository* di prova sulle macchine *mathsgalore* come remoto:

```
utente@macchinalocale:$ ssh utente@mathsgalore<-2-3-4>.unipi.it
utente@mathsgalore<-2-3-4>:$ mkdir mygitserver; cd mygitserver;
utente@mathsgalore<-2-3-4>:$ git init --bare hello-world.git
```

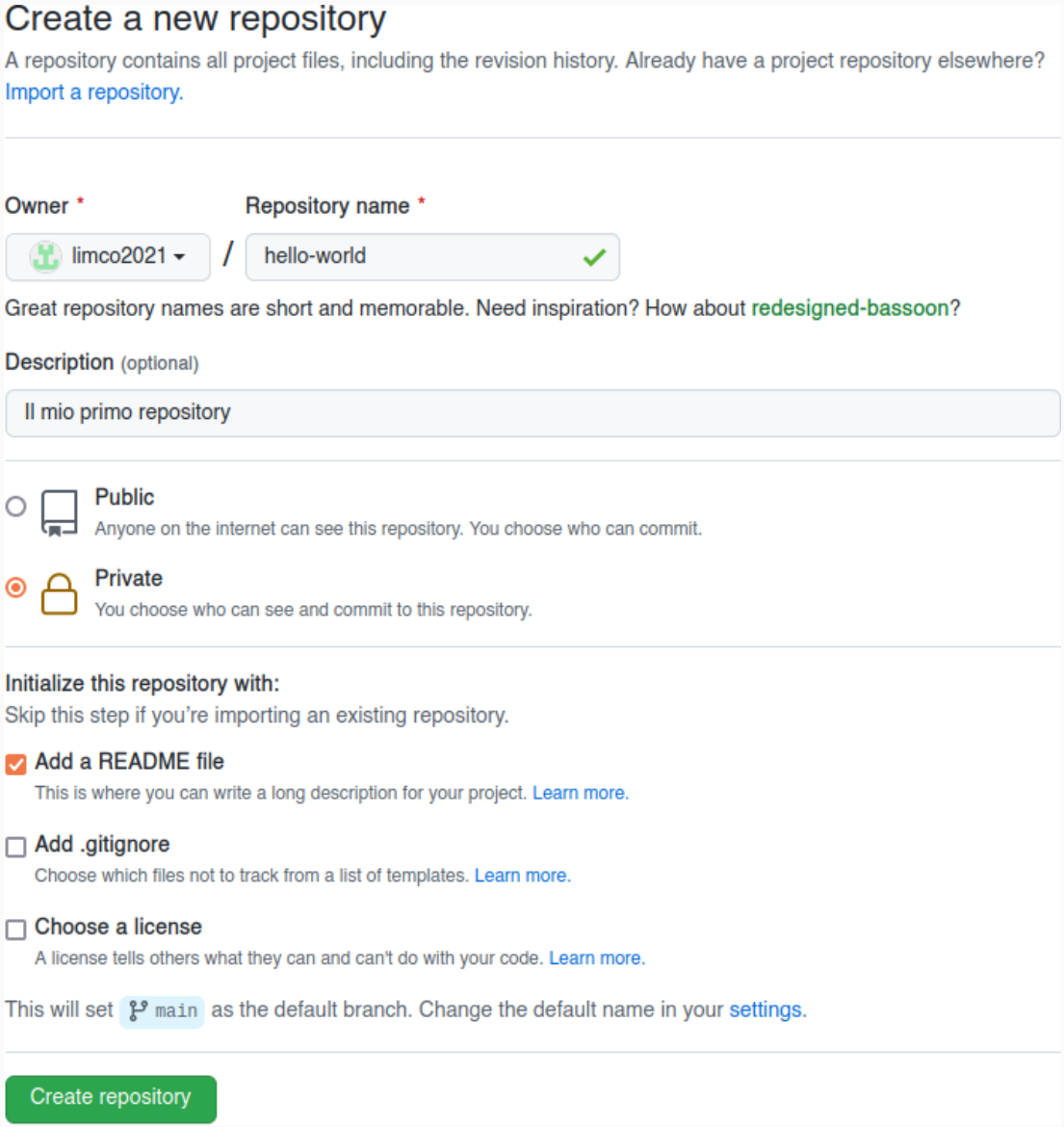
Hello-world repository

Costruiamolo passo-passo.



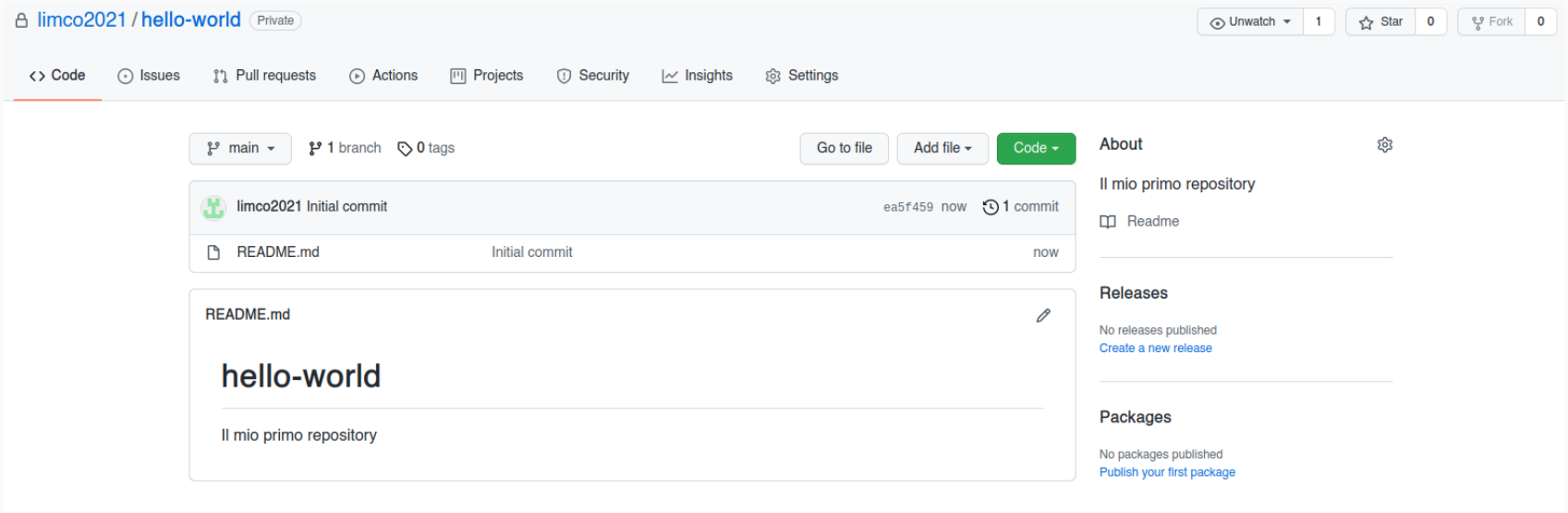
1. Nell'angolo in alto a destra di qualsiasi pagina, si scelga il menu a discesa e seleziona **New repository**.
2. Nella casella **Repository name**, si inserisca `hello-world`.
3. Nella casella **Description box**, si inserisca una breve descrizione: "Il mio primo *repository*"
4. Si selezionino le opzioni **Private** e **Add a README file**
5. Si faccia click su Create repository

Hello-world repository



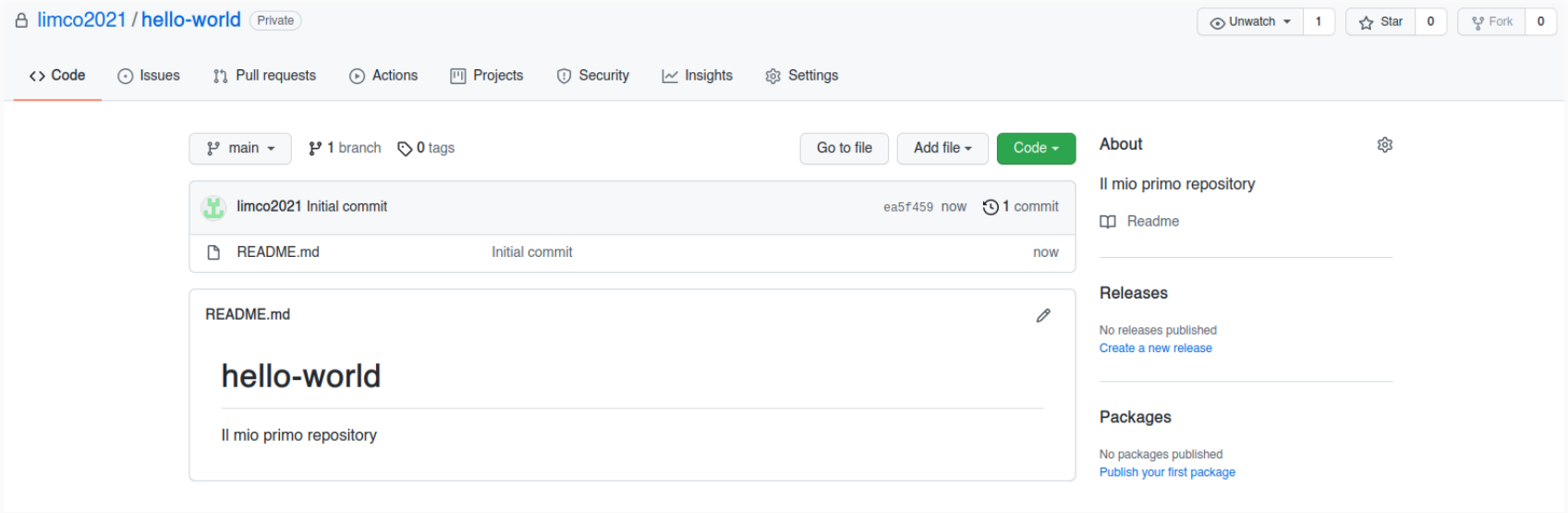
Hello-world repository

Dopo aver dato la conferma, vedremo il nostro nuovo *repository* che conterrà al suo interno solamente il file `README`.



Hello-world repository

Dopo aver dato la conferma, vedremo il nostro nuovo *repository* che conterrà al suo interno solamente il file `README`.



- Adesso abbiamo un luogo per i nostri file sul *server remoto*. Dobbiamo imparare ad interagire con esso per **recuperare i file**, **modificarli** e **sincronizzare le modifiche**.

Clone

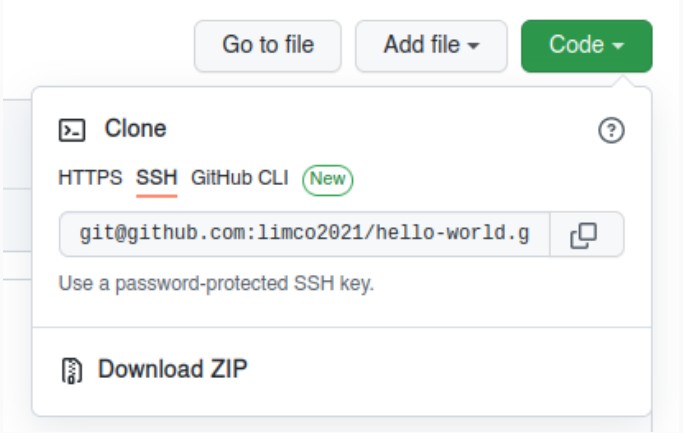
Per **prima cosa**, dobbiamo creare un **punto di sincronizzazione** per il materiale sulla nostra **macchina locale**. Questa operazione è detta operazione di `clone` ed è composta di due parti:

Clone

Per **prima cosa**, dobbiamo creare un **punto di sincronizzazione** per il materiale sulla nostra **macchina locale**. Questa operazione è detta operazione di `clone` ed è composta di due parti:

1. Per prima cosa recuperiamo l'**indirizzo presso cui il nostro repository risiede**.

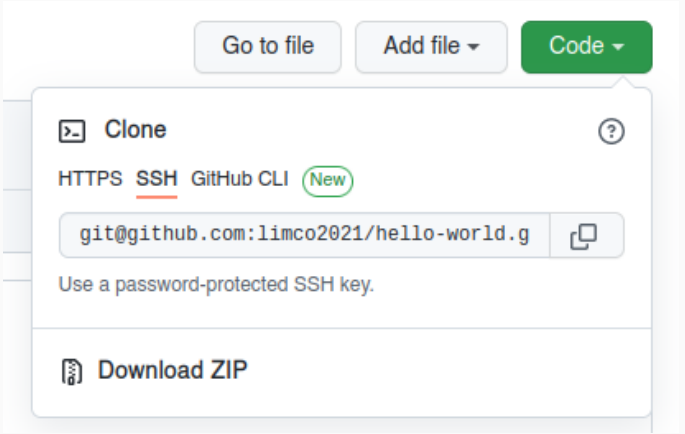
   Se avete deciso di **non usare** GitHub, questo è:




```
utente@mathsgalore<-2-3-4>.unipi.it:mygitserver/hello-world.git
```

Clone

Per **prima cosa**, dobbiamo creare un **punto di sincronizzazione** per il materiale sulla nostra **macchina locale**. Questa operazione è detta operazione di `clone` ed è composta di due parti:



1. Per prima cosa recuperiamo l'**indirizzo presso cui il nostro repository risiede**.

2. In una shell sulla macchina *mathsgalore* su cui abbiamo generato la chiave SSH eseguiamo il comando:
   `git clone git@github.com:limco2021/hello-world.git`
   sostituendo all'indirizzo quello ottenuto allo **step 1**.

```
f.durastante@mathsgalore4:~$ git clone git@github.com:limco2021/hello-world.git
Cloning into 'hello-world'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

Clone

Se eseguiamo il comando `ls` osserviamo di aver creato una cartella di nome `hello-world`, o, più in generale, chiamata come il *repository*.

Se facciamo `cd hello-world` seguito da `ls`, osserviamo che al suo interno troviamo il file `README`.

```
f.durastante@mathsgalore4:~$ ls
danteshort.txt  dante-tn.txt  dante.txt  examples.desktop  hello-world  prova  texample
f.durastante@mathsgalore4:~$ cd hello-world/
f.durastante@mathsgalore4:~/hello-world$ ls
README.md
```

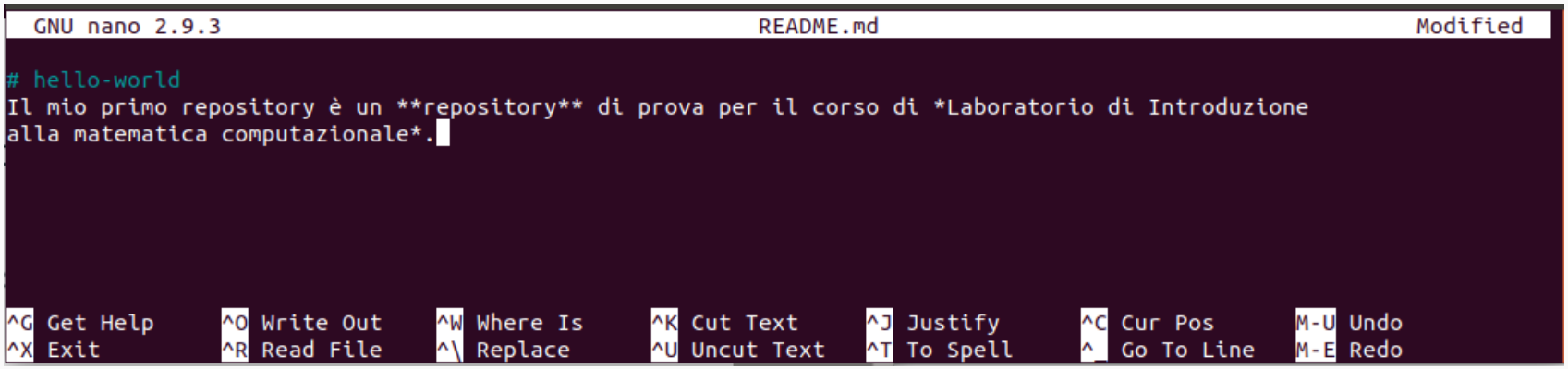
Adesso **abbiamo una copia** di tutto quello che è contenuto nel **repository** `hello-world` anche nella nostra **macchina locale**.

Glossario: clone
Un *clone* è sia la copia di un *repository* che risiede sul tuo computer invece che sul server di un sito web da qualche parte, sia l'atto di fare quella copia. Quando crei un *clone*, puoi modificare i file nel tuo editor preferito e **usare Git per tenere traccia delle tue modifiche** senza dover essere online. Il *repository* che hai clonato è ancora connesso alla versione remota in modo da poter inviare le modifiche locali al remoto per mantenerle sincronizzate quando sei online.

Facciamo una modifica:

Supponiamo ora di voler **modificare il nostro file README**.

Eseguiamo: `nano README.md` e scriviamo qualcosa nell'editor:



facciamo CTRL + O per salvare (confermando il nome del file con un ENTER) e poi CTRL + X per chiudere.

Ora c'è **una differenza** tra la **versione locale** e la **versione remota**!

git status

se eseguiamo il comando `git status`:

```
f.durastante@mathsgalore4:~/hello-world$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

no changes added to commit (use "git add" and/or "git commit -a")
```

ci vengono comunicate alcune informazioni:

- ci troviamo sul **branch** `main` (torneremo a discuterne tra poco),
- tutto quello che c'è sul repository online coincide con quello che abbiamo,
- abbiamo delle **differenze locali** e possiamo decidere tra due cose:
  - aggiungere le nostre modifiche: `git add README.md`,
  - riportare il file allo stato originale: `git checkout README.md`,

add e commit

Abbiamo deciso che **le nostre modifiche sono da preservare** quindi procediamo a fare:

`git add README.md`

e ripetiamo di nuovo `git status`:

```
f.durastante@mathsgalore4:~/hello-world$ git add README.md
f.durastante@mathsgalore4:~/hello-world$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
```

che ci dice che il file è pronto per essere inserito in un `commit`.

Dobbiamo tuttavia prima identificarci, per cui eseguiamo i comandi:

```
git config user.email "fabio.durastante@gmx.com"
git config user.name "Fabio Durastante"
```

add e commit

Siamo pronti per **eseguire il commit**:

`git commit -m "Aggiunte informazioni al README"`

che assegna alla nostra modifica un messaggio descrittivo dopo

`-m " messaggio di commit "`,

che deve descrivere **brevemente** l'argomento della **modifica** fatta.

```
f.durastante@mathsgalore4:~/hello-world$ git commit -m "Aggiunte informazioni al README"
[main 7731206] Aggiunte informazioni al README
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Possiamo considerare di nuovo cosa succede se eseguiamo `git status`:

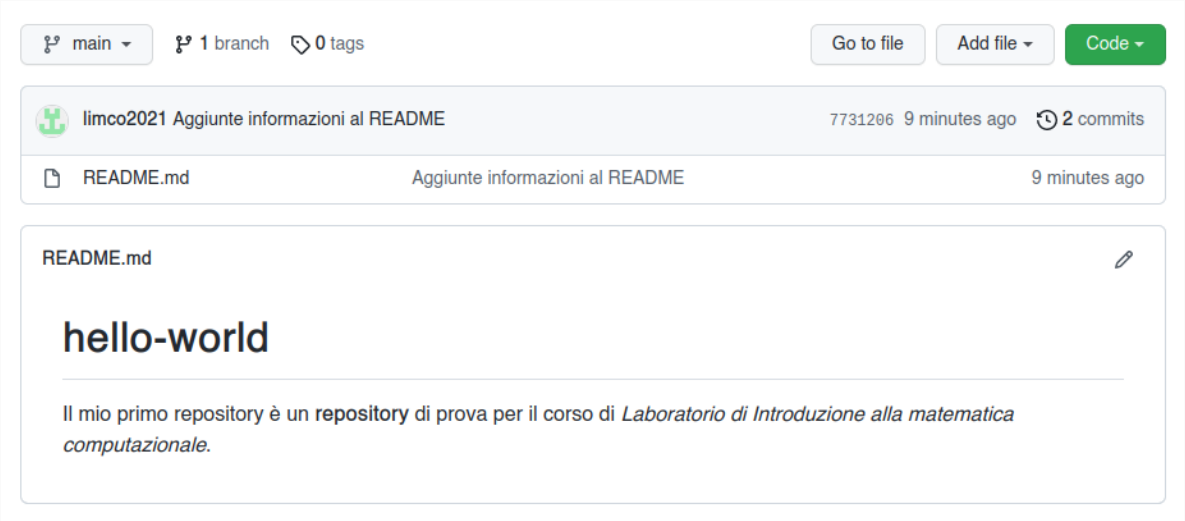
```
f.durastante@mathsgalore4:~/hello-world$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

git push

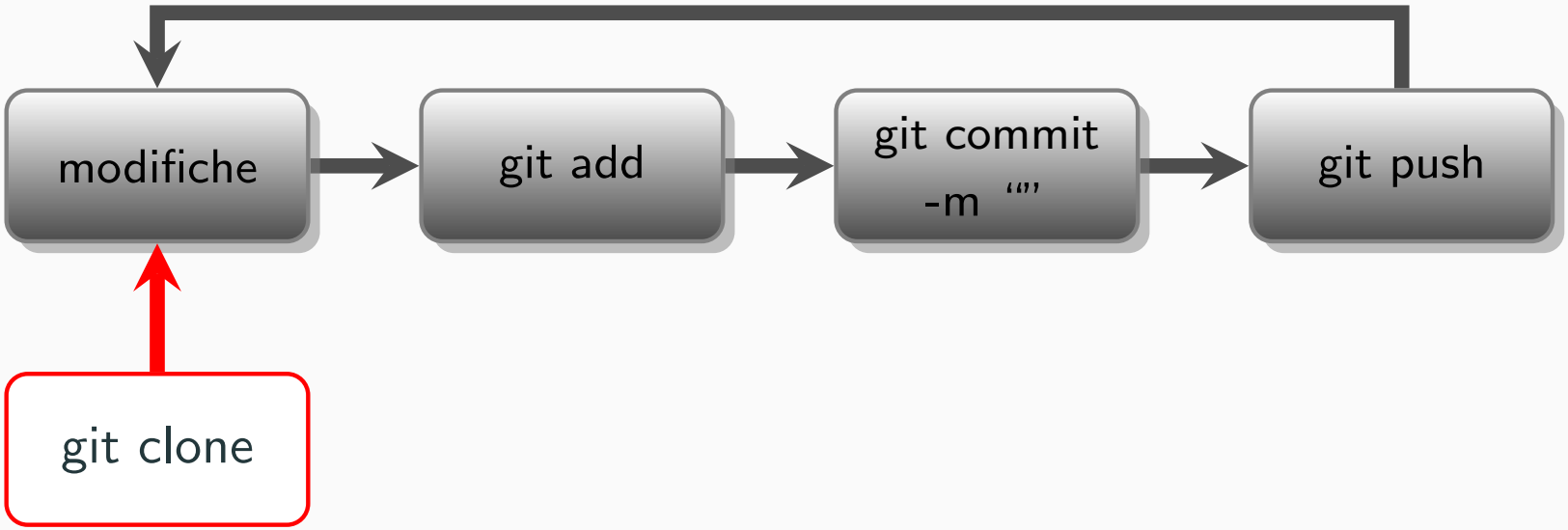
La modifica è pronta per essere "spinta" sul server remoto. Questa operazione è detta `push`:

```
f.durastante@mathsgalore4:~/hello-world$ git push
Warning: Permanently added the ECDSA host key for IP address '140.82.121.3' to the list of known hosts.
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 381 bytes | 381.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:limco2021/hello-world.git
   ea5f459..7731206  main -> main
```

Così facendo abbiamo ripristinato la sincronia tra locale e remoto!
Torniamo a vedere cosa è cambiato sull'interfaccia web:



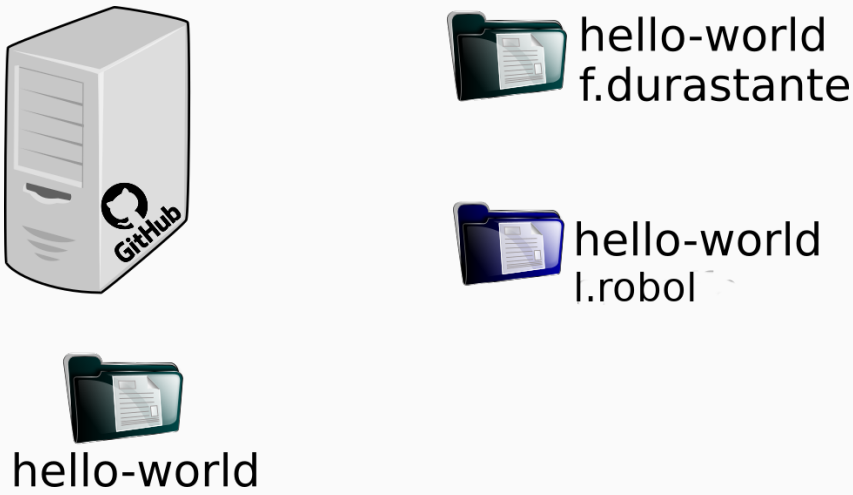
Riassumiamo



1. Creazione di un repository su GitHub,
2. Clone su una macchina locale (`git clone`),
3. Modifiche ai file locali fino alla soddisfazione (codice C, Matlab, sage, LaTeX, pagine web),
4. Preparazione di un *commit* (`git add ...`, `git commit -m " "`),
5. Sincronizzare il remoto (`git push`).

git pull

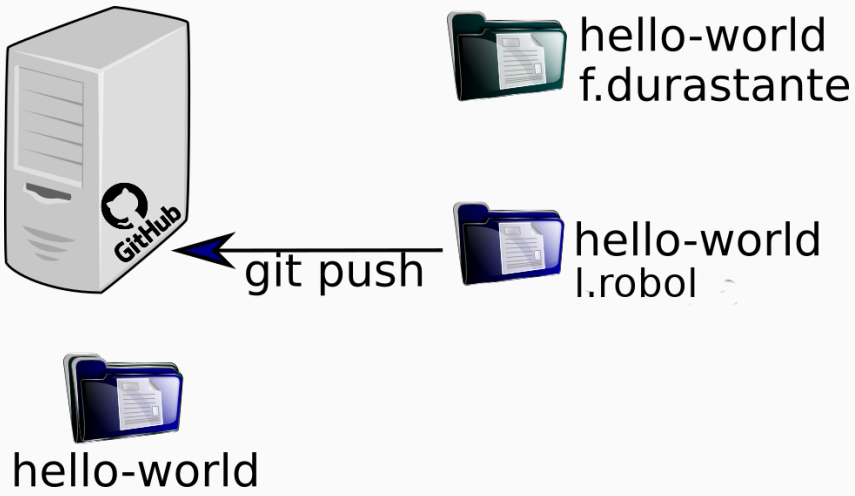
Immaginiamo ora di avere due (o più collaboratori) o macchine con cui lavoriamo sullo stesso *repository*.



L'utente `l.robol` ha fatto una sua modifica in locale, `f.durastante` e l'online non ne sanno nulla.

git pull

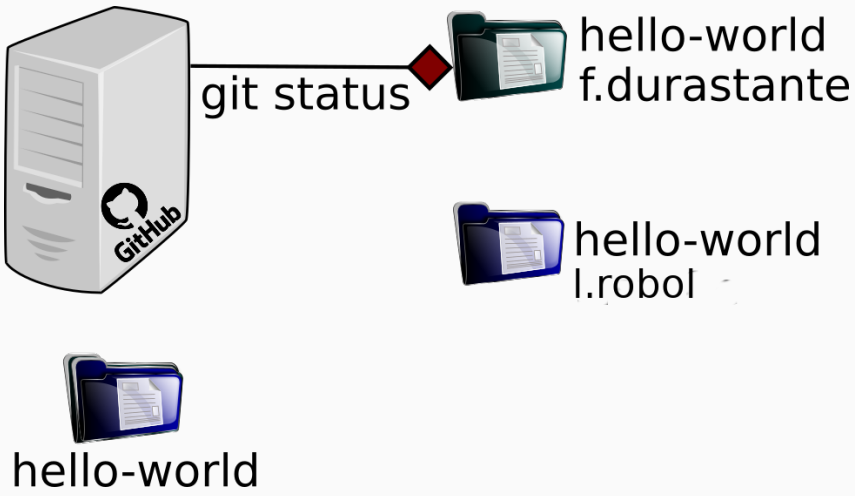
Immaginiamo ora di avere due (o più collaboratori) o macchine con cui lavoriamo sullo stesso *repository*.



`l.robol` è soddisfato dei suoi cambi, fa `git add`, `git commit` e `git push`: ora la versione sul *repository* è stata aggiornata.

git pull

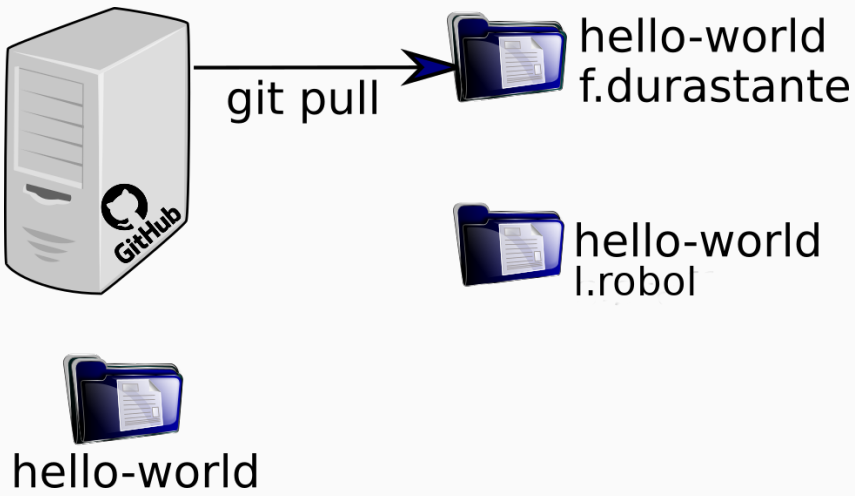
Immaginiamo ora di avere due (o più collaboratori) o macchine con cui lavoriamo sullo stesso *repository*.



Prima di mettersi a lavorare `f.durastante` si domanda se la sua versione è aggiornata, fa `git status` e scopre di essere indietro di un `commit`.

git pull

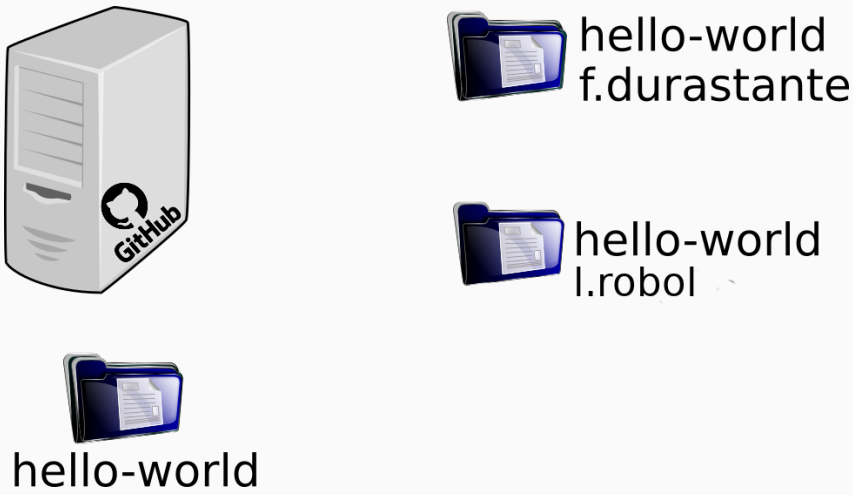
Immaginiamo ora di avere due (o più collaboratori) o macchine con cui lavoriamo sullo stesso *repository*.



Vuole mettersi in pari con le modifiche e chiama quindi `git pull` per "tirare" giù dal *repository* le modifiche.

git pull

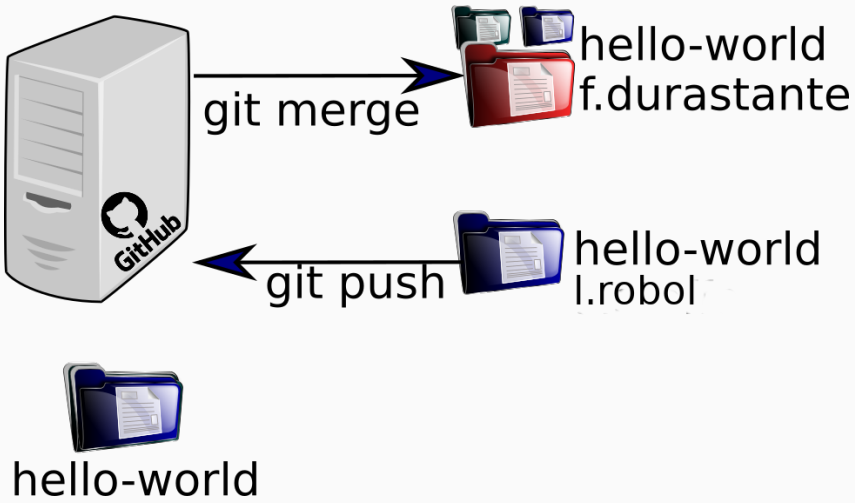
Immaginiamo ora di avere due (o più collaboratori) o macchine con cui lavoriamo sullo stesso *repository*.



Adesso tutte le versioni locali e i *repository* coincidono.

git pull

Immaginiamo ora di avere due (o più collaboratori) o macchine con cui lavoriamo sullo stesso *repository*.

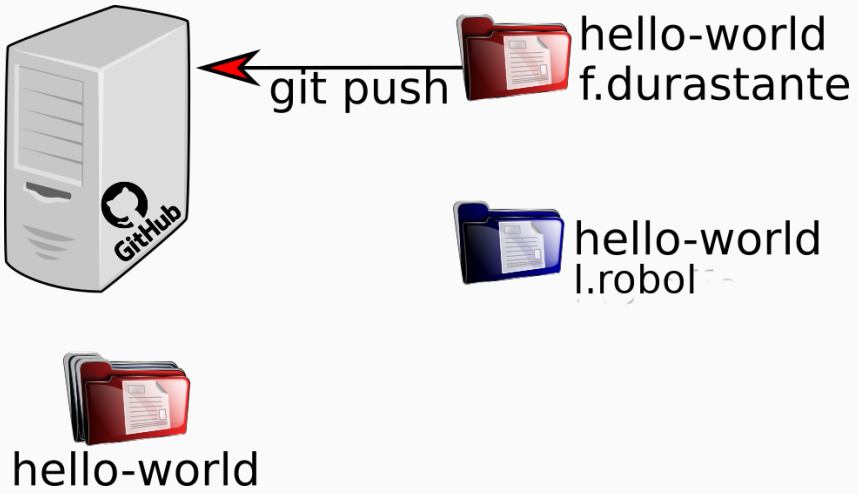


Risolvere conflitti

È facile immaginare situazioni in cui si creano dei conflitti tra le diverse versioni di diversi utenti, ad esempio `l.robol` e `f.durastante` fanno entrambi delle modifiche e tentano di farne `push`. Il primo ci riesce, il secondo dovrà fare un'operazione di `git merge` per risolvere i conflitti sulla sua copia locale prima di fare `commit` e `push`.

git pull

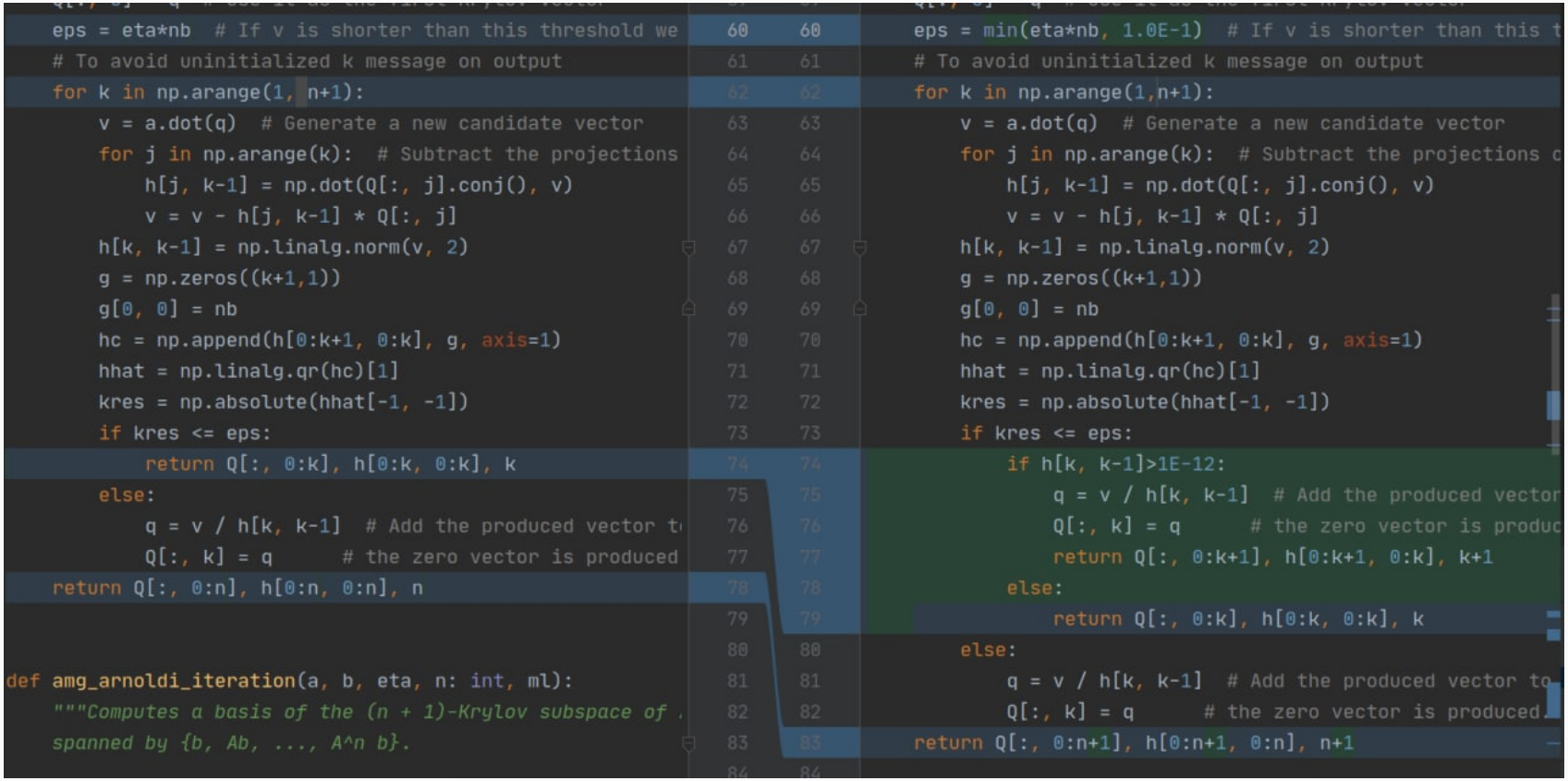
Immaginiamo ora di avere due (o più collaboratori) o macchine con cui lavoriamo sullo stesso *repository*.



Risolvere conflitti

È facile immaginare situazioni in cui si creano dei conflitti tra le diverse versioni di diversi utenti, ad esempio `l.robol` e `f.durastante` fanno entrambi delle modifiche e tentano di farne `push`. Il primo ci riesce, il secondo dovrà fare un'operazione di `git merge` per risolvere i conflitti sulla sua copia locale prima di fare `commit` e `push`.

Un esempio visivo di un'operazione di merge

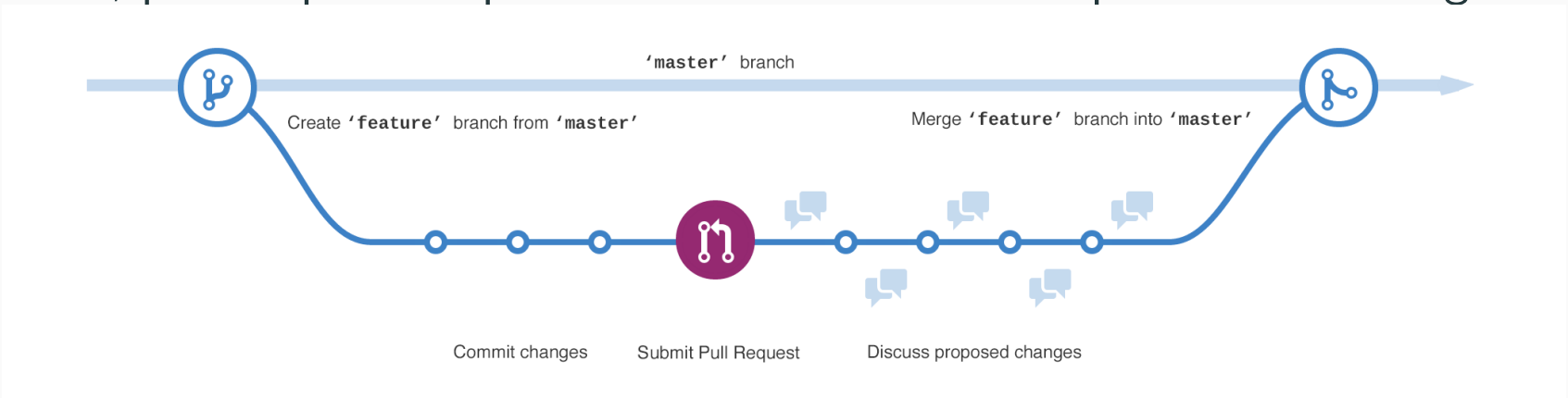


Branch

Il **branching** (ramificazione) consente di avere diverse versioni di un repository contemporaneamente.

Per impostazione predefinita, ogni nuovo *repository* su GitHub ha un **branch chiamato main** che è considerato il ramo delle versioni "definitive". L'idea dei branch è quella di usarli per sperimentare e apportare modifiche **prima** di renderle definitive in *main*.

Quando si crea un *branch* dal *main*, si sta facendo una copia, o un'**istantanea**, così com'è in quel momento. Se qualcun altro ha apportato modifiche al ramo principale mentre stavi lavorando sul tuo ramo, puoi importare quelle modifiche tramite un'operazione di merge.



Creare e muoversi tra i branch

Per **creare un nuovo** *branch* si usa il comando:

```
git checkout -b nomedelnuovobranch
```

```
f.durastante@mathsgalore4:~/hello-world$ git checkout -b nuovofile
Switched to a new branch 'nuovofile'
f.durastante@mathsgalore4:~/hello-world$ git status
On branch nuovofile
nothing to commit, working tree clean
```

per **muoversi tra i branch** si usa invece il comando:

```
git checkout nomedelbranch
```

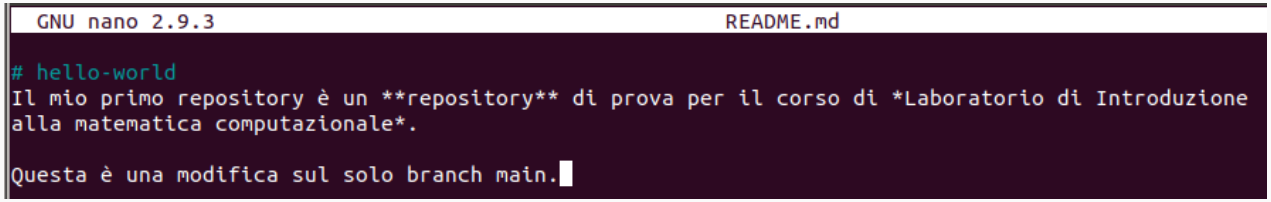
```
f.durastante@mathsgalore4:~/hello-world$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

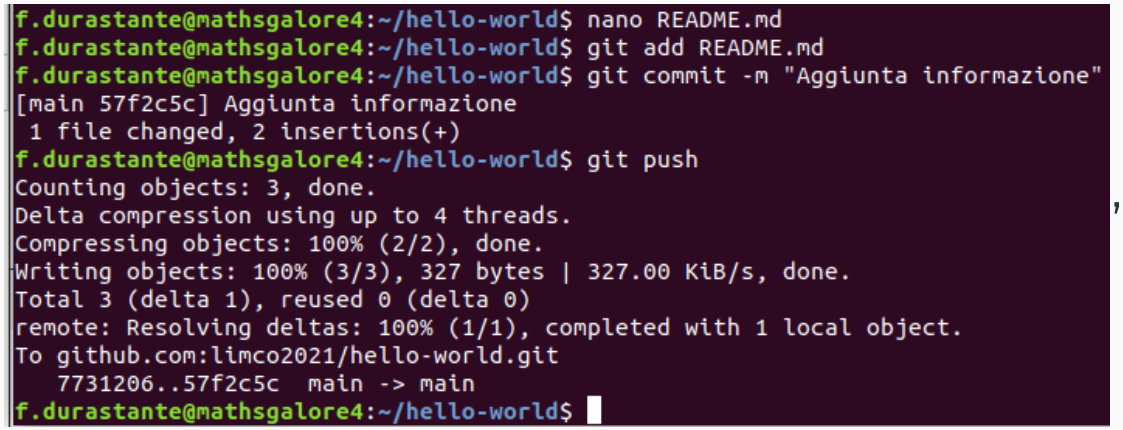
Adesso dobbiamo rendere partecipe il *repository* del nuovo branch, facendo:

```
git push --set-upstream origin nuovofile
```

Un esercizio di branching e merging

Ora che siamo tornati in `main`, facciamo una nuova modifica al file `README.md`, ne facciamo `add`, `commit` e `push`.

1. 

2. 

,

Un esercizio di branching e merging

Ora che siamo tornati in `main`, facciamo una nuova modifica al file `README.md`, ne facciamo add, `commit` e `push`.

1.
```
  GNU nano 2.9.3                        README.md

# hello-world
Il mio primo repository è un **repository** di prova per il corso di *Laboratorio di Introduzione
alla matematica computazionale*.

Questa è una modifica sul solo branch main.
```
2. `git add README.md; git commit -m "Aggiunta informazione"; git push`,

Torniamo nel *branch* che avevamo appena creato con `git checkout nuovofile`, possiamo chiedere ora quali sono le differenze rispetto al branch *main*: `git diff main`:

```
f.durastante@mathsgalore4:~/hello-world$ git diff main
diff --git a/README.md b/README.md
index b4fc1cd..3753df7 100644
--- a/README.md
+++ b/README.md
@@ -1,5 +1,3 @@
 # hello-world
 Il mio primo repository è un **repository** di prova per il corso di *Laboratorio di Introduzione
 alla matematica computazionale*.
-
-Questa è una modifica sul solo branch main.
```

Un esercizio di branching e merging

Decidiamo che questa modifica è rilevante per quello che vogliamo fare e la importiamo nel nostro branch: `git merge main`.

```
f.durastante@mathsgalore4:~/hello-world$ git merge main
Updating 7731206..57f2c5c
Fast-forward
 README.md | 2 ++
 1 file changed, 2 insertions(+)
```

In questo caso `git` si accorge che c'è solo un commit di differenza e l'effetto è analogo a quello di aver fatto un *git pull*.

Adesso che abbiamo **sistemato la nostra versione locale**, possiamo concludere l'operazione con un `git push`.

Documentazione e Informazioni

Queste sono le **istruzioni basilari** per utilizzare Git e GitHub per gestire un piccolo progetto in una o poche persone. È possibile utilizzarlo in modo **più sofisticato**, ma questo sfugge agli interessi limitati che abbiamo qui.

Alcune referenze utili sono:

training.github.com/downloads/it/github-git-cheat-sheet/

e

git-scm.com/docs

con in particolare la pagina: https://ndpsoftware.com/git-cheatsheet.html.

Nel **tempo che ci rimane** facciamo il setup di un repository GitHub che possa fare da host per delle pagine web (e che useremo poi nella prossima e ultima lezione di questa prima parte).

GitHub Pages

GitHub Pages

GitHub offre un servizio di **hosting per pagine web** basato di nuovo su un particolare repository.

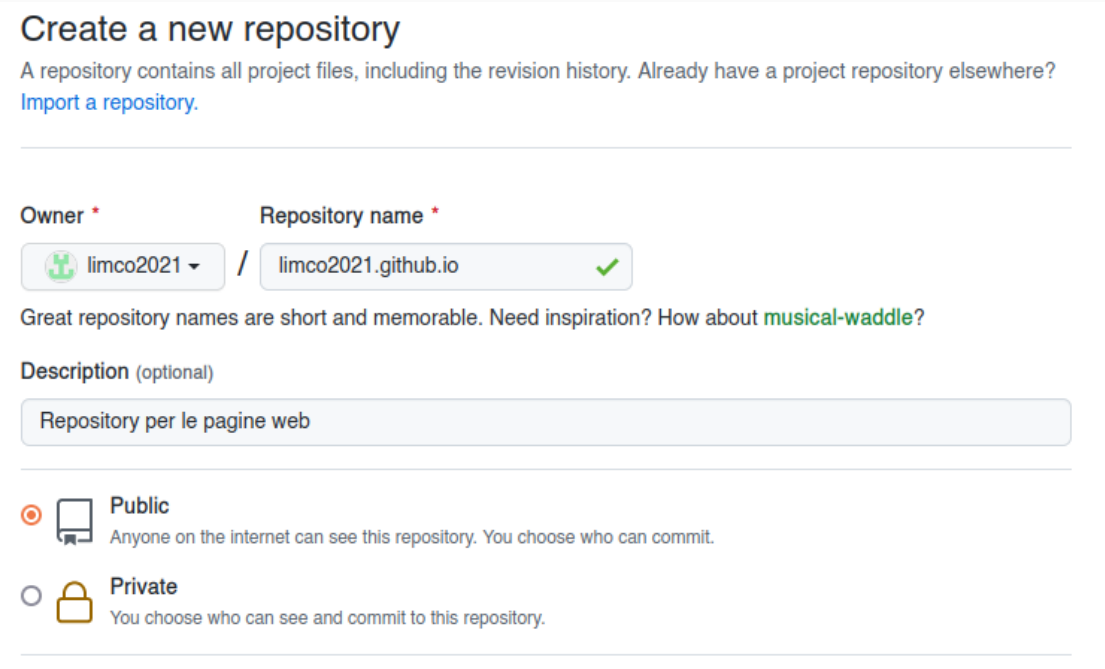
Facciamo i pochi passi necessari ad attivarlo e a *caricare una prima pagina web di prova*.

GitHub Pages

GitHub offre un servizio di **hosting per pagine web** basato di nuovo su un particolare repository.

Facciamo i pochi passi necessari ad attivarlo e a *caricare una prima pagina web di prova*.

1. Andiamo su GitHub e creiamo un nuovo **repository pubblico** chiamato username.github.io, dove *username* è il vostro nome utente su GitHub.



GitHub Pages

GitHub offre un servizio di **hosting per pagine web** basato di nuovo su un particolare repository.

Facciamo i pochi passi necessari ad attivarlo e a *caricare una prima pagina web di prova*.

1. Andiamo su GitHub e creiamo un nuovo **repository pubblico** chiamato username.github.io, dove *username* è il vostro nome utente su GitHub.
2. Abbiamo creato un **repository vuoto**, quindi dobbiamo fare qualche manovra aggiuntiva...

GitHub Pages

1. Cloniamo il repository (*username* va sostituito con il *vostro* username.)
   `git clone git@github.com:username/username.github.io.git`

```
f.durastante@mathsgalore4:~$ git clone git@github.com:limco2021/limco2021.github.io.git
Cloning into 'limco2021.github.io'...
warning: You appear to have cloned an empty repository.
```

GitHub Pages

1. Cloniamo il repository (*username* va sostituito con il *vostro* username.)
2. Generiamo la nostra pagina web di prova:

```
cd username.github.io
echo "Hello world!" > index.html
```

GitHub Pages

1. Cloniamo il repository (*username* va sostituito con il *vostro* username.)
2. Generiamo la nostra pagina web di prova:
3. **Inizializziamo il repository** e eseguiamo la catena `add`, `commit` e `push`

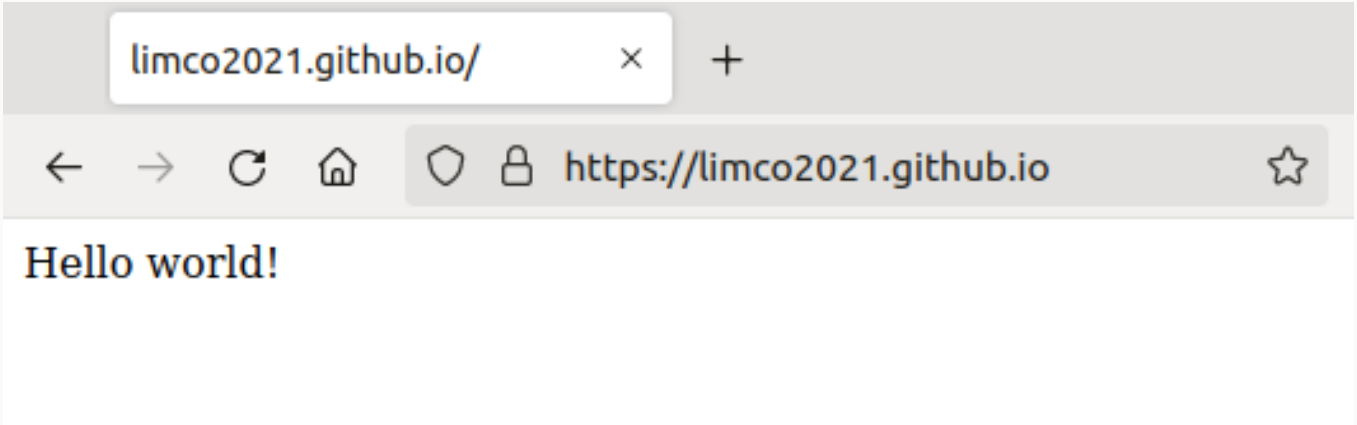
```
git init
git add index.html
git commit -m "La mia prima pagina web!"
git push
```

```
f.durastante@mathsgalore4:~/limco2021.github.io$ git init
Reinitialized existing Git repository in /home/f.durastante/limco2021.github.io/.git/
f.durastante@mathsgalore4:~/limco2021.github.io$ git add index.html
f.durastante@mathsgalore4:~/limco2021.github.io$ git commit -m "La mia prima pagina web!"
[master (root-commit) d3030fd] La mia prima pagina web!
 1 file changed, 1 insertion(+)
 create mode 100644 index.html
f.durastante@mathsgalore4:~/limco2021.github.io$ git push
Counting objects: 3, done.
Writing objects: 100% (3/3), 240 bytes | 240.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:limco2021/limco2021.github.io.git
 * [new branch]      master -> master
```

GitHub Pages

1. Cloniamo il repository (*username* va sostituito con il *vostro* username.)
2. Generiamo la nostra pagina web di prova:
3. **Inizializziamo il repository** e eseguiamo la catena `add`, `commit` e `push`

   ```
   git init
   git add index.html
   git commit -m "La mia prima pagina web!"
   git push
   ```
4. Possiamo ora aprire in un *browser* l'indirizzo `https://username.github.io` e vedere la nostra pagina web:



Generare pagine web

Nel prossimo laboratorio ci eserciteremo nel **generare pagine web** utilizzando in maniera diretta l'HTML.

Tuttavia, esistono diversi sistemi di generazione automatica di pagine web statiche. GitHub suggerisce di usare Jekyll, per cui trovate guide dettagliate su:

https://jekyllrb.com/

Ad **esempio** il sito: fdurastante.github.io è costruito con questo sistema.

In **ogni caso** per avere cognizione di causa di quello che si sta facendo è molto opportuno conoscere e saper leggere il codice HTML (anche quello generato da Jekyll).

Un servizio in house

I *macchinisti* del dipartimento hanno messo su un servizio GIT *in house* raggiungibile all'indirizzo:

`https://git.phc.dm.unipi.it/`

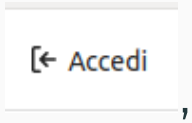
Si tratta di una installazione del servizio *open source*: **Gitea**

Un servizio in house

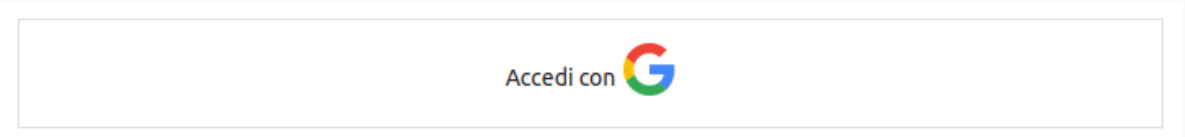
I *macchinisti* del dipartimento hanno messo su un servizio GIT *in house* raggiungibile all'indirizzo:

`https://git.phc.dm.unipi.it/`

Si tratta di una installazione del servizio *open source*: **Gitea**

- Potete utilizzarlo con le vostre **credenziali di ateneo**: 

, veicolate tramite il servizio di autenticazione Google:



Un servizio in house

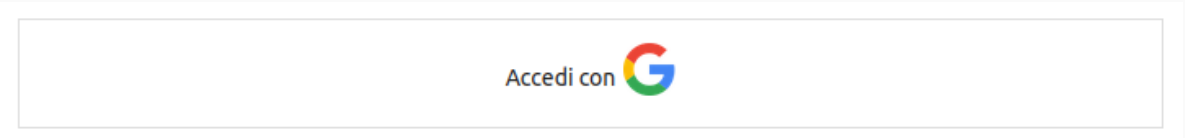
I *macchinisti* del dipartimento hanno messo su un servizio GIT *in house* raggiungibile all'indirizzo:

`https://git.phc.dm.unipi.it/`

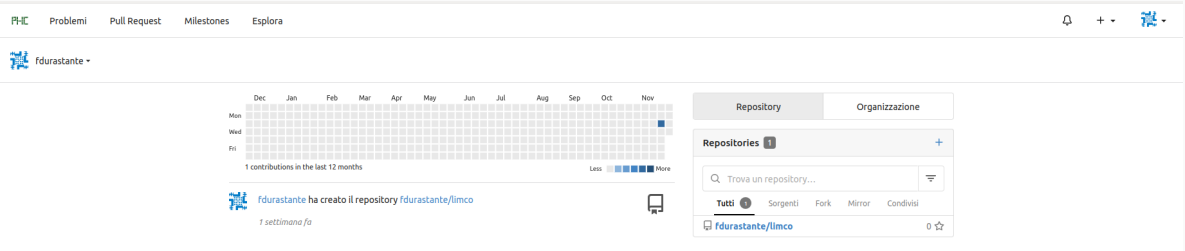
Si tratta di una installazione del servizio *open source*: **Gitea**

- Potete utilizzarlo con le vostre **credenziali di ateneo**: 

, veicolate tramite il servizio di autenticazione Google:

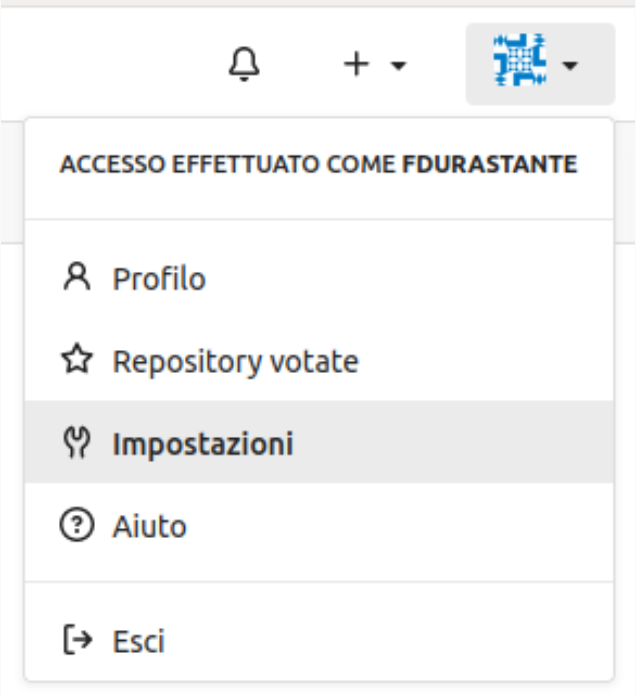


- Interfaccia e modalità di utilizzo sono molto simili a quelle di GitHub:



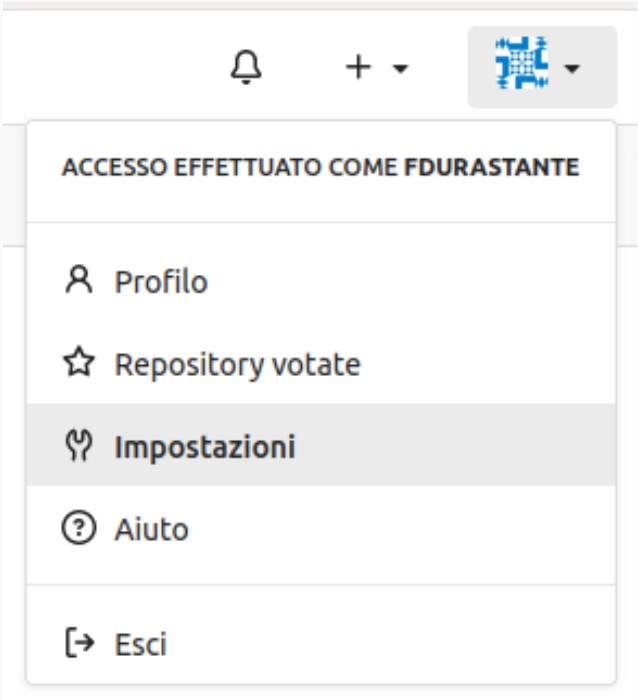
Un servizio in house

Dal menù impostazioni avete accesso ad alcune delle configurazioni che abbiamo visto per il servizio GitHub:

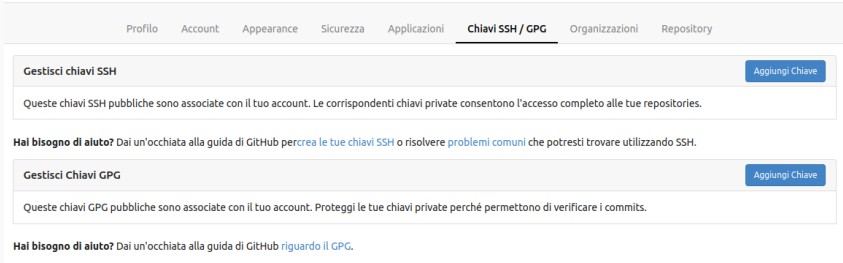


Un servizio in house

Dal menù impostazioni avete accesso ad alcune delle configurazioni che abbiamo visto per il servizio GitHub:

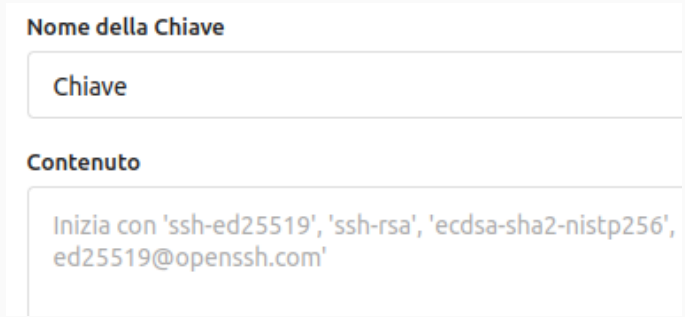


- Inserimento chiave SSH:

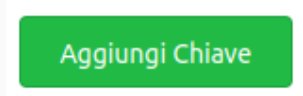


- 

- Che si inserisce in modo analogo:

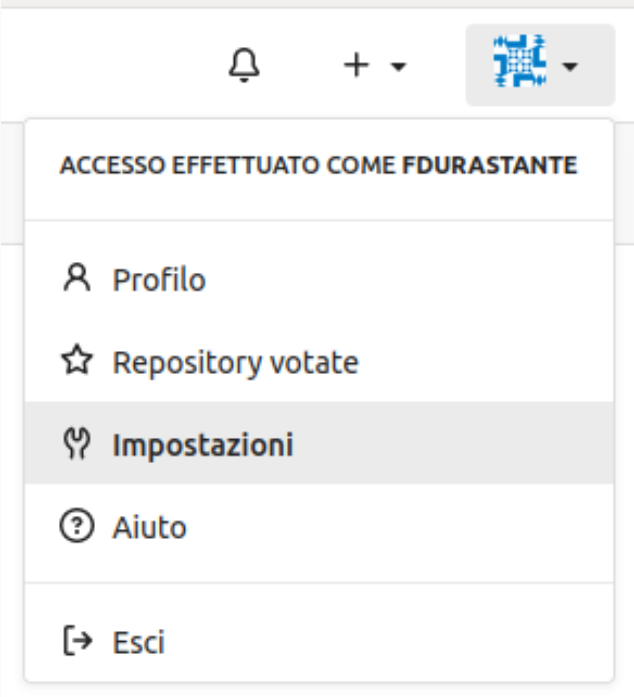


- Concludendo con:



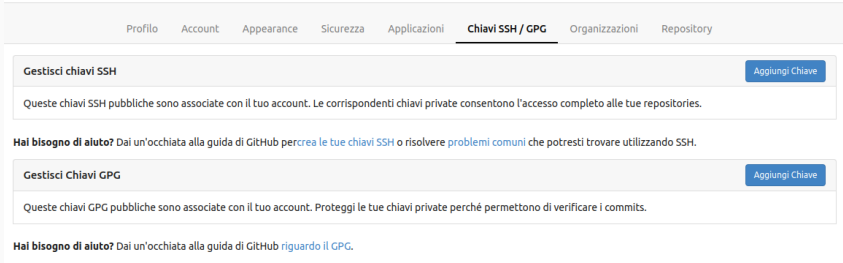
Un servizio in house

Dal menù impostazioni avete accesso ad alcune delle configurazioni che abbiamo visto per il servizio GitHub:



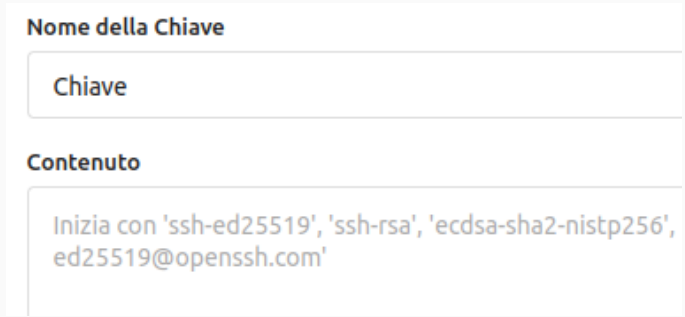
Fate un **account** qui **solo se avete intenzione di utilizzarlo**, aiutate i vostri colleghi macchinisti!

- Inserimento chiave SSH:



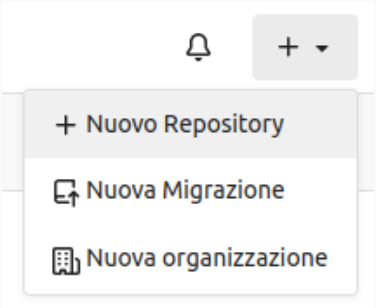
- 

- Che si inserisce in modo analogo:



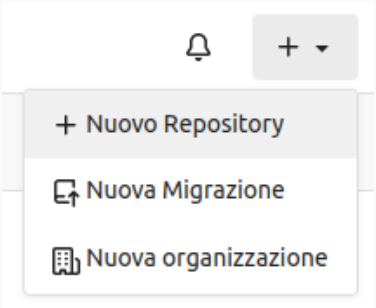
- Concludendo con:

Un servizio in house

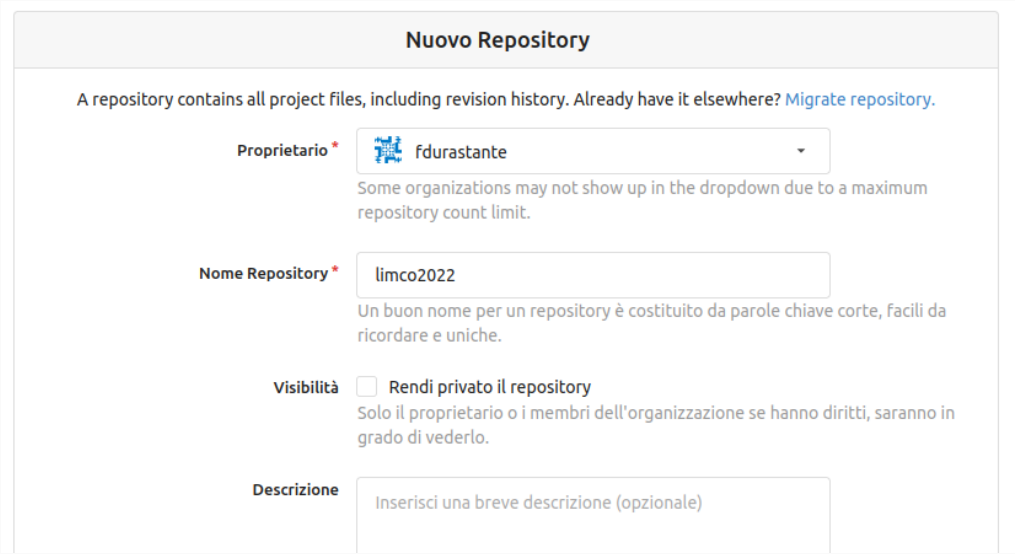


- Anche creare un nuovo repository è analogo a quanto già visto!

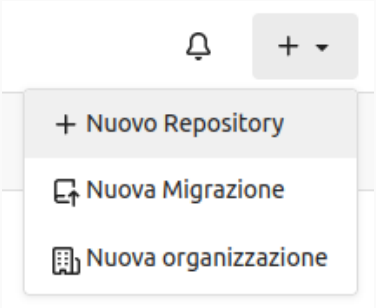
Un servizio in house



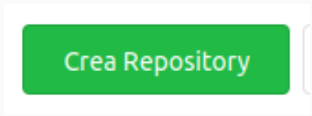
- Anche creare un nuovo repository è analogo a quanto già visto!

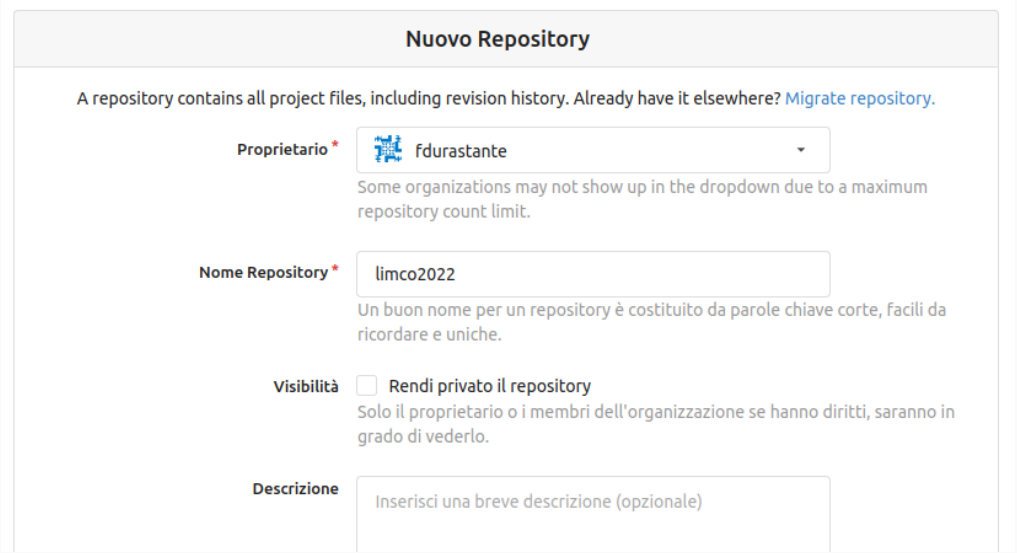


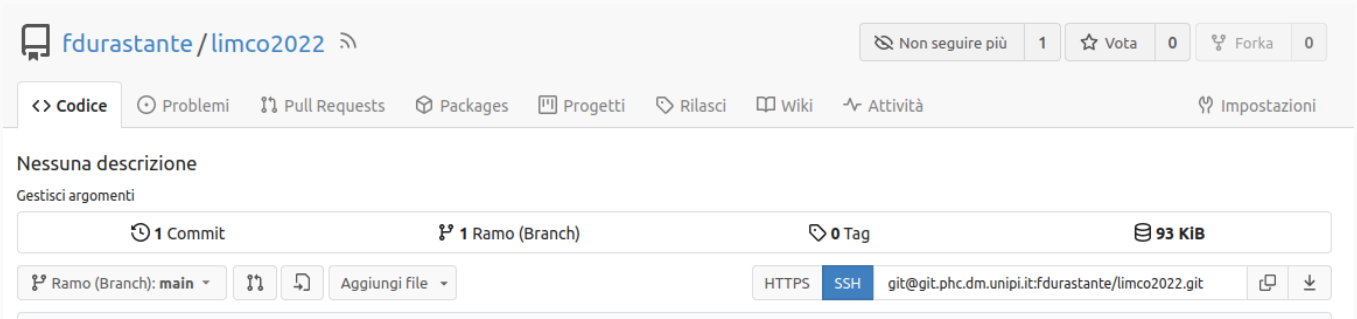
Un servizio in house



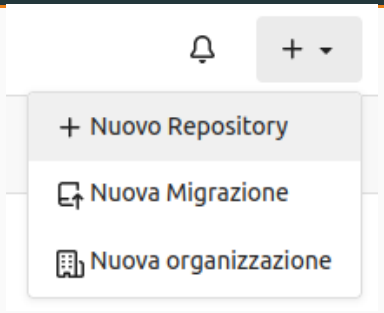
- Anche creare un nuovo repository è analogo a quanto già visto!
- E poi

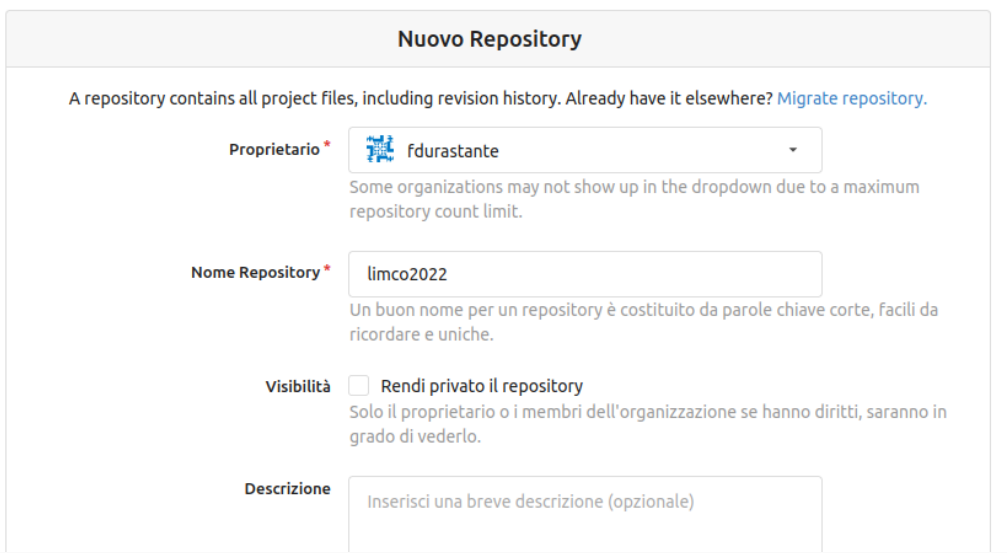




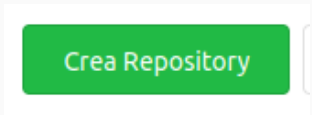


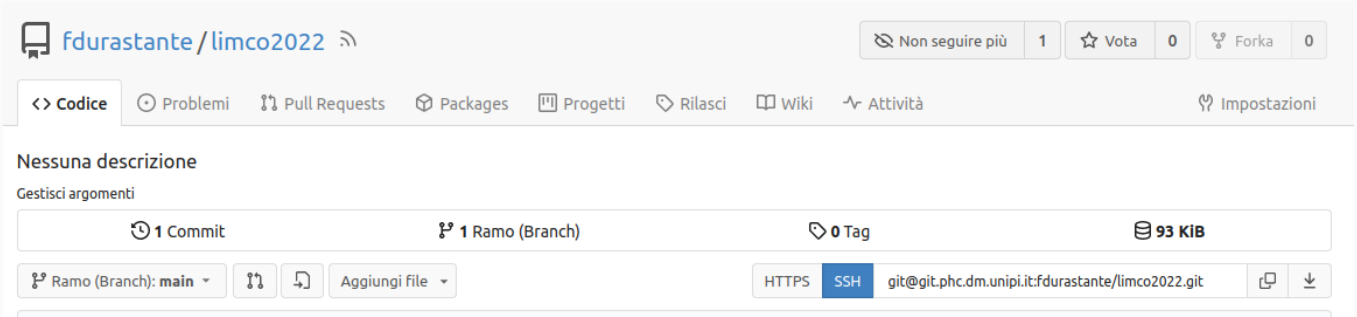
Un servizio in house





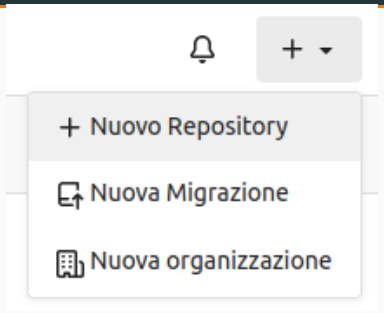
- Anche creare un nuovo repository è analogo a quanto già visto!
- E poi



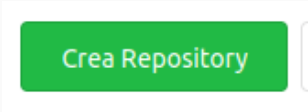



```
git clone git@git.phc.dm.unipi.it:fdurastante/limco2022.git
cd limco2022
```

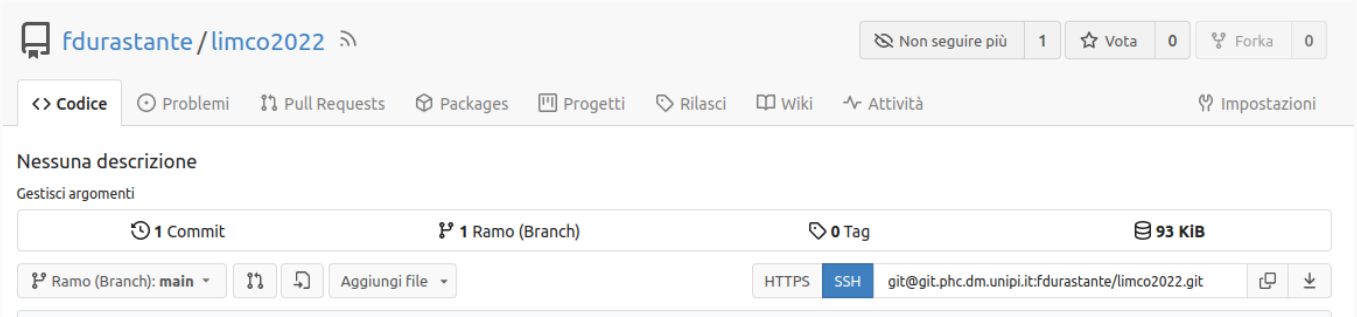
Un servizio in house



- Anche creare un nuovo repository è analogo a quanto già visto!
- E poi




```
cirdan@x580gd:~/Documenti/Didattica/limco$ git clone git@git.phc.dm.unipi.it:fdurastante/limco2022.git
Clone in 'limco2022' in corso...
The authenticity of host 'git.phc.dm.unipi.it (131.114.51.97)' can't be established.
ED25519 key fingerprint is SHA256:LC/3rhzMgQO+CJ1X46at4MzqSfRcZqA9paeGjQMQ4OQ.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'git.phc.dm.unipi.it' (ED25519) to the list of known hosts.
X11 forwarding request failed on channel 0
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Ricezione degli oggetti: 100% (3/3), fatto.
cirdan@x580gd:~/Documenti/Didattica/limco$ ls
limco2022
cirdan@x580gd:~/Documenti/Didattica/limco$ cd limco2022/
cirdan@x580gd:~/Documenti/Didattica/limco/limco2022$ ls
README.md
cirdan@x580gd:~/Documenti/Didattica/limco/limco2022$
```




```
git clone git@git.phc.dm.unipi.it:fdurastante/limco2022.git
cd limco2022
```