

GIT, GitHub e Software Version Control

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.



GRUPPO ISC
Information Sharing Company



MARINA
MILITARE

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.



GRUPPOISC
Information Sharing Company



MARINA
MILITARE

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.

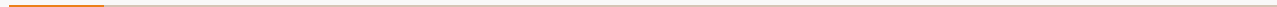


GRUPPO ISC
Information Sharing Company



MARINA
MILITARE

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 sar`adi 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 funzionalita` di gestione del codice sorgente di Git, oltre ad alcune altre funzionalit`a proprietarie. Ha sede in California, ed `è una **filiale di Microsoft dal 2018**.



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 sar`adi usare un servizio terzo, gratuito che ci offra questo servizio:

<https://github.com/>



ATLASSIAN



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



GRUPPO ISC
Information Sharing Company

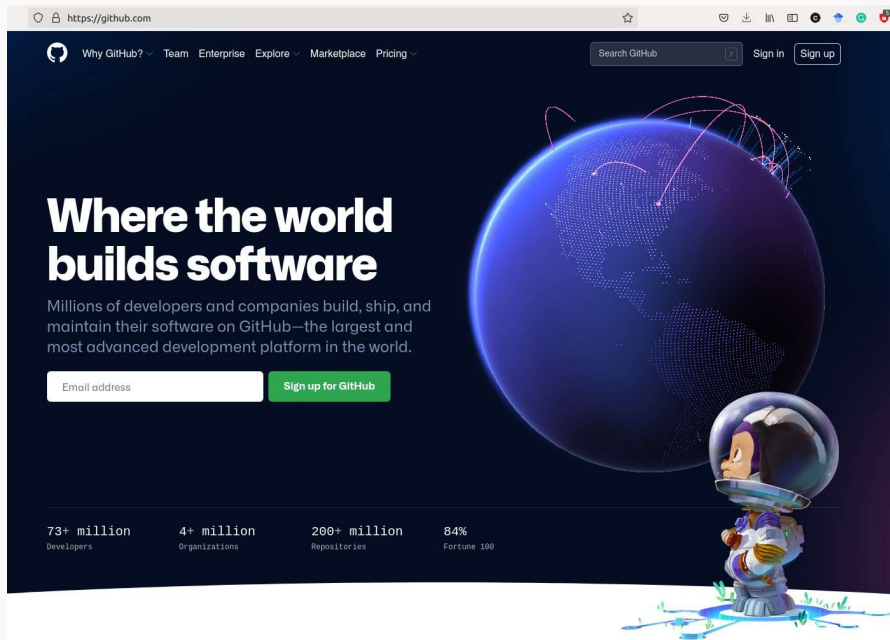


MARINA
MILITARE

Registrarsi su GitHub

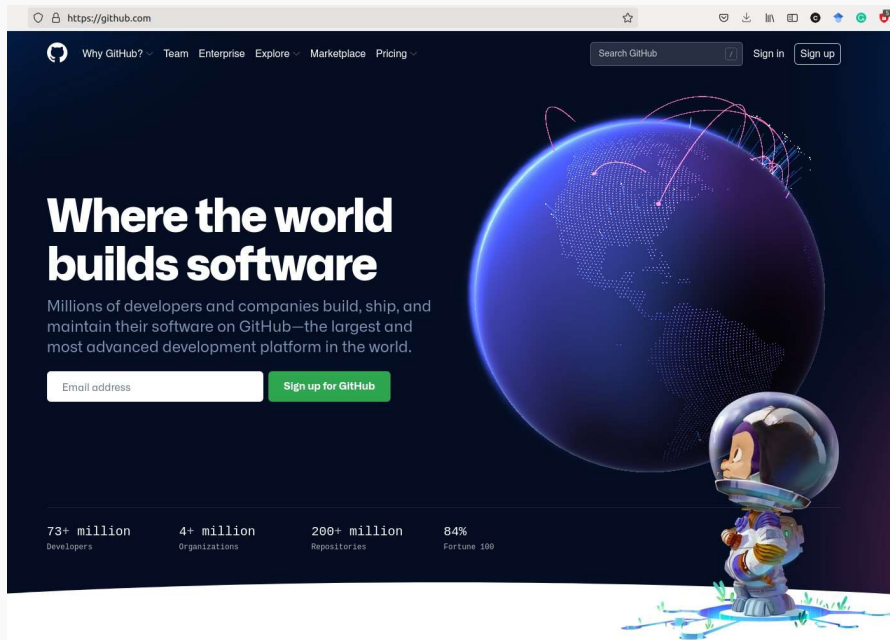
La prima operazione da compiere `quella di **registrarsi su GitHub**.

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



Registrarsi su GitHub

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



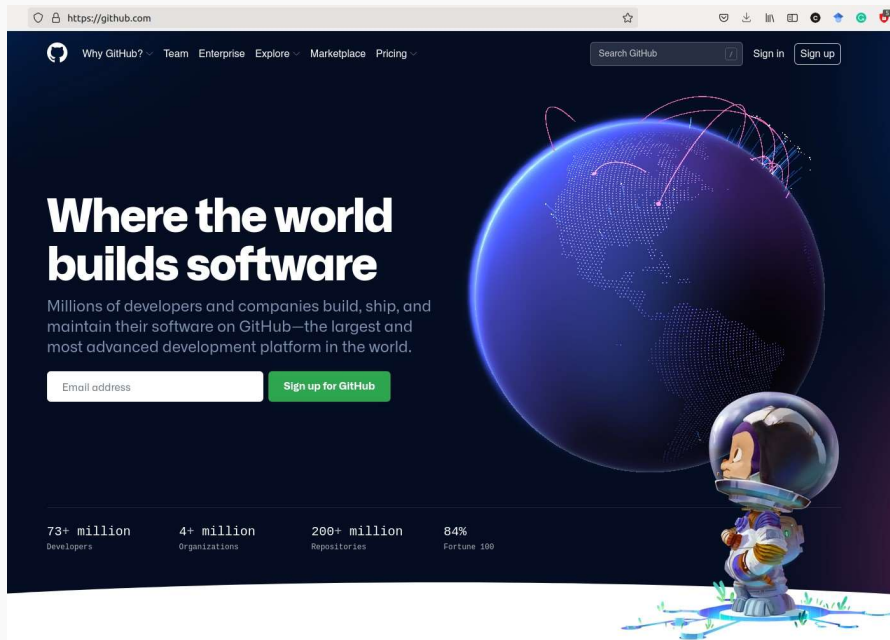
1. Inserite un indirizzo E-Mail (e.g, n.cognome@gruppoisc.it) e fate click sul bottone:

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 `quella di **registrarsi su GitHub**.



1. Inserite un indirizzo E-Mail (e.g, n.cognome@gruppoisc.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.
MARINA MILITARE

Registrarsi su GitHub

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

4. Segnate n per le mail pubblicitarie,

1. Inserite un indirizzo E-Mail (e.g, n.cognome@gruppoisc.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**.

4. Segnate n per le mail pubblicitarie,
5. Confermate e risolvete il problema per farvi riconoscere come esseri umani.

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

A green rectangular button with the text "Sign up for GitHub" in white.


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**.

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.

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

A green rectangular button with the text "Sign up for GitHub" in white.

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**.

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.

1. Inserite un indirizzo E-Mail (e.g, n.cognome@gruppoisc.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



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

Create your first project

Ready to start building? Create a repository for a new idea or bring over an existing repository to keep contributing to it.

[Create repository](#) [Import repository](#)

Recent activity

When you take actions across GitHub, we'll provide links to that activity here.

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

[Read the guide](#) [Start a project](#)

Universe 2021

Missed any Universe sessions? All content is available on demand now so you can view on your own time.

[Watch the sessions on demand](#)

All activity

Introduce yourself

The easiest way to introduce yourself on GitHub is by creating a README in a repository about you! You can start here:

limco2021 / README.md

```
1 - 🌟 Hi, I'm @limco2021
2 - .. I'm interested in ...
3 - 🌱 I'm currently learning ...
4 - ❤️ I'm looking to collaborate on ...
5 - 📫 How to reach me ...
6
```

[Dismiss this](#) [Continue](#)


Discover interesting projects and people to populate your personal news feed.

Your news feed helps you keep up with recent activity on repositories you [watch](#) or [star](#) and people you [follow](#).

[Explore GitHub](#)

🔔 ProTip! The feed shows you events from people you [follow](#) and repositories you [watch](#) or [star](#).

🔔 Subscribe to your news feed

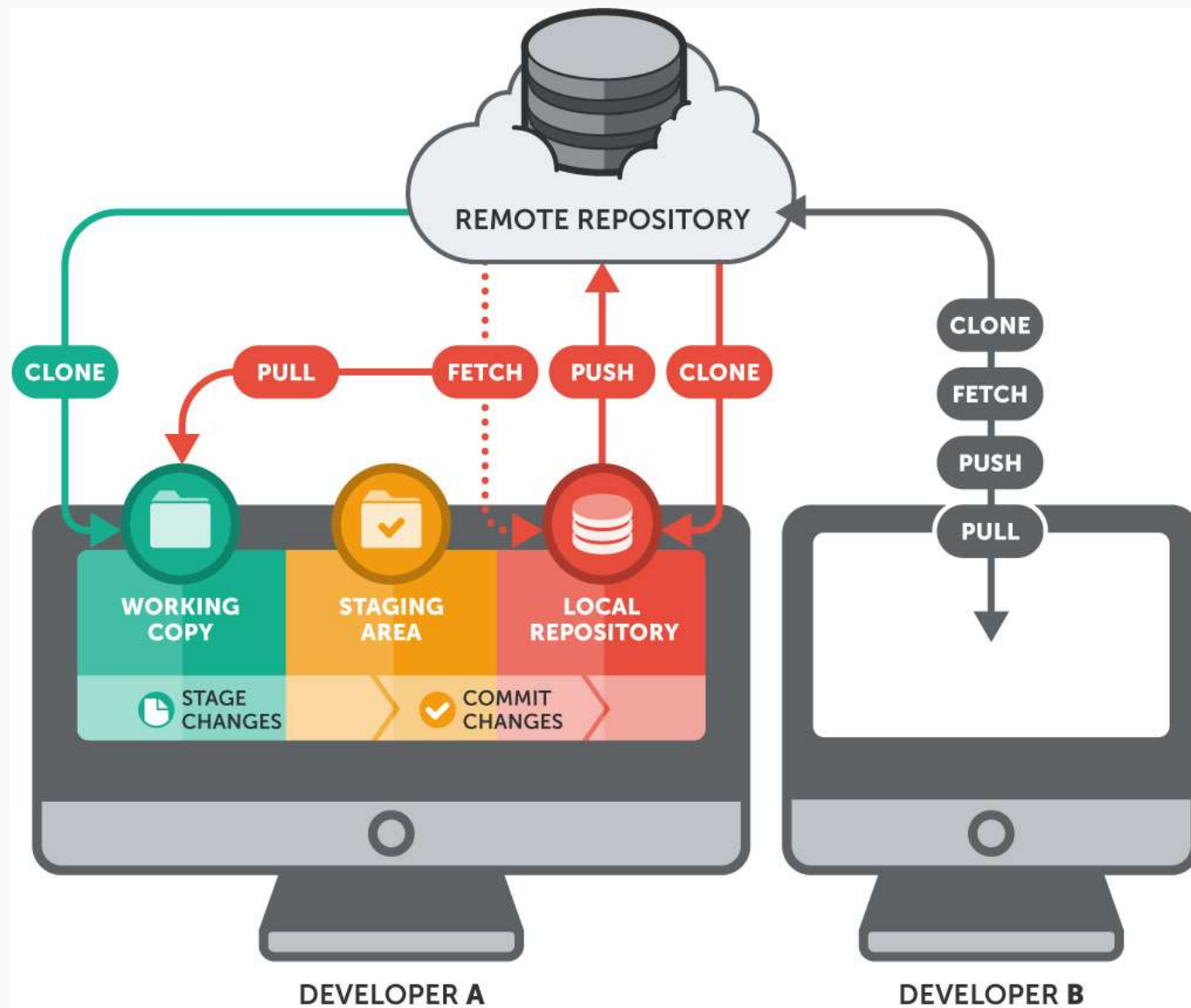
 © 2021 GitHub, Inc.

[Blog](#)
[About](#)

[API](#)
[Training](#)

[Terms](#)
[Privacy](#)

SCHEMA GIT



Hello-world repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

 limco2021 ▾

Repository name *

/ hello-world ✓

Great repository names are short and memorable. Need inspiration? How about [redesigned-bassoon?](#)

Description (optional)

Il mio primo repository



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☒ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

☐ Choose a license

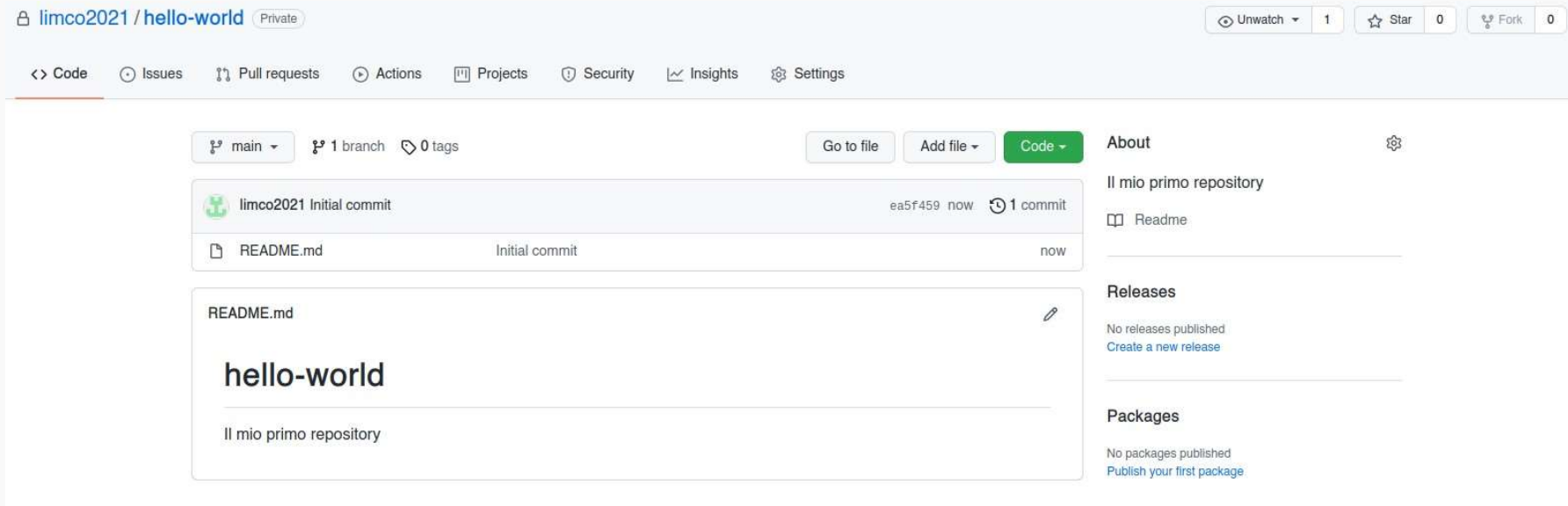
A license tells others what they can and can't do with your code. [Learn more.](#)

This will set  main as the default branch. Change the default name in your [settings](#).

Create repository

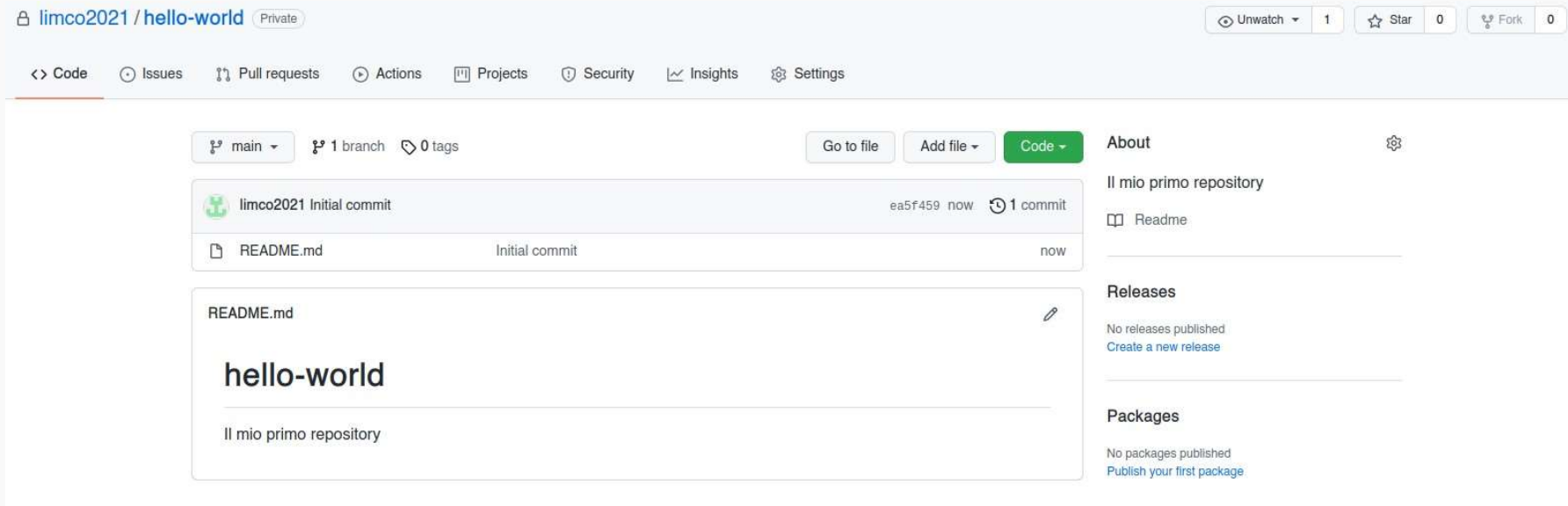
Hello-world repository

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**.

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:



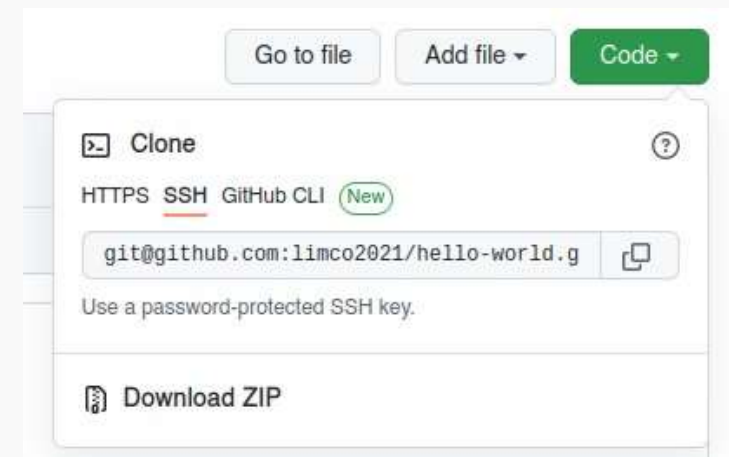
Clone

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

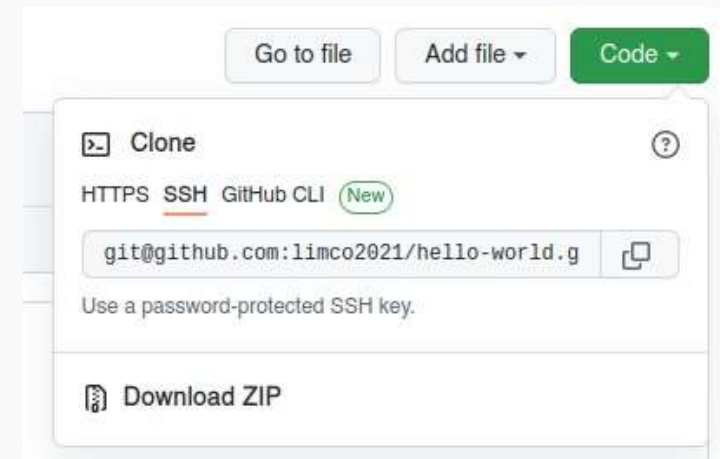
`utente@mathsgalore<-2-3-4>.gruppoisc.it:mygitserver/hello-world.git`



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.
```


Se eseguiamo il comando `ls` osserviamo di aver creato una cartella di nome `hello-world`, o, più 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:

```
GNU nano 2.9.3          README.md          Modified
# hello-world
Il mio primo repository è un **repository** di prova per il corso di *Laboratorio di Introduzione
alla matematica computazionale*.
```

^G Get Help	^O Write Out	^W Where Is	^K Cut Text	^J Justify	^C Cur Pos	M-U Undo
^X Exit	^R Read File	^I Replace	^U Uncut Text	^T To Spell	^_ Go To Line	M-E Redo

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
```

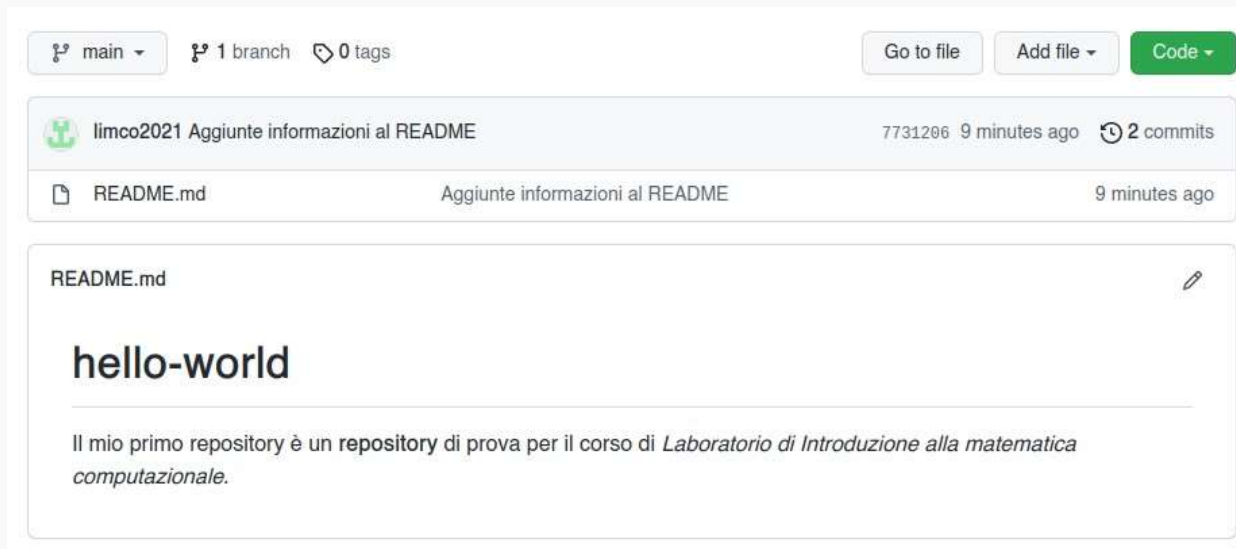


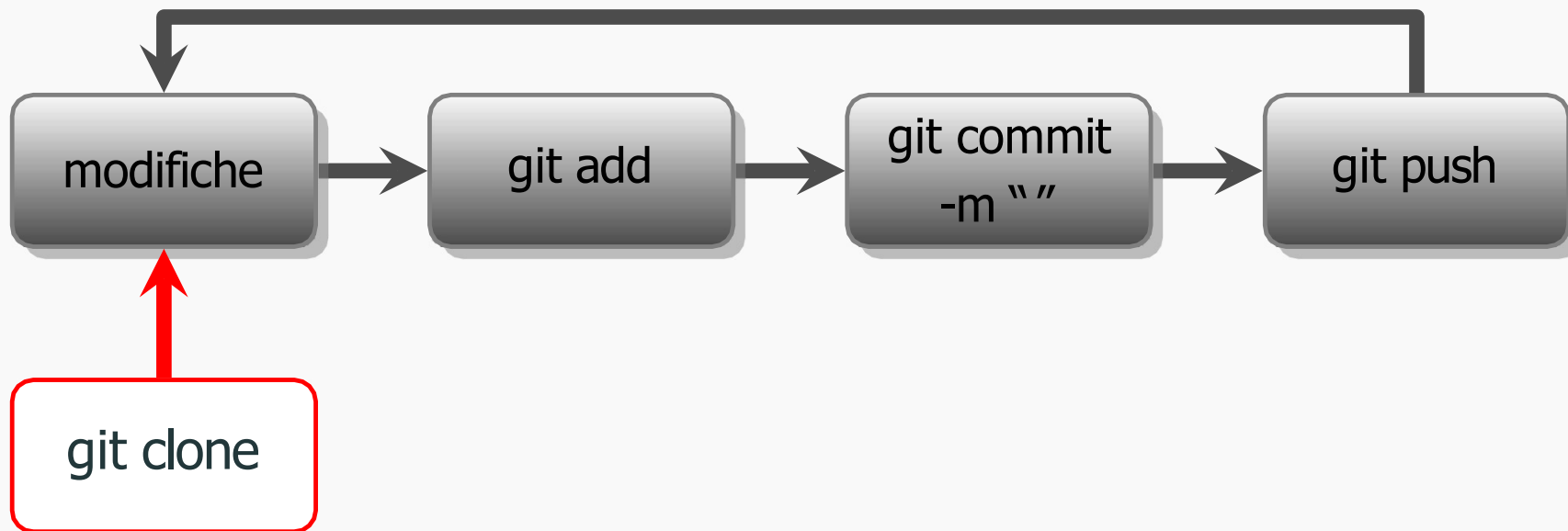
git push

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
```

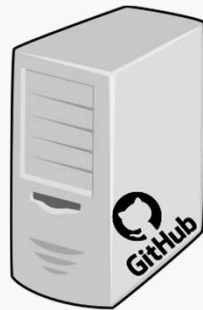
Così facendo abbiamo **ripristinato la sincronia tra locale e remoto!**
Torniamo a vedere cosa cambia sull'interfaccia web:





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, L^AT_EX, pagine web),
4. Preparazione di un *commit* (`git add ...`, `git commit -m " "`),
5. Sincronizzare il remoto (`git push`).

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



hello-world
f.durastante



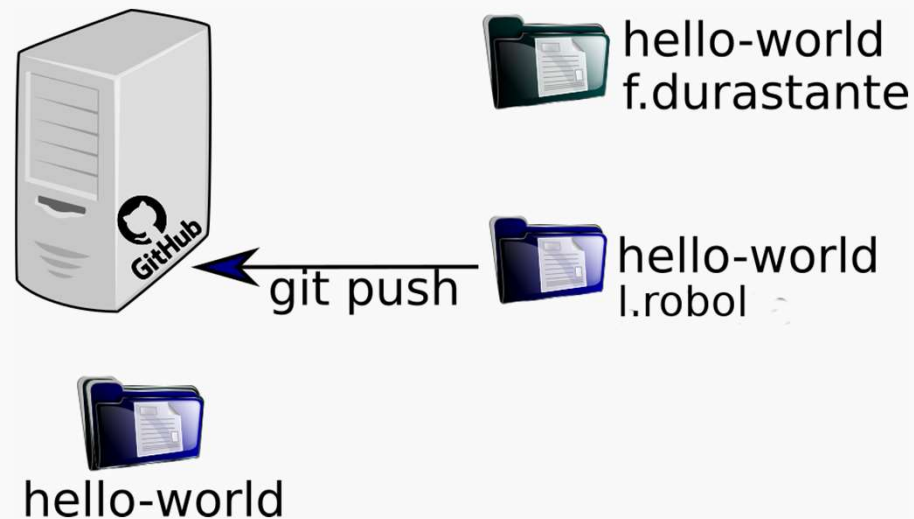
hello-world
l.robol



hello-world

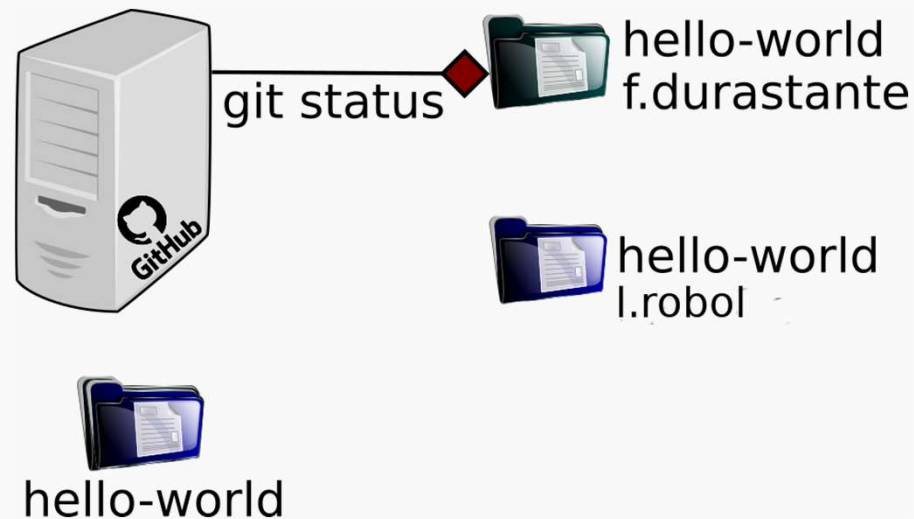
L'utente l.robol ha fatto una sua modifica in locale, f.durastante e l'online non ne sanno nulla.

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



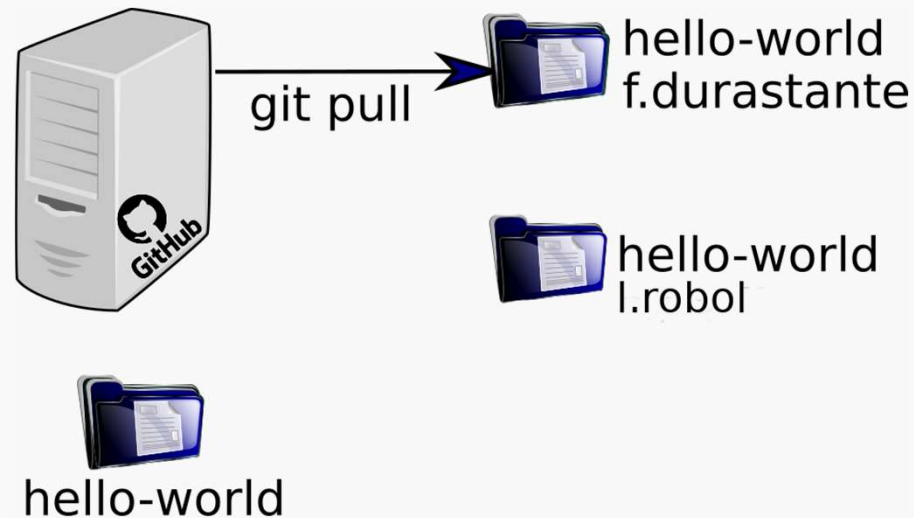
l.robol è soddisfatto dei suoi cambi, fa `git add`, `git commit` e `git push`: ora la versione sul *repository* è stata aggiornata.

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.

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.

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



hello-world
f.durastante



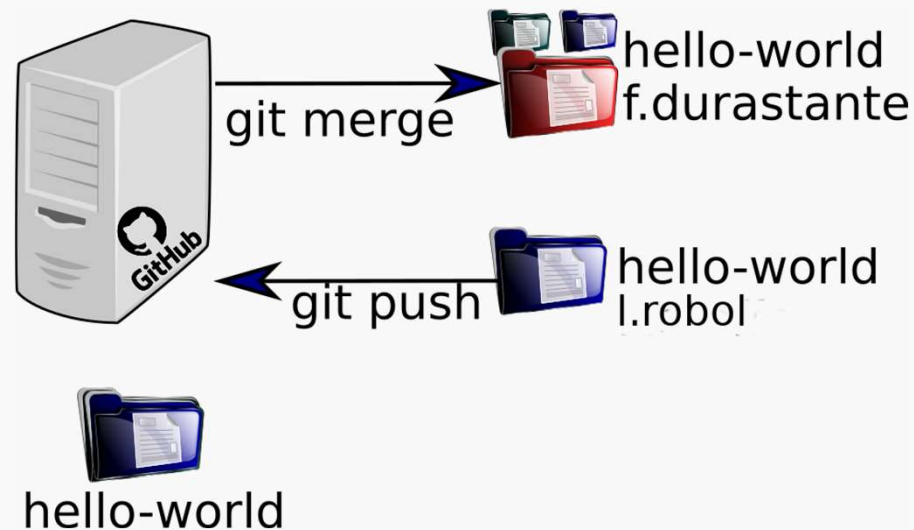
hello-world
l.robol



hello-world

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

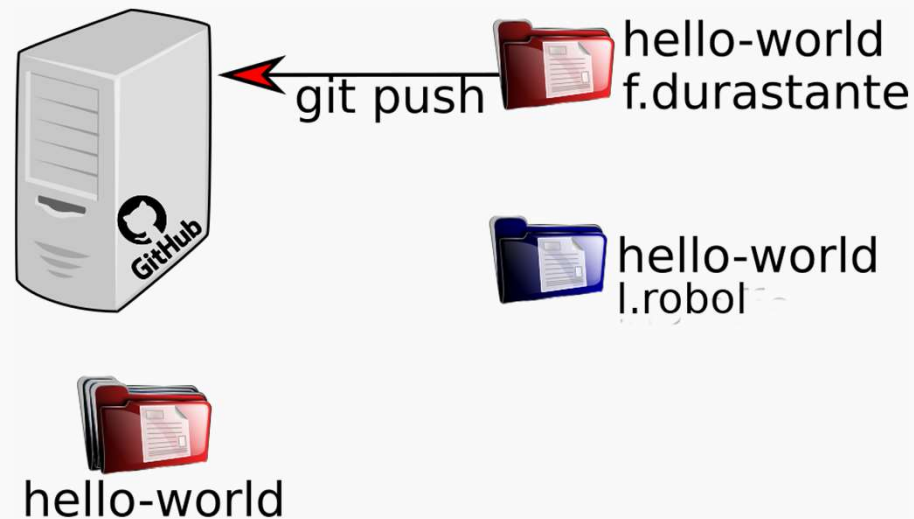
Immaginiamo ora di avere due (o più collaboratori) o macchine con cui lavoriamo sullo stesso repository.



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.

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

```
q17, 61 q = v / h[k, k-1] # Add the produced vector to
eps = eta*nb # If v is shorter than this threshold we
# To avoid uninitialized k message on output
for k in np.arange(1, n+1):
    v = a.dot(q) # Generate a new candidate vector
    for j in np.arange(k): # Subtract the projections
        h[j, k-1] = np.dot(Q[:, j].conj(), v)
        v = v - h[j, k-1] * Q[:, j]
    h[k, k-1] = np.linalg.norm(v, 2)
    g = np.zeros((k+1, 1))
    g[0, 0] = nb
    hc = np.append(h[0:k+1, 0:k], g, axis=1)
    hhat = np.linalg.qr(hc)[1]
    kres = np.absolute(hhat[-1, -1])
    if kres <= eps:
        return Q[:, 0:k], h[0:k, 0:k], k
    else:
        q = v / h[k, k-1] # Add the produced vector to
        Q[:, k] = q # the zero vector is produced
return Q[:, 0:n], h[0:n, 0:n], n

def amg_arnoldi_iteration(a, b, eta, n: int, ml):
    """Computes a basis of the (n + 1)-Krylov subspace of
    spanned by {b, Ab, ..., A^n b}.
```

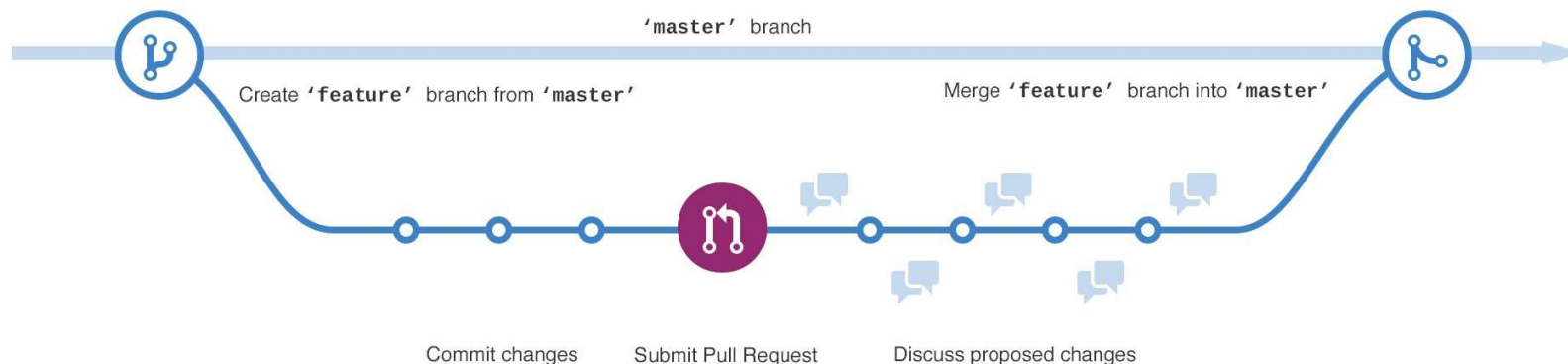
```
q17, 61 q = v / h[k, k-1] # Add the produced vector
eps = min(eta*nb, 1.0E-1) # If v is shorter than this t
# To avoid uninitialized k message on output
for k in np.arange(1, n+1):
    v = a.dot(q) # Generate a new candidate vector
    for j in np.arange(k): # Subtract the projections c
        h[j, k-1] = np.dot(Q[:, j].conj(), v)
        v = v - h[j, k-1] * Q[:, j]
    h[k, k-1] = np.linalg.norm(v, 2)
    g = np.zeros((k+1, 1))
    g[0, 0] = nb
    hc = np.append(h[0:k+1, 0:k], g, axis=1)
    hhat = np.linalg.qr(hc)[1]
    kres = np.absolute(hhat[-1, -1])
    if kres <= eps:
        if h[k, k-1] > 1E-12:
            q = v / h[k, k-1] # Add the produced vector
            Q[:, k] = q # the zero vector is produc
            return Q[:, 0:k+1], h[0:k+1, 0:k], k+1
        else:
            return Q[:, 0:k], h[0:k, 0:k], k
    else:
        q = v / h[k, k-1] # Add the produced vector to
        Q[:, k] = q # the zero vector is produced
return Q[:, 0:n+1], h[0:n+1, 0:n], n+1
```

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 nomedelnuovobranh
```

```
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.

```
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.

```
f.durastante@mathsgalore4:~/hello-world$ nano README.md
f.durastante@mathsgalore4:~/hello-world$ git add README.md
f.durastante@mathsgalore4:~/hello-world$ git commit -m "Aggiunta informazione"
[main 57f2c5c] Aggiunta informazione
 1 file changed, 2 insertions(+)
f.durastante@mathsgalore4:~/hello-world$ git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 327 bytes | 327.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:limco2021/hello-world.git
 7731206..57f2c5c  main -> main
f.durastante@mathsgalore4:~/hello-world$
```

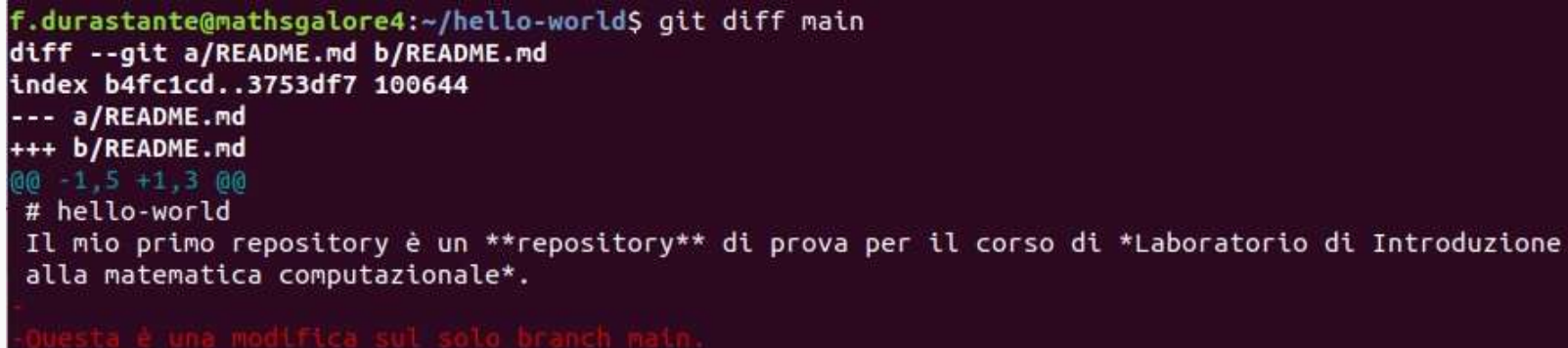


Un esercizio di branching e merging

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

1. 
2. `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`.



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

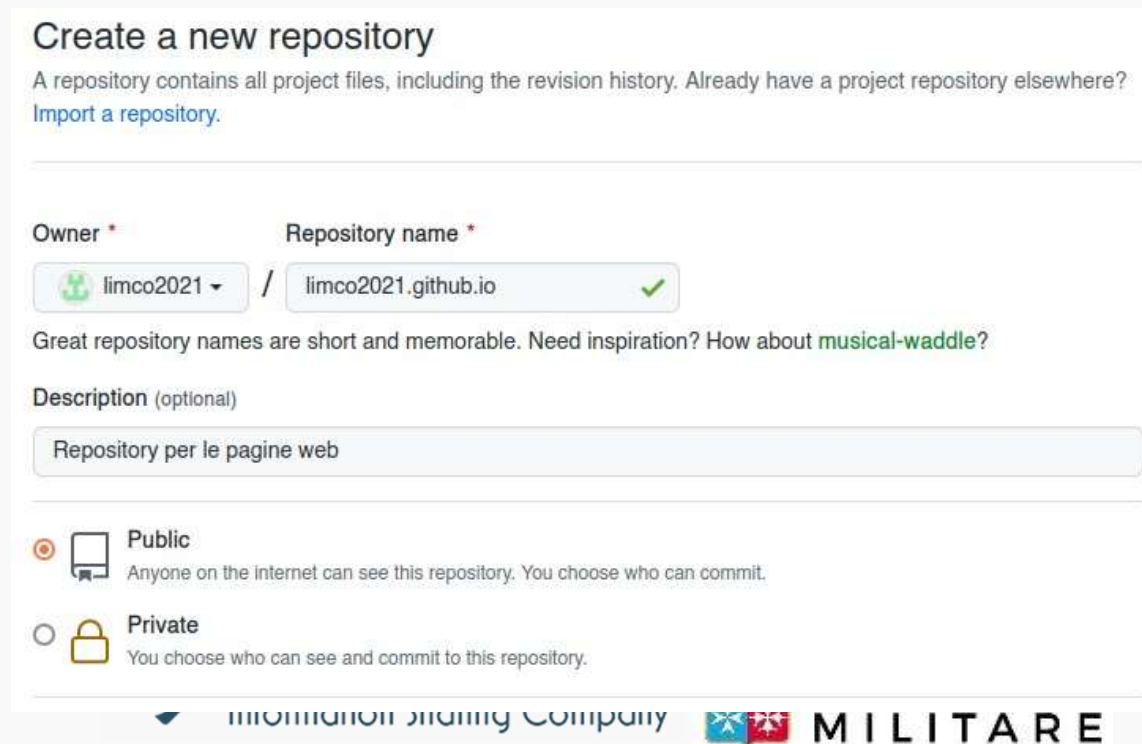
Facciamo i pochi passi necessari ad attivarlo e a *caricare una prima pagina web di prova*.



GitHub 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.



The screenshot shows the 'Create a new repository' page on GitHub. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, there are two input fields: 'Owner' and 'Repository name'. The 'Owner' field is set to 'limco2021' and the 'Repository name' field is set to 'limco2021.github.io', which is marked as valid with a green checkmark. Below these fields, there is a suggestion for repository names: 'Great repository names are short and memorable. Need inspiration? How about musical-waddle?'. There is also a 'Description (optional)' field with the text 'Repository per le pagine web'. At the bottom, there are two radio buttons for repository visibility: 'Public' (selected) and 'Private'. The 'Public' option is described as 'Anyone on the internet can see this repository. You choose who can commit.' and the 'Private' option is described as 'You choose who can see and commit to this repository.'

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner * / Repository name *

limco2021 / limco2021.github.io ✓

Great repository names are short and memorable. Need inspiration? How about [musical-waddle?](#)

Description (optional)

Repository per le pagine web

☒ Public
Anyone on the internet can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Information sharing Company MILITARE

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...



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.
```



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
```

```
f.durastante@mathsgalore4:~$ cd limco2021.github.io/  
f.durastante@mathsgalore4:~/limco2021.github.io$ echo "Hello world!" > index.html
```



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:

