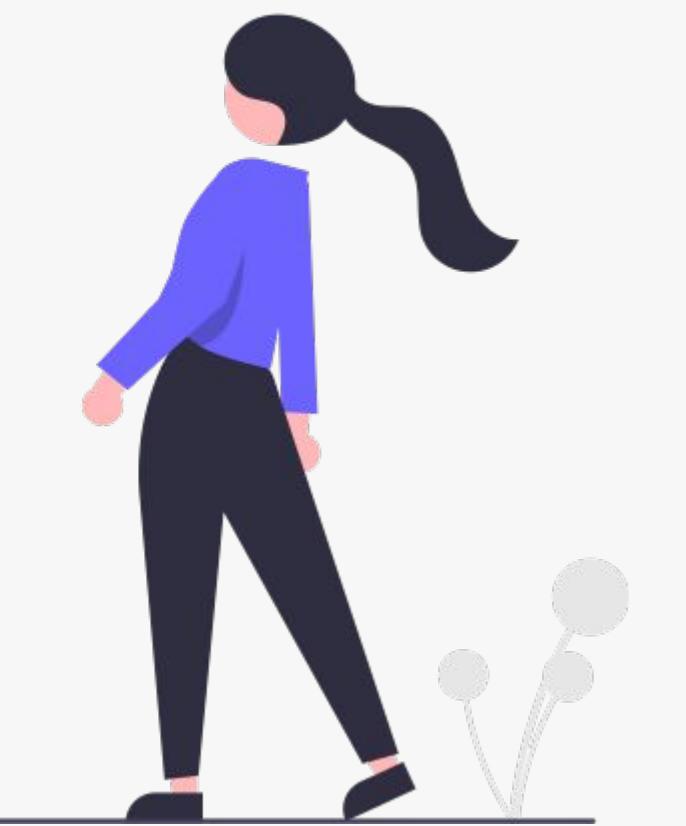
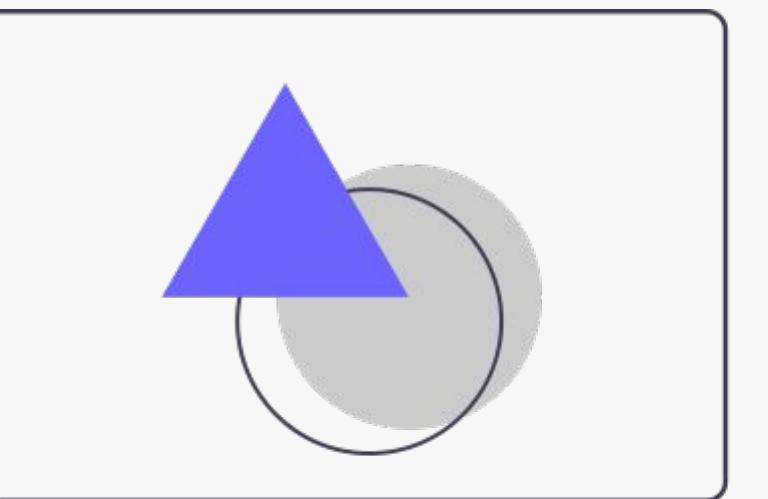


Git e Github

START HERE

Introdução Git e Github

- Entendendo o que é git e o github
- Configurando o Git
- Essencial do Git
- Repositórios remotos
- Ramificações (Branches)
- Outros comandos interessantes

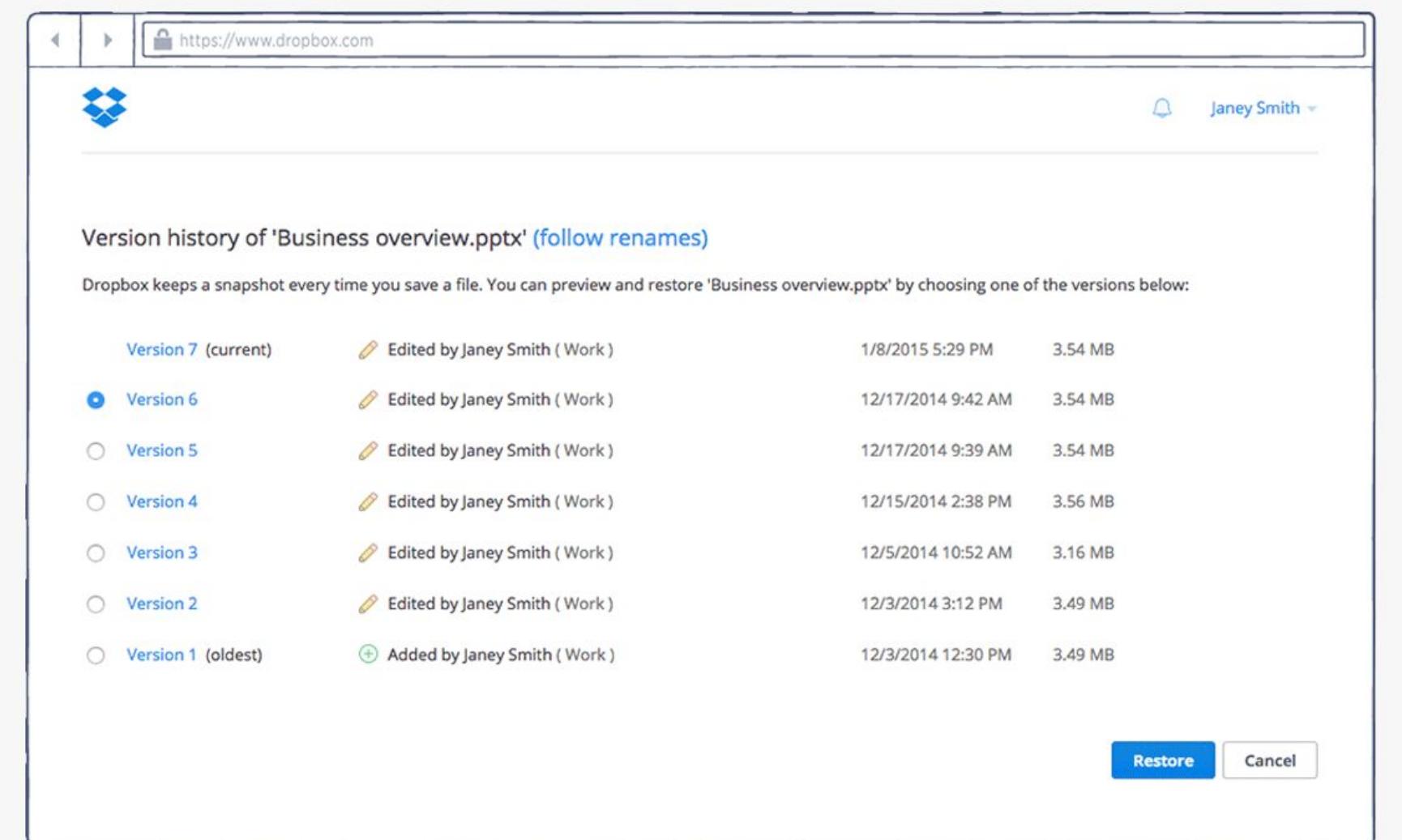


Git

Controle de Versão

É um sistema que registra as alterações de qualquer tipo de arquivo para que você possa lembrar de versões específicas mais tarde.

Usar um controle de versão, significa se você um dia estragar tudo ou perder arquivos, você poderá facilmente recuperar.

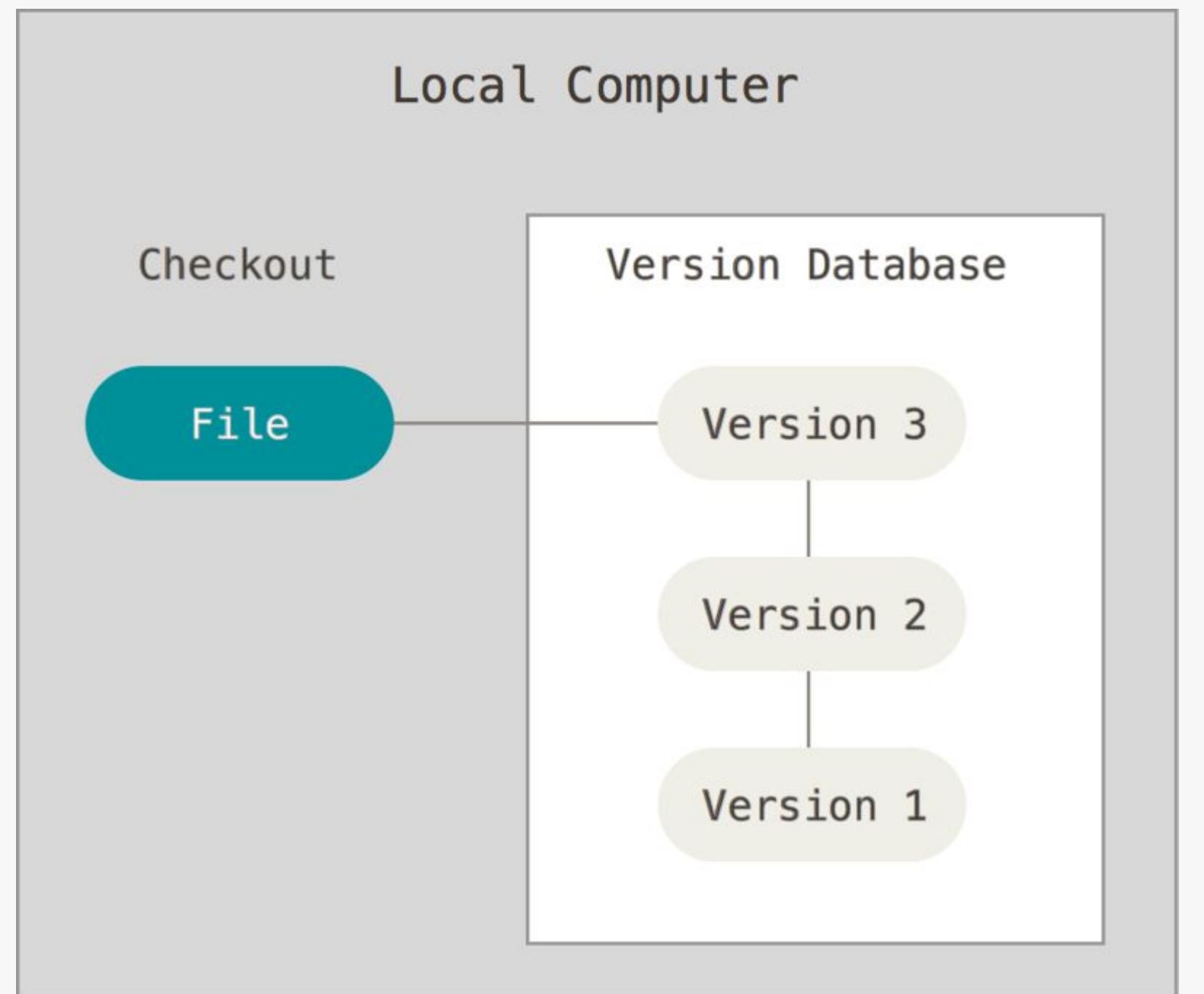


Git

Controle de Versão

O método de muitas pessoas é copiar arquivos para outro diretório, talvez nomeando inteligentemente com a data e hora, ou apenas criando cópias. Esta abordagem é muito comum porque é muito simples, porém é muito propensa a erros.

Muito facilmente você esquecer qual diretório está ou que versão você está mexendo agora e acabar acidentalmente sobreescrevendo os arquivos erroneamente.

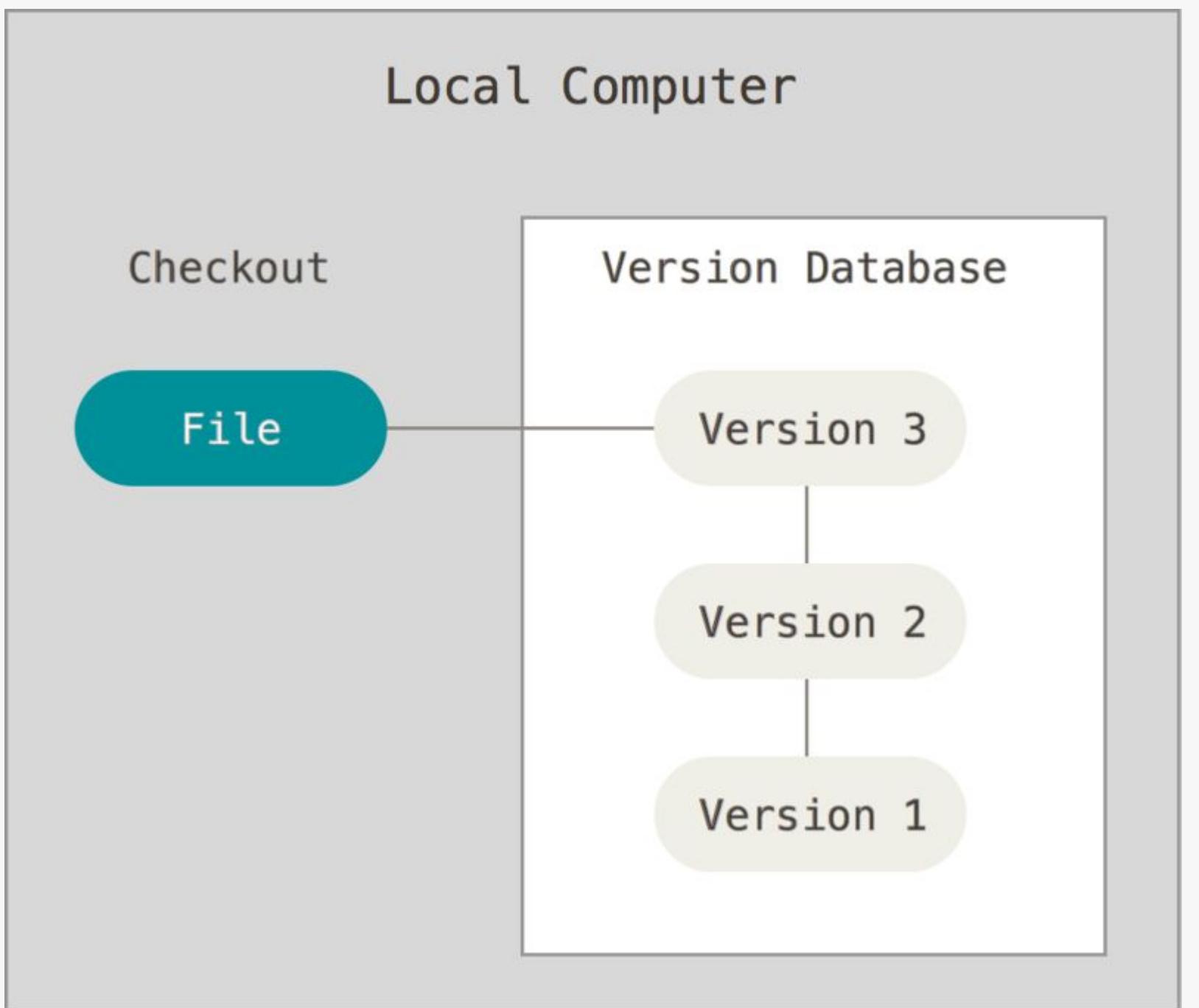


Git

Controle de Versão

Sendo assim, muitos programadores desenvolveram sistemas para armazenamento das versões dos arquivos em um banco de dados local, onde mantém todas as alterações dos arquivos.

E se o hd queimar? e se o banco de dados for adulterado? e caso tenhamos mais de um colaborador?

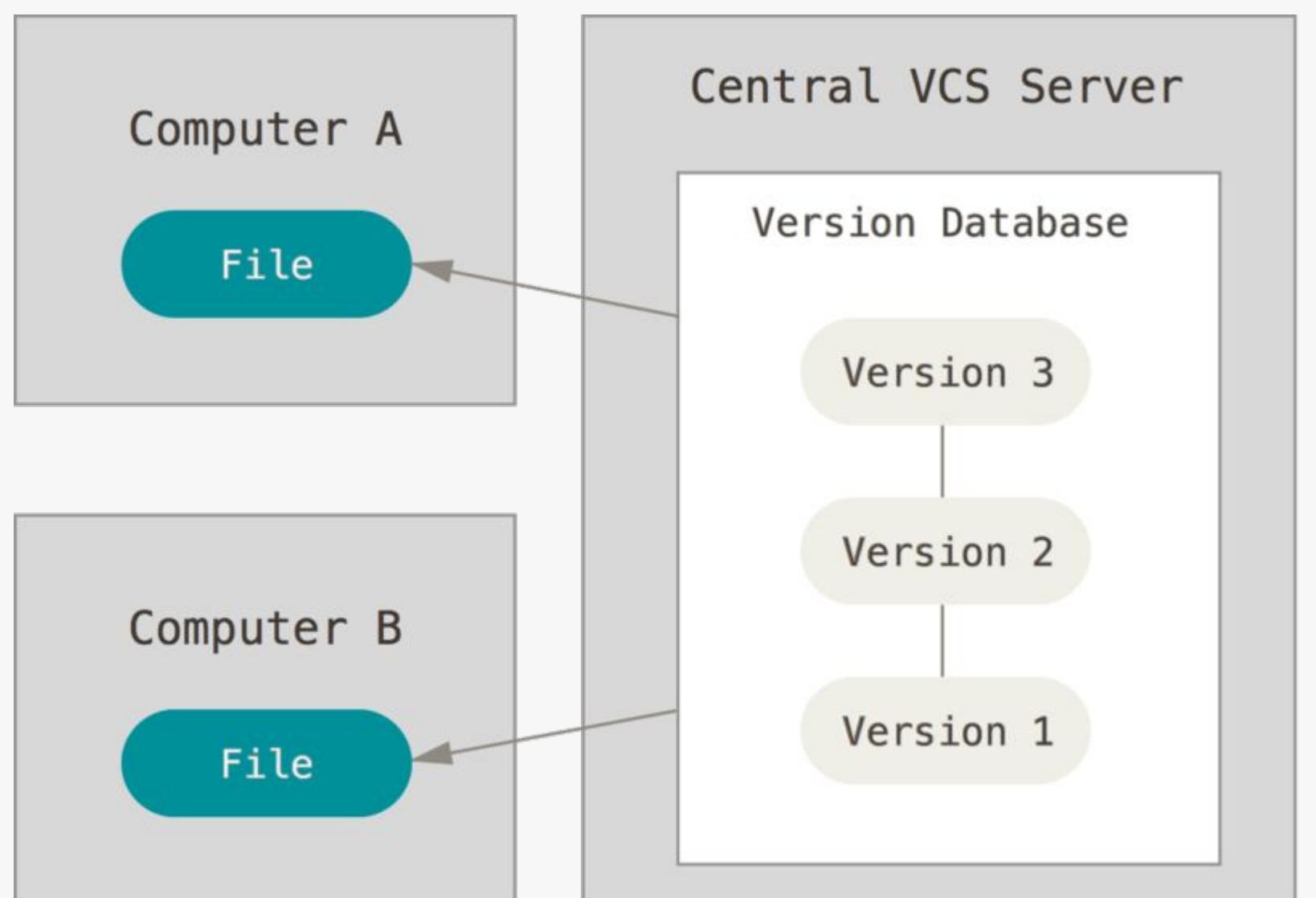


Git

Controle de Versão

Tendo em vista a necessidade de colaboração entre diversos desenvolvedores no mesmo projeto, surgem os **Sistemas Centralizados de Controle de Versão**.

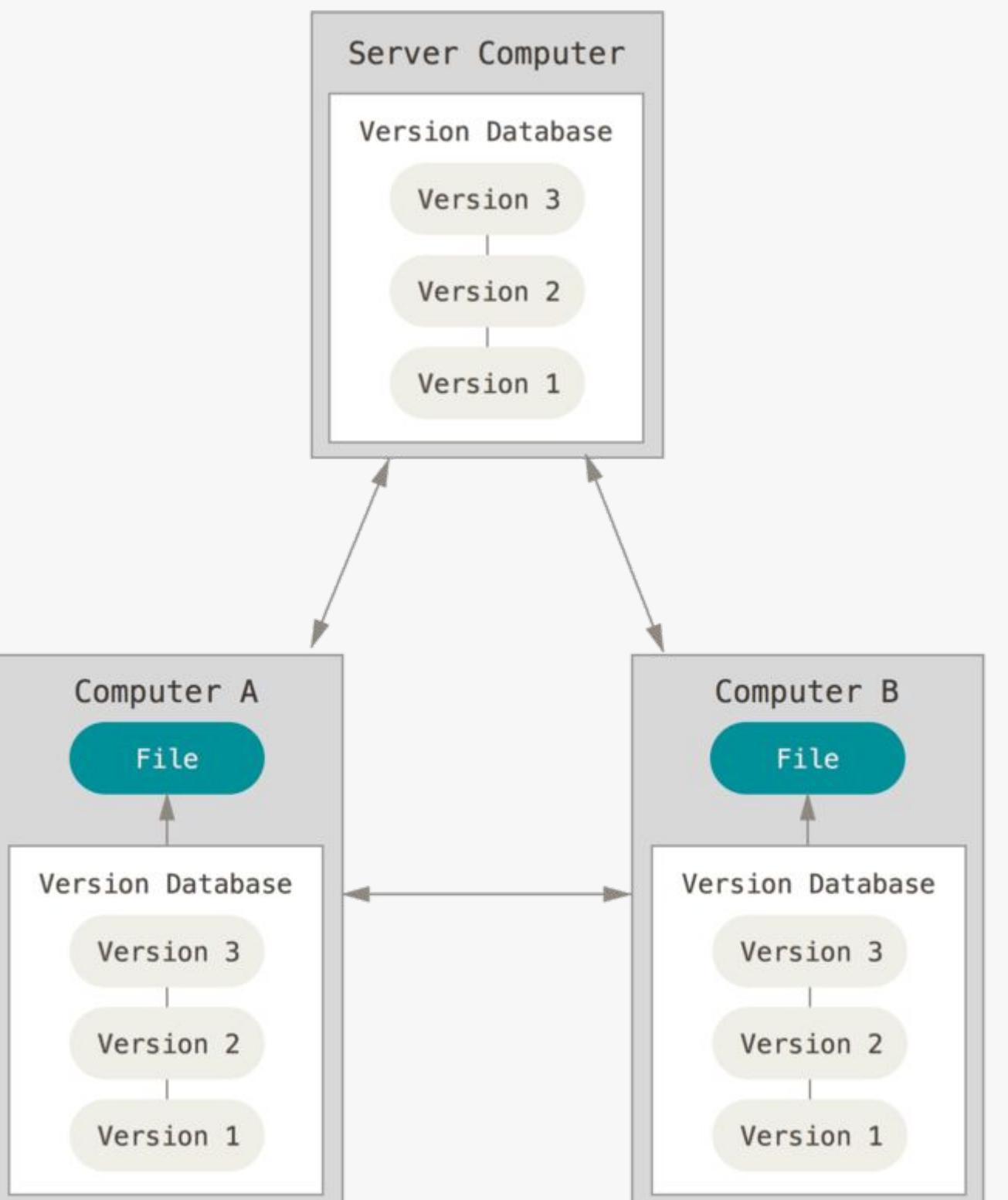
Um dos grandes problemas desse sistema seria o próprio servidor. Se ele ficar indisponível por 1 hora, todos os colaboradores não conseguiram trabalhar nos arquivos durante este tempo. Outro ponto gravíssimo é que se o disco rígido do banco de dados central for corrompido e backups não forem apropriadamente mantidos, é perdido absolutamente tudo.



Git

Controle de Versão

É assim então, que os **Sistemas Distribuídos de Controle de Versão** surgem (**Git, bitkeeper**). Em um sistema distribuído, os clientes não apenas usam o estado mais recente de arquivos, mas também, **duplicam** o repositório localmente. Assim, se qualquer servidor morrer e esses sistemas estiverem colaborando por meio dele, qualquer um dos repositórios de clientes pode ser copiado de volta para o servidor e restaurá-lo. Cada **clone** é de fato um backup completo de todos os dados.

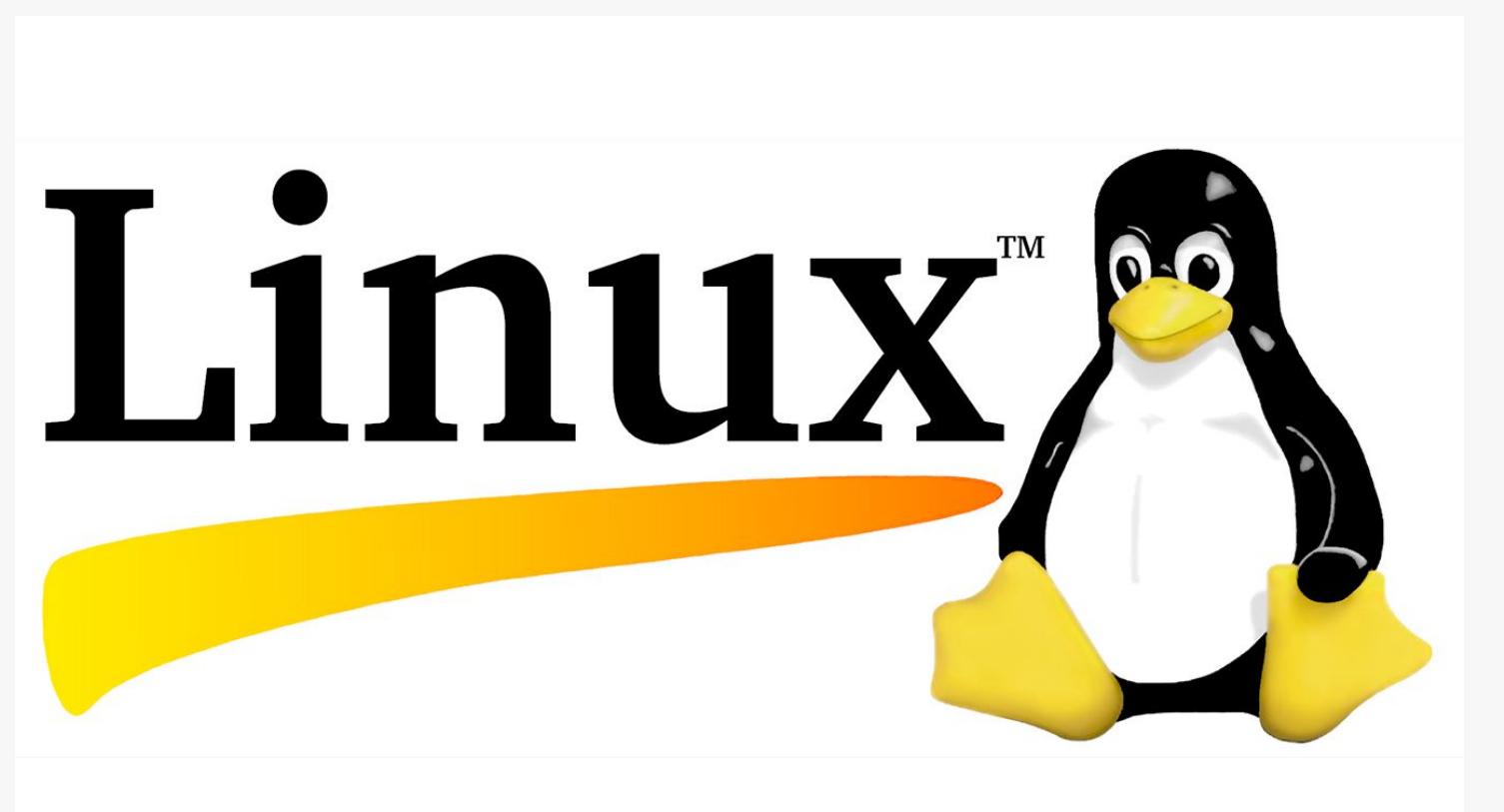


Git

História do Git

O Git surgiu através de uma necessidade do criador do Linux, o **Linus torvalds**. Este homem, criador do kernel do Linux, recebeu uma chamada da empresa **BitKeeper**, ferramenta onde utilizava como controle de versão distribuída para o Linux desde 2002. A chamada era de que a ferramenta iria ser cobrada a partir de então.

Com isso, o que o Linus Torvalds fez?



Git

História do Git

Sim, ele criou seu próprio controle de versão distribuído chamado **Git**!

E ainda foi além, resolveu problemas que a **BitKeeper** tinha na época, trazendo:

- Velocidade
- Simplicidade
- Forte suporte para desenvolvimento não-linear (milhares de ramos paralelos)
- Completamente distribuído
- Capaz de lidar com projetos grandes como o kernel do Linux com eficiência (velocidade e tamanho dos dados)



Git

História do Git

- Sim, ele criou seu próprio controle de versão distribuído chamado **Git**!
- E ainda foi além, resolveu problemas que a **BitKeeper** tinha na época, trazendo:
- Velocidade
 - Simplicidade
 - Forte suporte para desenvolvimento não-linear (milhares de ramos paralelos)
 - Completamente distribuído
 - Capaz de lidar com projetos grandes como o kernel do Linux com eficiência (velocidade e tamanho dos dados)

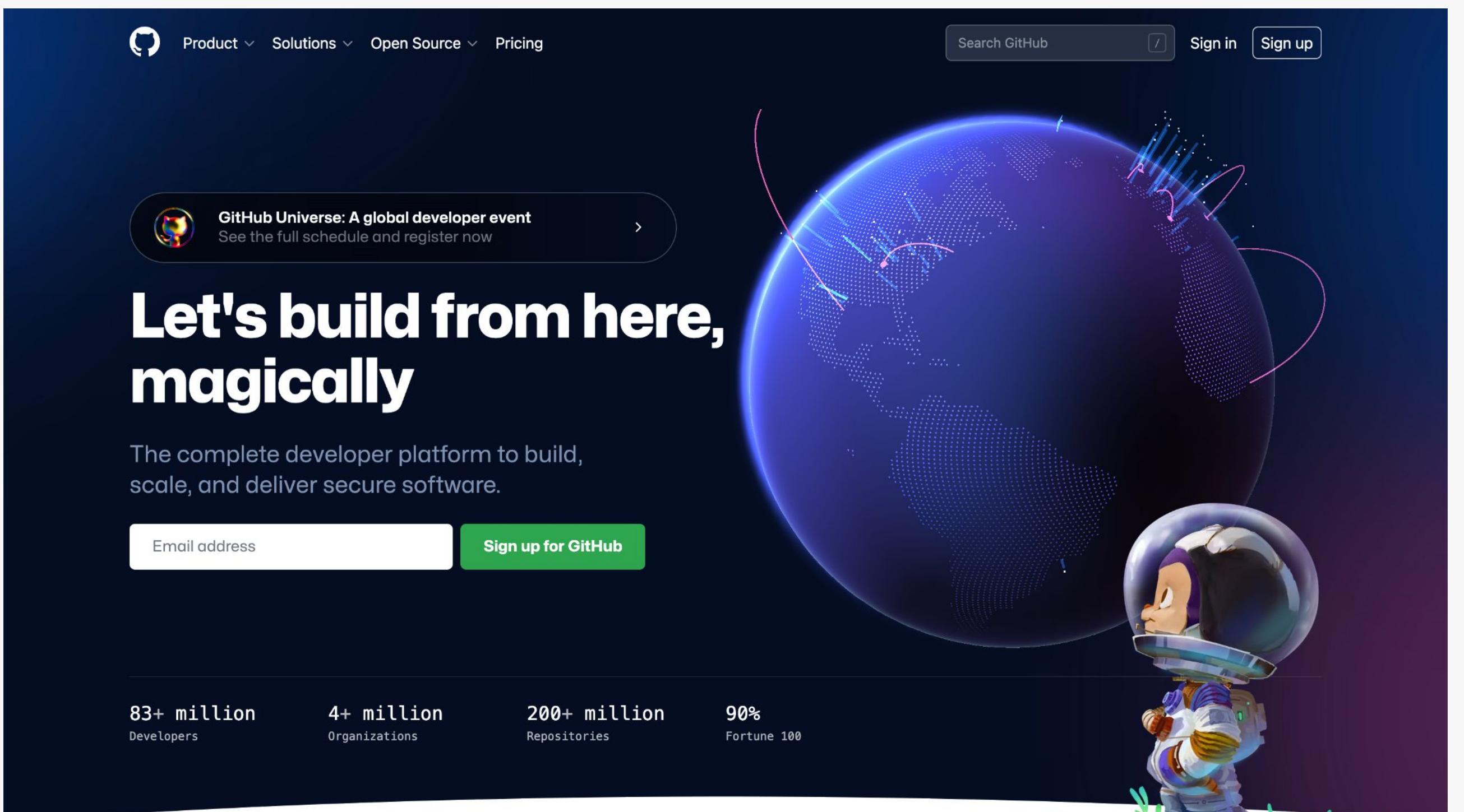


Github

O que é Github

É um serviço web compartilhado para projetos que utilizam **Git** para o seu versionamento.

Ou seja, o **Github** não é o **Git**, **Git** é o sistema de controle de versão distribuído e o **Github** são os seus arquivos na web, é o meio em que torna o seu repositório remoto.



Git Instalando e Configurando

Então chega de história e vamos botar a mão na massa!!

MacOS:

```
$ brew install git
```

Windows:

<https://github.com/git-for-windows/git/releases/download/v2.38.1.windows.1/Git-2.38.1-64-bit.exe>

Ubuntu:

```
# apt-get install git
```

The screenshot shows the official Git website at git-scm.com. The main navigation bar includes links for About, Documentation, Downloads (which is highlighted in red), and Community. The Downloads section features a large image of a Mac monitor displaying the latest source release version 2.38.1. Below the monitor, there are download links for macOS, Windows, and Linux/Unix. A note indicates that older releases are available on GitHub. The page also includes sections for GUI Clients, Logos, and a link to get the latest development version via Git itself.

<https://git-scm.com/downloads>

Git

Instalando e Configurando

O Git ele armazena suas informações em 3 lugares:

1. git config do sistema como um todo
2. git config do usuário
3. git config do projeto específico

```
$ git config --global user.name "Fábio S Takaki"
```

```
$ git config --global user.email "fabio.stakaki@gmail.com"
```



Git

Iniciando um repositório

```
[→ github mkdir project-1
[→ github cd project-1
[→ project-1 git init
Initialized empty Git repository in /Users/fabiotakaki/Documents/Rubcube/courses/github/project-1/.git/
→ project-1 git:(main) cd .git
[→ .git git:(main) ls
[HEAD      description  info      refs
config     hooks        objects
→ .git git:(main)
```

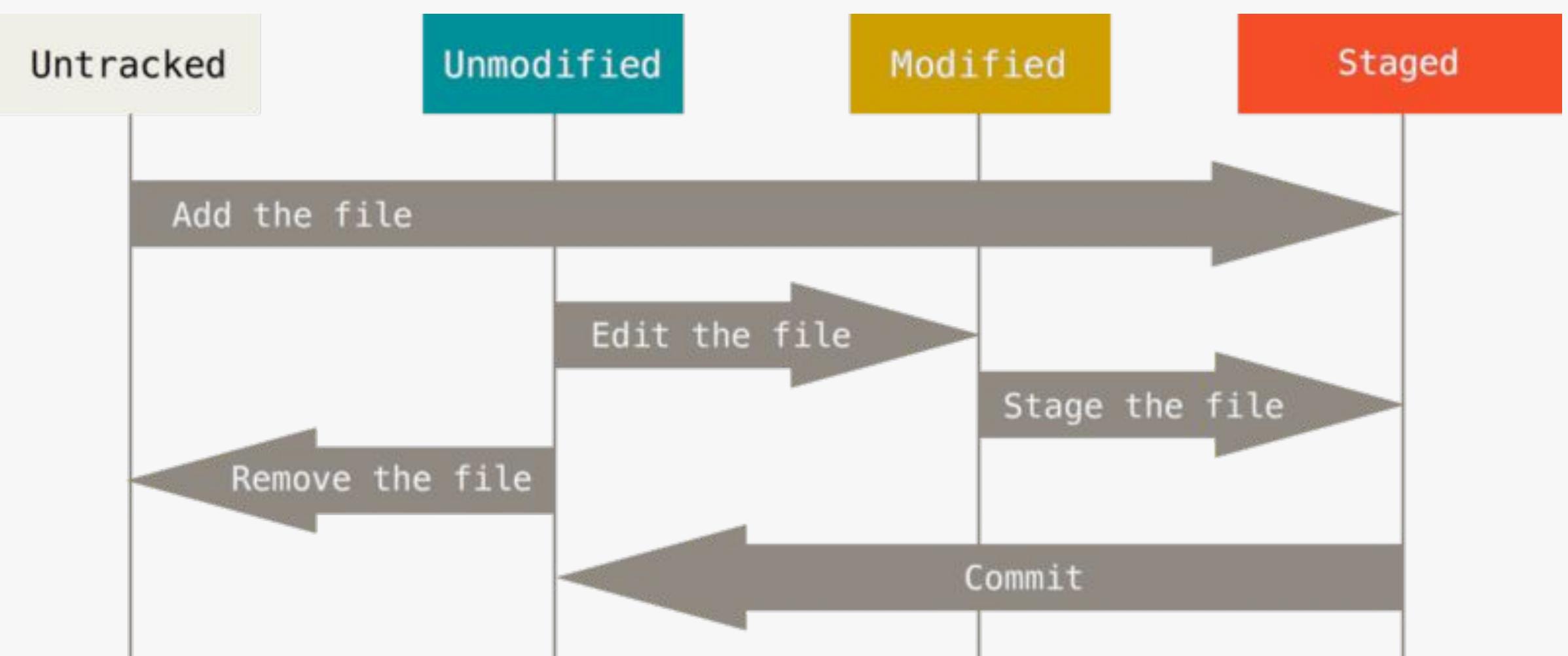
```
$ mkdir project-1
$ cd project-1
$ git init
```

Git

Ciclo de vida

Temos 4 estados dos arquivos que compõe o repositório Git:

1. **Untracked**: quando o arquivo acabou de ser adicionado/modificado no seu repositório mas ainda não foi visto pelo git.
2. **Unmodified**: ao ser adicionado no git, ele passa ser considerado unmodified. Ou seja, ele existe no git mas não teve nenhuma modificação



Git

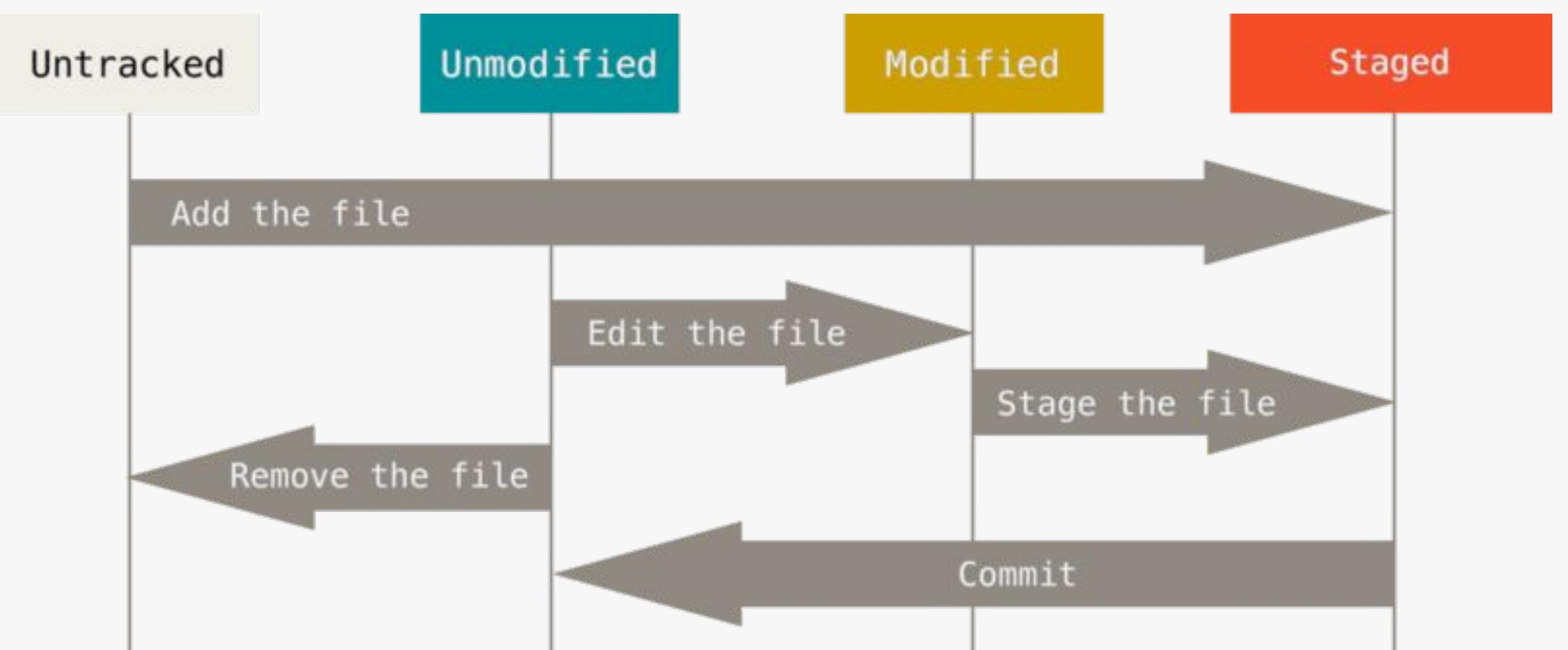
Ciclo de vida

Temos 4 estados dos arquivos que compõe o repositório Git:

3. **Modified**: se o arquivo foi modificado e já é conhecido pelo git, ele vira **modified**

4. **Staged**: depois de modificado, é possível transportar o arquivo em **staged** que ele já está pronto para ser fechado.

Ao ser **commitado** os arquivos que estão em staged, eles voltam ser unmodified.



Git

Ciclo de vida

Vamos entender na prática!

```
$ git status
```

```
$ echo "# Curso de Github" >>  
README.md
```

```
$ git status
```

```
$ git add README.md
```

```
$ git status
```

```
$ git commit -m "first commit"
```

```
$ git status
```

```
[→ .git git:(main) cd ..  
[→ project-1 git:(main) git status  
On branch main  
  
No commits yet  
  
nothing to commit (create/copy files and use "git add" to track)  
[→ project-1 git:(main) echo "# Curso de Github" >> README.md  
[→ project-1 git:(main) x git status  
On branch main  
  
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)  
[→ project-1 git:(main) x git add README.md  
[→ project-1 git:(main) x git status  
On branch main  
  
No commits yet  
  
Changes to be committed:  
  (use "git rm --cached <file>..." to unstage)  
    new file:   README.md  
  
[→ project-1 git:(main) x git commit -m "first commit"  
[main (root-commit) c072d01] first commit  
  1 file changed, 1 insertion(+)  
  create mode 100644 README.md  
[→ project-1 git:(main) git status  
On branch main  
nothing to commit, working tree clean  
→ project-1 git:(main)
```

Git Logs

Vamos entender na prática!

```
$ git log
```

```
$ git log --author="Fabio"
```

```
$ git shortlog
```

```
$ git show <hash>
```

```
commit c072d01ecd7a2f19863fe92ed4686e7127be5af1 (HEAD -> main)
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>
Date:   Sun Oct 23 16:53:56 2022 -0300

    first commit
(END)
```

```
commit c072d01ecd7a2f19863fe92ed4686e7127be5af1 (HEAD -> main)
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>
Date:   Sun Oct 23 16:53:56 2022 -0300

    first commit

diff --git a/README.md b/README.md
new file mode 100644
index 000000..3a6f8ba
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+# Curso de Github
(END)
```

Git

Diferenças

Vamos entender na prática!

```
$ git status
```

```
$ code README.md
```

Vamos acrescentar algum texto!

```
$ git diff
```

```
$ git diff --name-only
```

```
diff --git a/README.md b/README.md
index 3a6f8ba..292451c 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,3 @@
 # Curso de Github
+
+Muito bacana!!\n \
\ No newline at end of file
(END)
```

README.md
(END)

Git

Desfazendo coisas

Vamos entender na prática!

\$ code README.md

Modificar o arquivo!

\$ git status

\$ git diff

\$ git checkout README.md

```
[→ project-1 git:(main) code README.md
[→ project-1 git:(main) git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
[→ project-1 git:(main) ✘ git diff
[→ project-1 git:(main) ✘ git checkout README.md
Updated 1 path from the index
[→ project-1 git:(main) █
```

```
diff --git a/README.md b/README.md
index 3a6f8ba..5a4b1c4 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,3 @@
  # Curso de Github
+
+beleza
\ No newline at end of file
(END)
```

Git

Desfazendo coisas

Vamos entender na prática!

```
$ git add README.md
```

```
$ git commit -m "feat: new comment on  
README"
```

```
$ git log
```

Como podemos desfazer algo commitado?

```
[→ project-1 git:(main) ✘ git add README.md  
[→ project-1 git:(main) ✘ git commit -m "feat: new comment on Readme"  
[main c207fbc] feat: new comment on Readme  
 1 file changed, 2 insertions(+)  
[→ project-1 git:(main) git log
```

```
commit c207fbc1c658cf69b3037d4c06339345a534d651 (HEAD -> main)  
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>  
Date:   Sun Oct 23 18:19:39 2022 -0300  
  
        feat: new comment on Readme  
  
commit c072d01ecd7a2f19863fe92ed4686e7127be5af1  
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>  
Date:   Sun Oct 23 16:53:56 2022 -0300  
  
        first commit  
(END)
```

Git

Desfazendo coisas

Git Reset

1. **Soft**: vai pegar as modificações, matar o commit e deixar **staged** para commitar
2. **Mixed**: vai pegar as modificações, matar o commit e irá voltar pra **modified**
3. **Hard**: vai matar tudo que foi feito no commit

The screenshot shows the official Git documentation website at <https://git-scm.com>. The specific page is for the command `git reset`. The header includes the Git logo and the tagline "local-branching-on-the-cheap". A search bar is at the top right. The left sidebar has links for "About", "Documentation" (which is currently selected), "Reference" (with sub-links for "Book", "Videos", and "External Links"), "Downloads", and "Community". The main content area shows the version information "Version 2.38.1" and the last update date "git-reset last updated in 2.38.1". It also shows language options "Topics" and "English". The page content includes sections for "NAME", "SYNOPSIS", and "DESCRIPTION". The "SYNOPSIS" section contains the command syntax:

```
git reset [-q] [<tree-ish>] [--] <pathspec>...
git reset [-q] [--pathspec-from-file=<file>] [--pathspec-file-nul] [<tree-ish>]
git reset (--patch | -p) [<tree-ish>] [--] [<pathspec>...]
git reset [--soft | --mixed [-N] | --hard | --merge | --keep] [-q]
[<commit>]
```

The "DESCRIPTION" section explains that in the first three forms, entries from `<tree-ish>` are copied to the index. In the last form, the current branch head (`HEAD`) is set to `<commit>`, optionally modifying the index and working tree to match. The `<tree-ish> / <commit>` defaults to `HEAD` in all forms.

<https://git-scm.com/docs/git-reset>

Git Reset Soft

Desfazendo coisas

Vamos entender na prática!

```
$ git log
```

```
$ git reset --soft <HASH_COMMIT>
```

```
$ git status
```

```
commit c207fbc1c658cf69b3037d4c06339345a534d651 (HEAD -> main)
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>
Date:   Sun Oct 23 18:19:39 2022 -0300

  feat: new comment on Readme

commit c072d01ecd7a2f19863fe92ed4686e7127be5af1
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>
Date:   Sun Oct 23 16:53:56 2022 -0300

  first commit
(END)
```

```
[→ project-1 git:(main) git log
[→ project-1 git:(main) git reset --soft c072d01ecd7a2f19863fe92ed4686e7127be5af1
[→ project-1 git:(main) ✘ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified: README.md

[→ project-1 git:(main) ✘ ]
```

Git Reset Mixed

Desfazendo coisas

Vamos entender na prática!

```
$ git commit -m "new comment on  
README"
```

```
$ git status
```

```
$ git log
```

```
$ git reset --mixed <HASH_COMMIT>
```

```
$ git status
```

```
commit 53b631e4ce7049bc1a4fc300f9444fc9433925 (HEAD -> main)  
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>  
Date: Sun Oct 23 18:28:05 2022 -0300
```

```
feat: new comment on README
```

```
commit c072d01ecd7a2f19863fe92ed4686e7127be5af1  
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>  
Date: Sun Oct 23 16:53:56 2022 -0300
```

```
first commit  
(END)
```

```
[→ project-1 git:(main) git log  
[→ project-1 git:(main) git reset --mixed c072d01ecd7a2f19863fe92ed4686e7127be5af1  
Unstaged changes after reset:  
M README.md  
[→ project-1 git:(main) ✘ git status  
On branch main  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
    modified: README.md  
  
no changes added to commit (use "git add" and/or "git commit -a")  
→ project-1 git:(main) ✘
```

Git Reset Hard

Desfazendo coisas

Vamos entender na prática!

```
$ git add README.md
```

```
$ git status
```

```
$ git commit -m "new comment on  
README"
```

```
$ git log
```

```
$ git reset --hard <HASH_COMMIT>
```

```
$ git log
```

```
commit 368ec4465e8e7021c297d7fd3899d3a45fec7420 (HEAD -> main)  
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>  
Date: Sun Oct 23 18:30:46 2022 -0300
```

```
feat: new comment on README
```

```
commit c072d01ecd7a2f19863fe92ed4686e7127be5af1  
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>  
Date: Sun Oct 23 16:53:56 2022 -0300
```

```
first commit
```

```
(END)
```

```
→ project-1 git:(main) ✘ git add README.md  
→ project-1 git:(main) ✘ git status  
On branch main  
Changes to be committed:  
(use "git restore --staged <file>..." to unstage)  
modified: README.md
```

```
→ project-1 git:(main) ✘ git commit -m "feat: new comment on README"  
[main 368ec44] feat: new comment on README  
1 file changed, 2 insertions(+)  
→ project-1 git:(main) git log  
→ project-1 git:(main) git log  
→ project-1 git:(main) git reset --hard c072d01ecd7a2f19863fe92ed4686e7127be5af1  
HEAD is now at c072d01 first commit  
→ project-1 git:(main) git log
```

```
commit c072d01ecd7a2f19863fe92ed4686e7127be5af1 (HEAD -> main)  
Author: Fabio S. Takaki <fabio.stakaki@gmail.com>  
Date: Sun Oct 23 16:53:56 2022 -0300
```

```
first commit
```

```
(END)
```

Git

O que sabemos até então

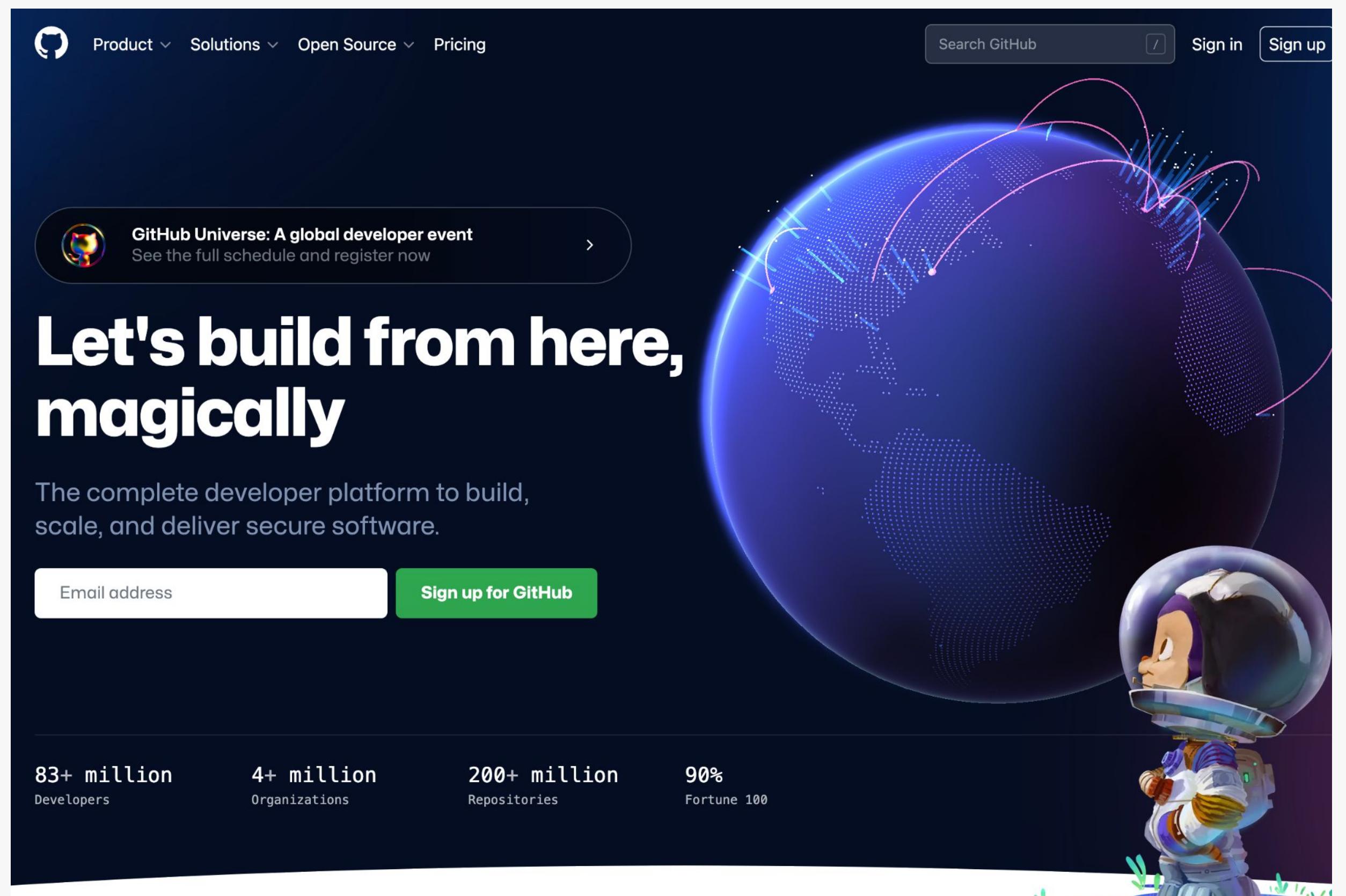
- Ciclo de vida dos arquivos
- Adicionar arquivos para Staged - git add <arquivos>
- Criar Snapshots (Fotos) - git commit -m "<mensagem>"
- Visualizar os snapshots - git show <HASH>
- Visualizar diferenças - git diff
- Desfazer coisas:
 - desfazer arquivo **modified**: git checkout <arquivos>
 - desfazer arquivos em **staged**: git reset <arquivos>
 - desfazer commits:
 - git reset –soft: desfaz e joga pra staged
 - git reset –mixed: desfaz e joga pra modified
 - git reset –hard: desfaz e exclui todas as alterações



Github Repositórios Remotos

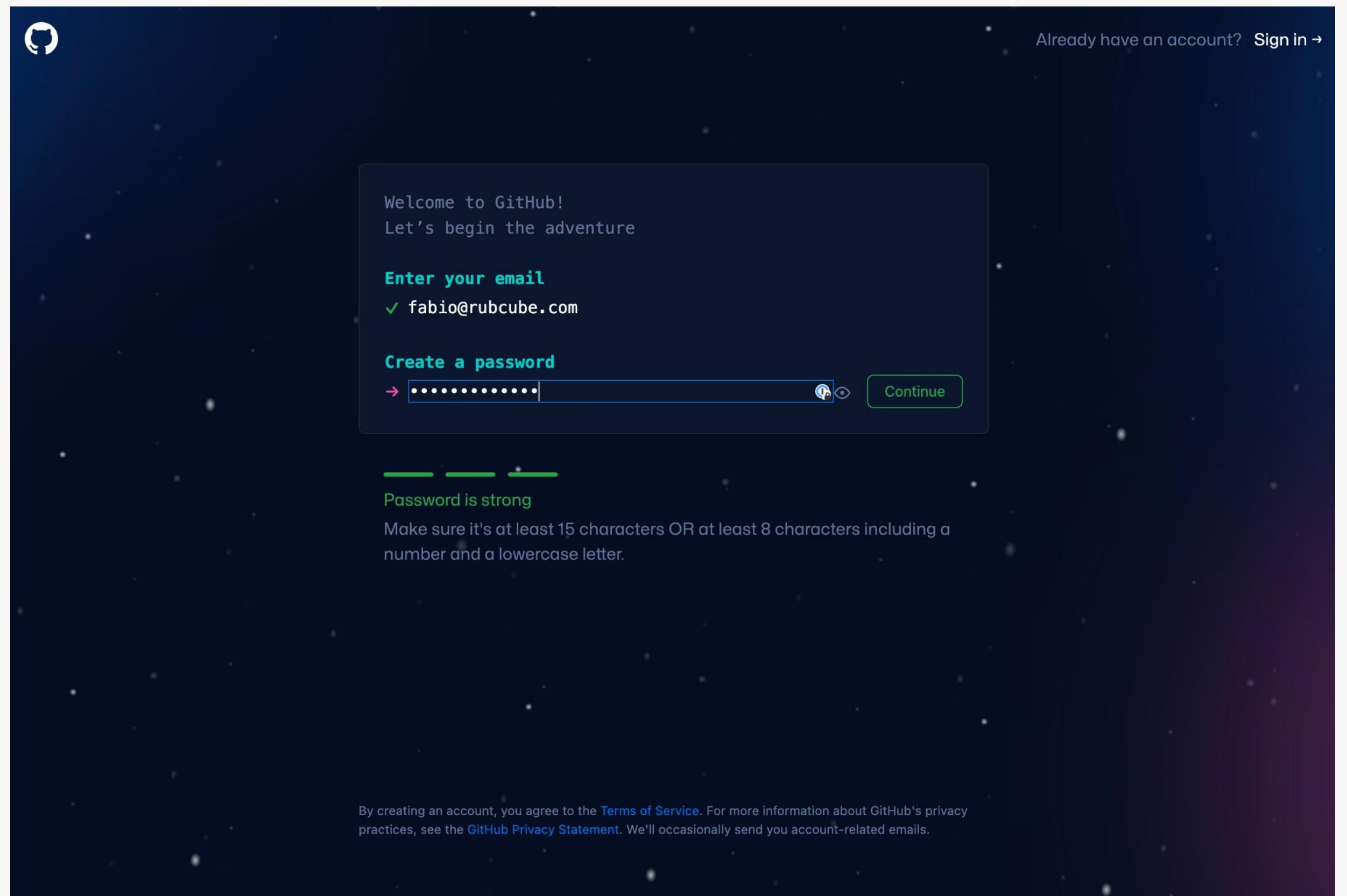
É um serviço web compartilhado para projetos que utilizam **Git** para o seu versionamento. **Github** é reconhecida por ser a maior plataforma de código open-source do mundo.

Começaremos criando uma conta e um repositório na plataforma!



Github Repositórios Remotos

- Acessar o <https://www.github.com/>
- Criar uma conta.



Github Repositórios Remotos

- Criar um repositório com nome
curso-github
- Deixar como público mesmo

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 *

 fabiotakaki /

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

Description (optional)

 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](#).

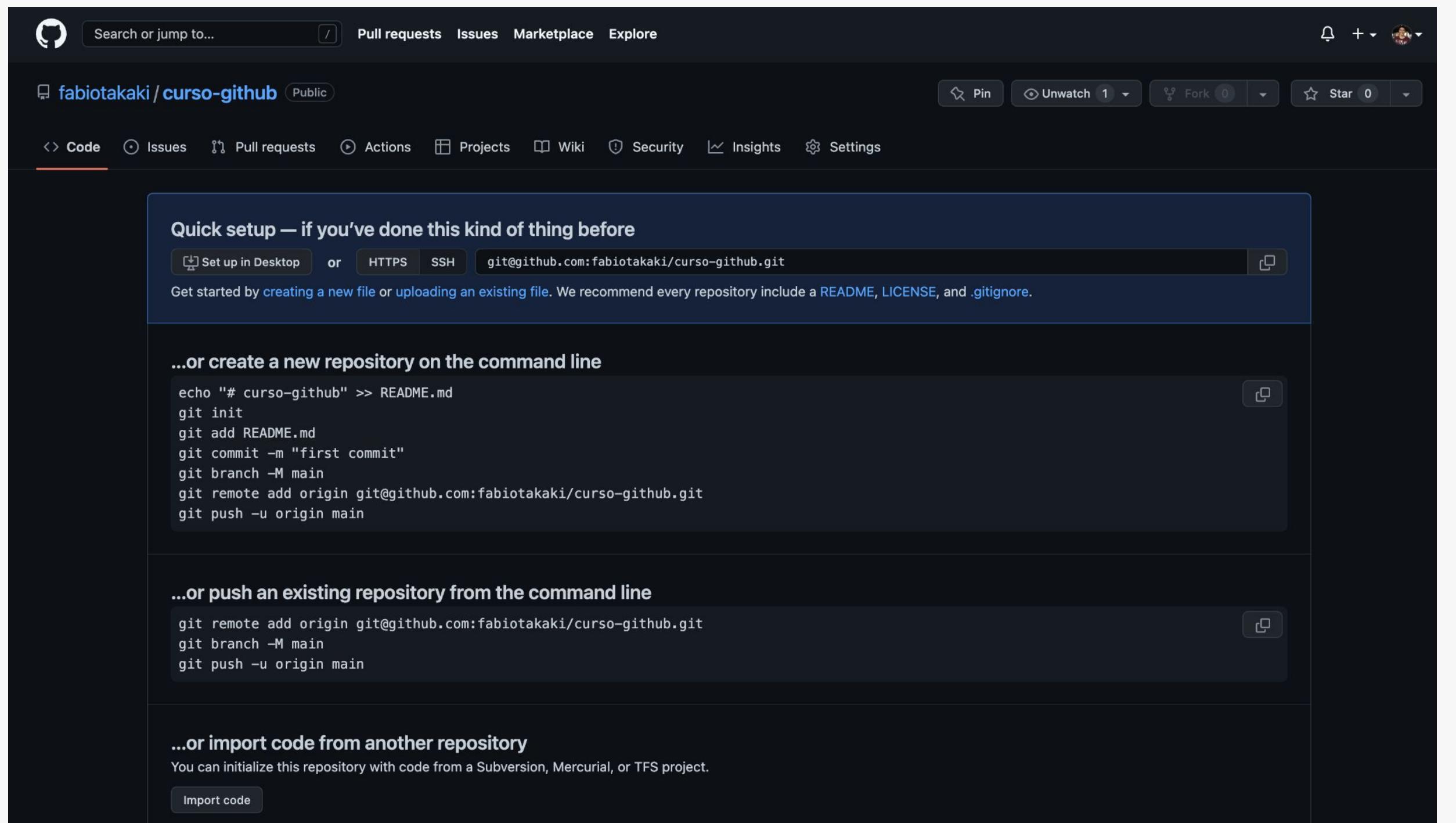
Choose a license

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

 You are creating a public repository in your personal account.

Github Repositórios Remotos

- Criar um repositório com nome **curso-github**
- Deixar como público mesmo



Github Repositórios Remotos

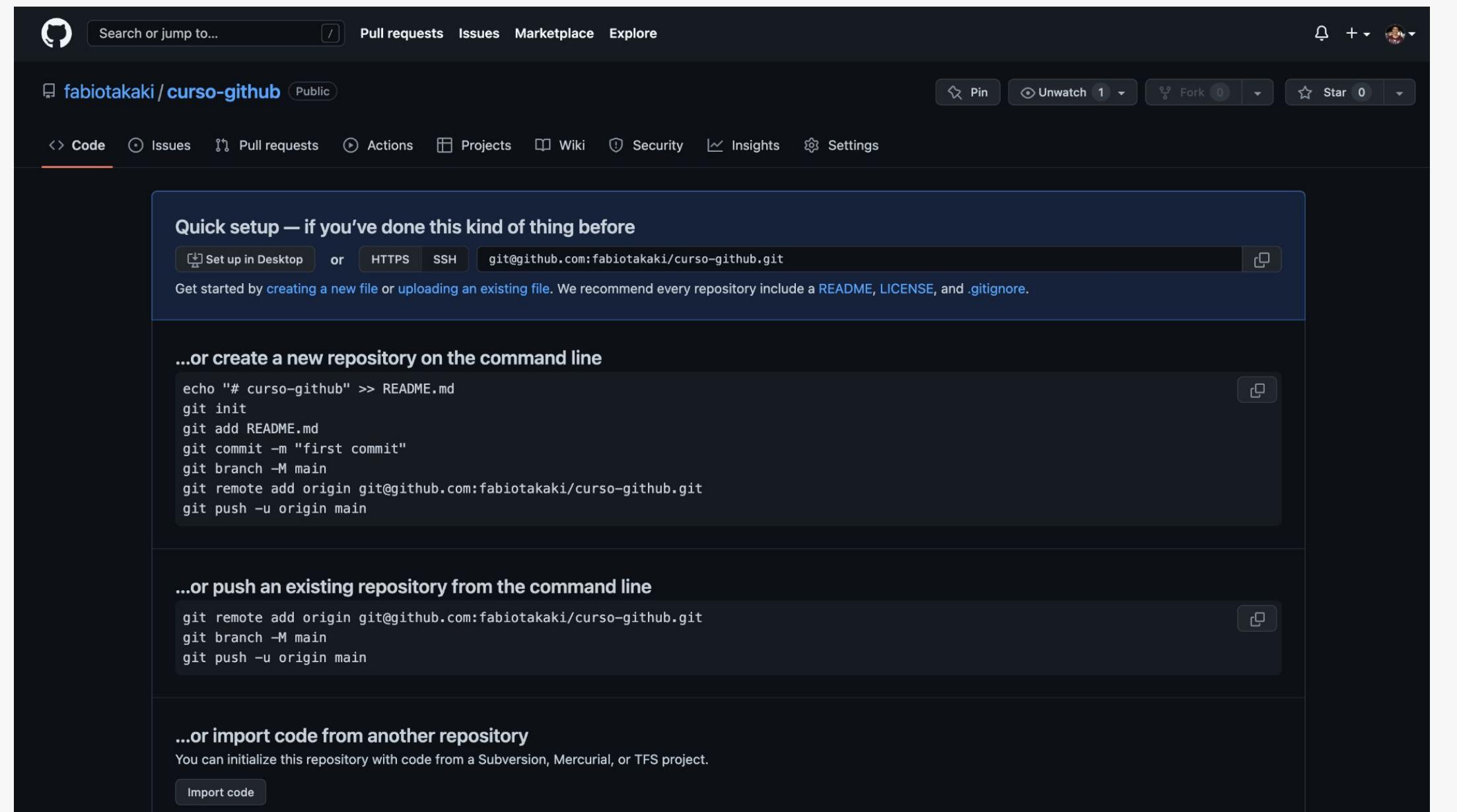
Para conseguirmos ligar o repositório local com o repositório remoto, precisaremos configurar já as chaves ssh's:

- Para gerar a chave ssh, execute abaixo:

```
$ ssh-keygen -t rsa -b 4096 -C
```

"your_email@example.com"

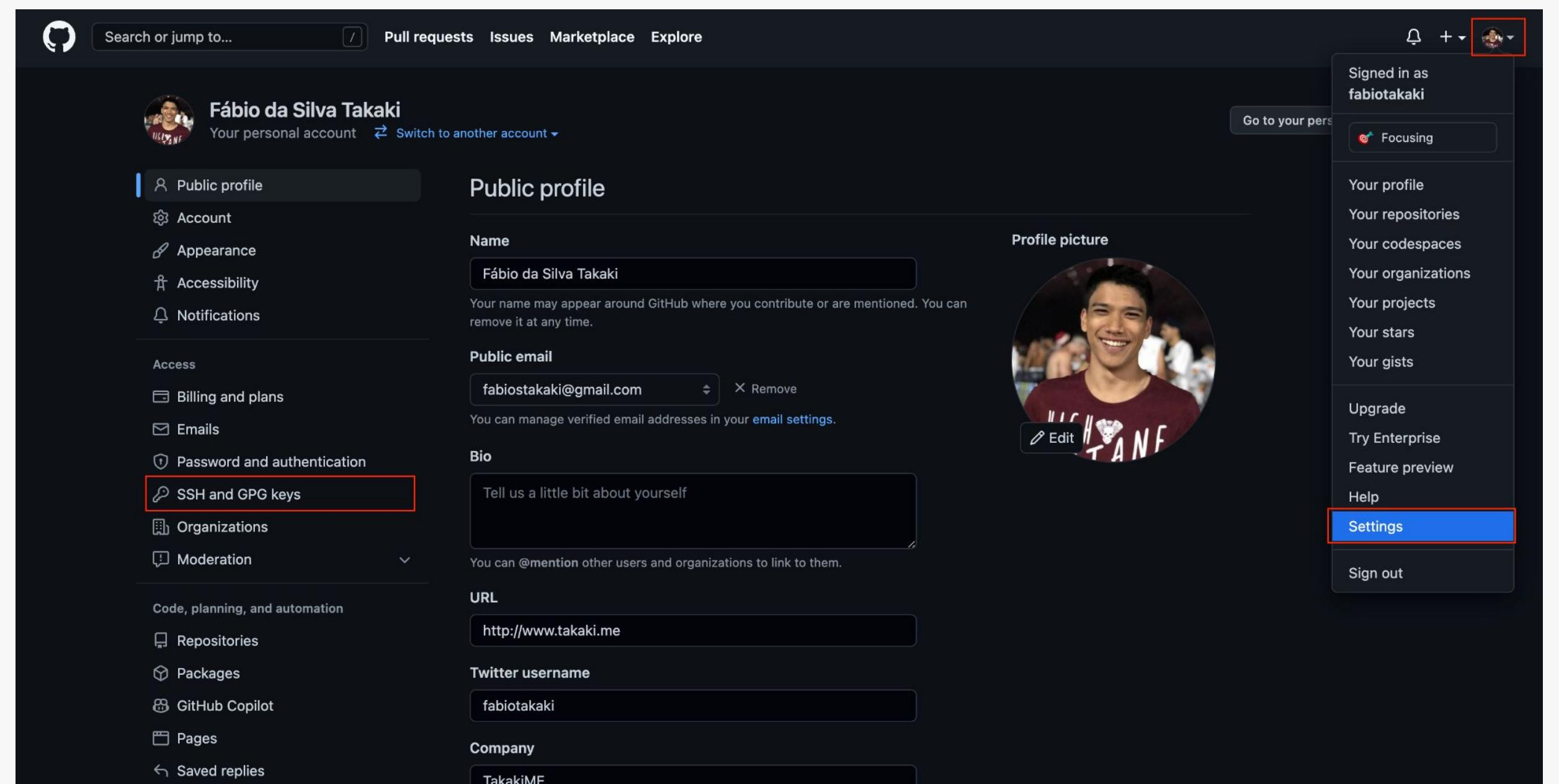
```
$ cat ~/.ssh/id_rsa.pub
```



Github Repositórios Remotos

Depois de gerado, precisamos configurar a chave pública dentro do **Github**:

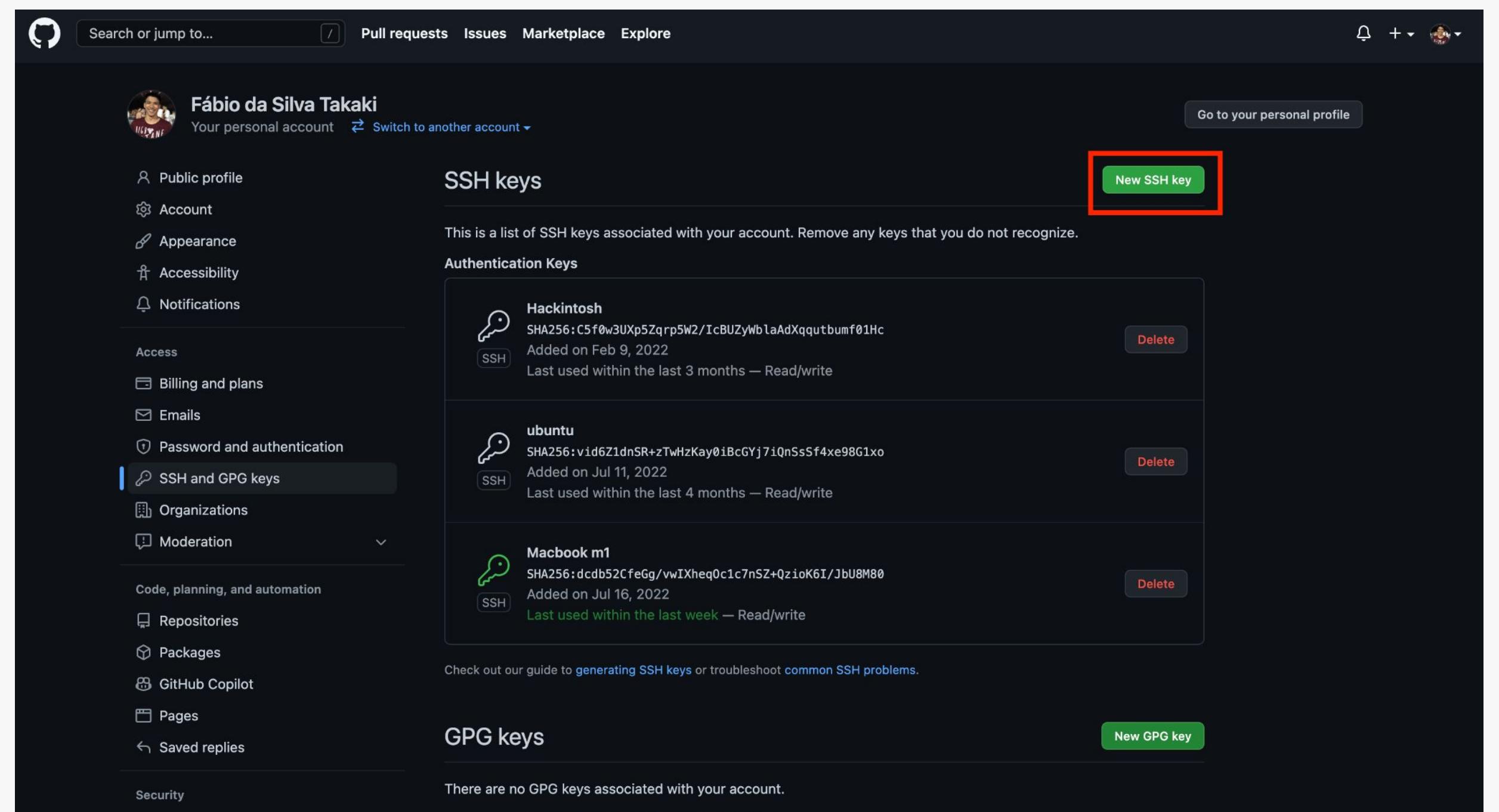
- Clicar na sua fotinha de perfil superior direito
- Clicar em SSH and GPG keys



Github Repositórios Remotos

Depois de gerado, precisamos configurar a chave pública dentro do **Github**:

- Clicar na sua fotinha de perfil superior direito
- Clicar em SSH and GPG keys
- Depois em **New SSH Key**

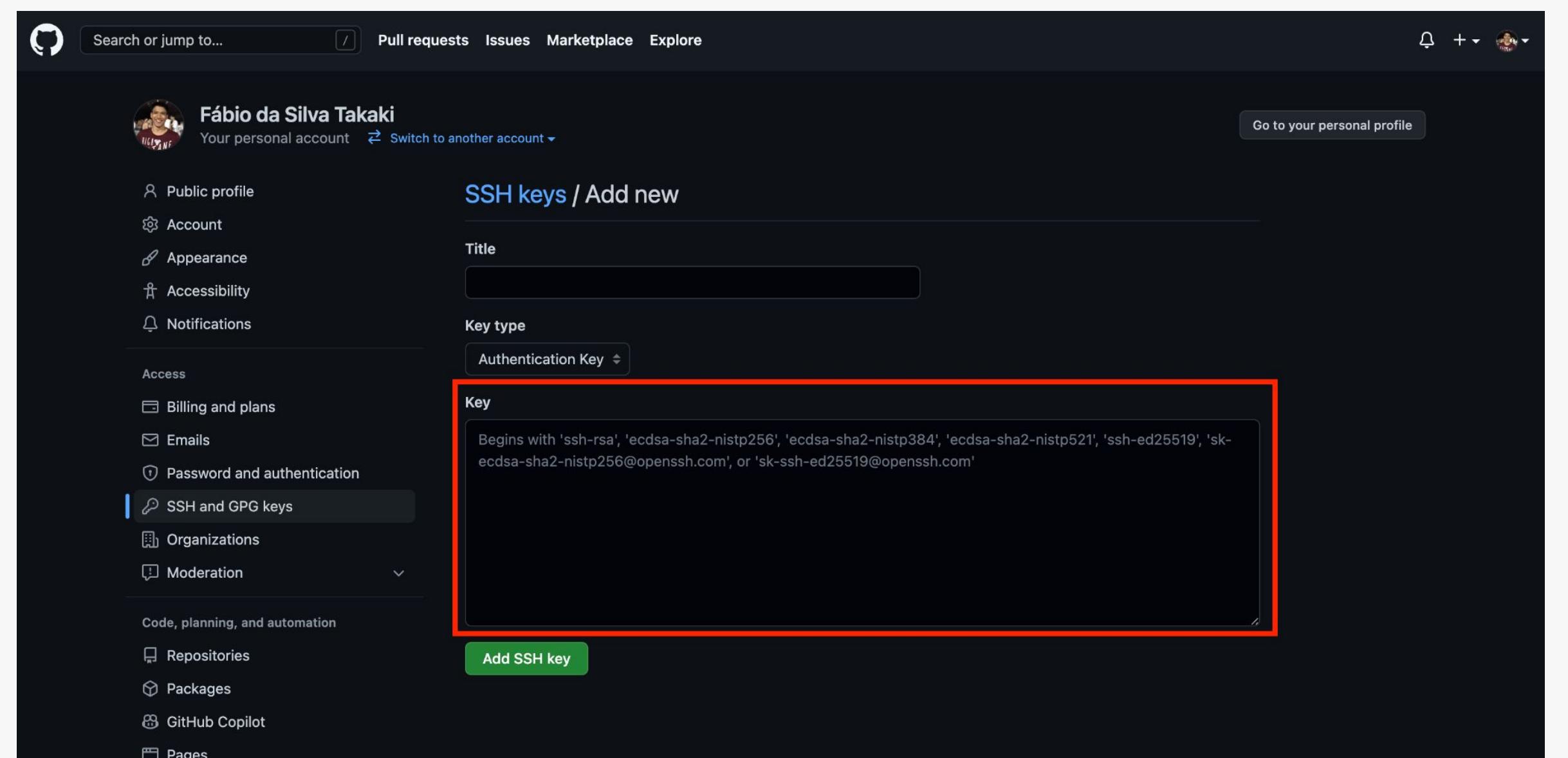


Github Repositórios Remotos

Com isso, vamos digitar no terminal o seguinte comando para copiar a chave pública:

```
$ cat ~/.ssh/id_rsa.pub
```

Ao digitar o comando, volte pro **Github**, digite um título que indique de onde vem a chave, copie o resultado que foi retornado do comando e cole na área vermelha como sinalizado ao lado. Por fim, clique em **Add SSH Key**.



Github Repositórios Remotos

Pronto! Agora você está preparado para conectar o repositório local do git com o remoto.

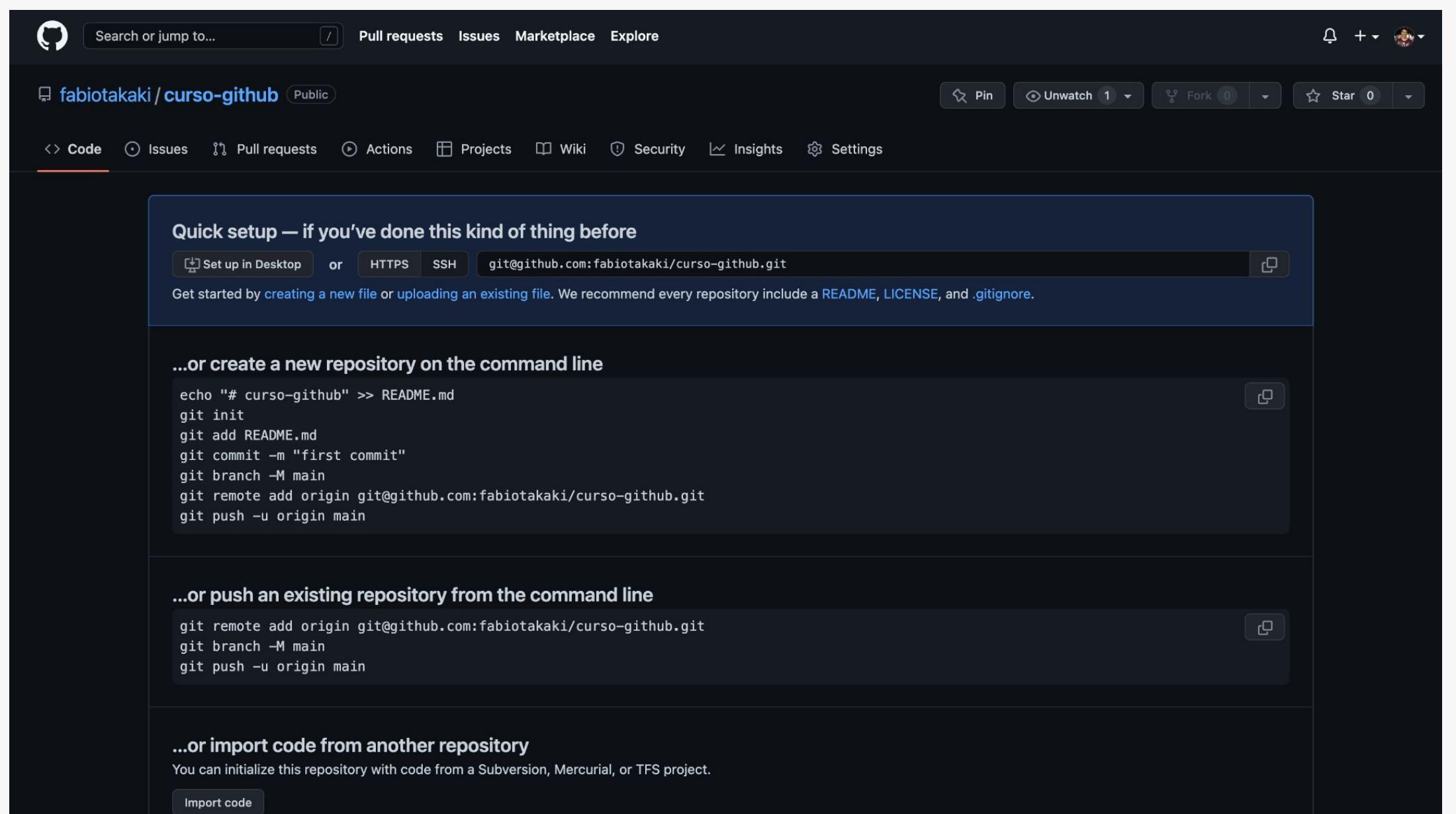
Voltaremos ao repositório e seguiremos o passo a passo que o Github indica quando já temos um repositório existente local.

\$ git remote add origin

git@github.com:fabiotakaki/curso-github.git

\$ git branch -M main

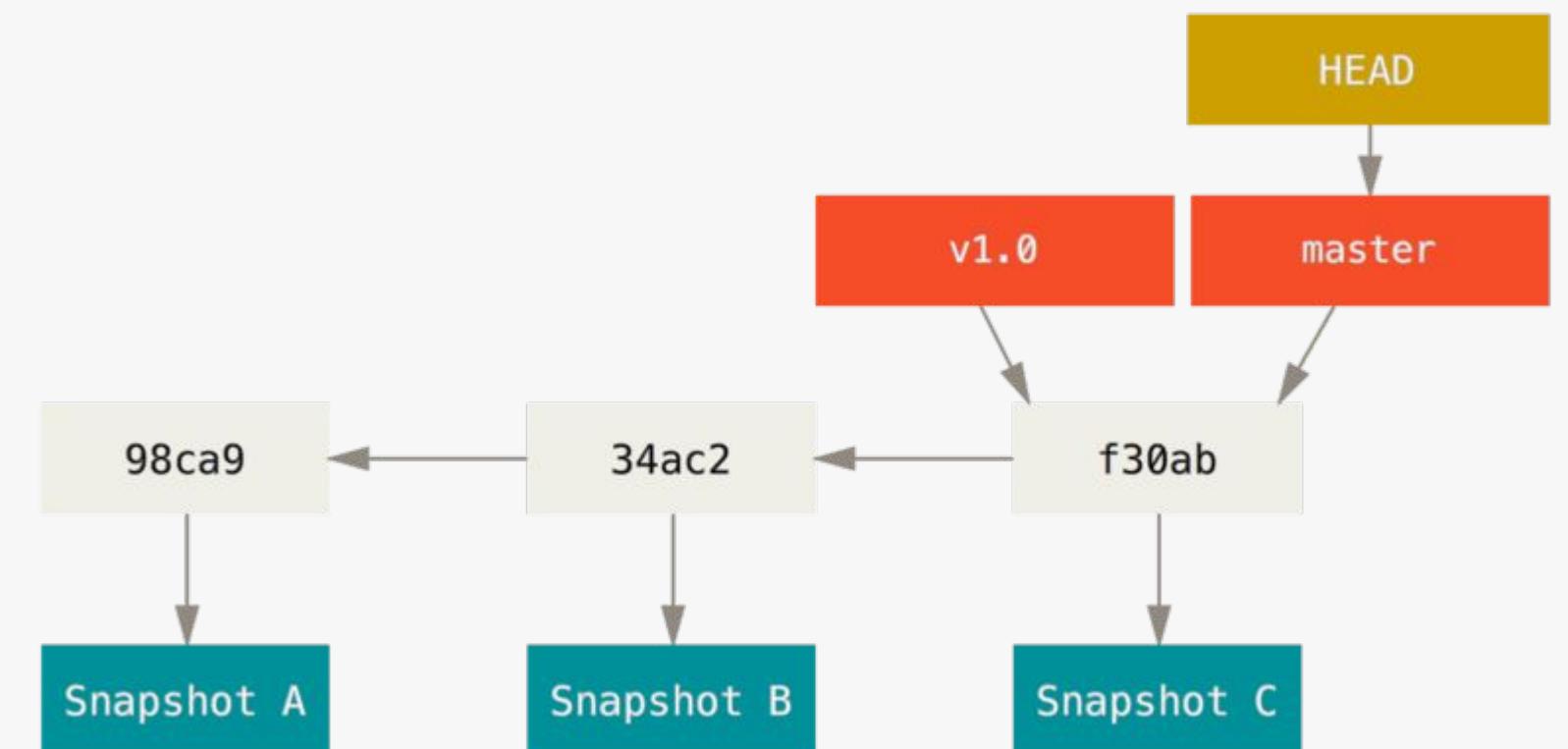
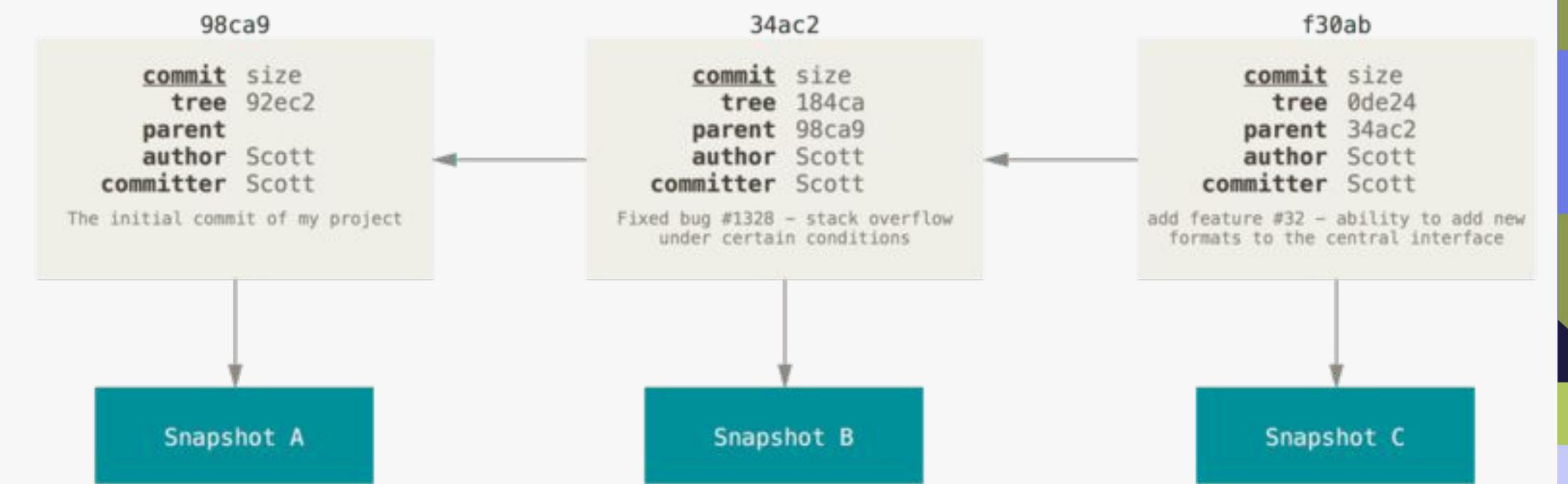
\$ git push -u origin main



Git Branch

A **Branch** (ramificação) é um ponteiro móvel que aponta para um **Commit**. Com isso, é possível você trabalhar em uma linha diferente da principal sem propriamente alterá-la. Existem diversas vantagens além desta como:

- facilmente você pode criar e remover branches conforme necessidade
- múltiplas pessoas trabalhando com o mesmo código sem afetar uns aos outros
- evita conflitos



Git Branch

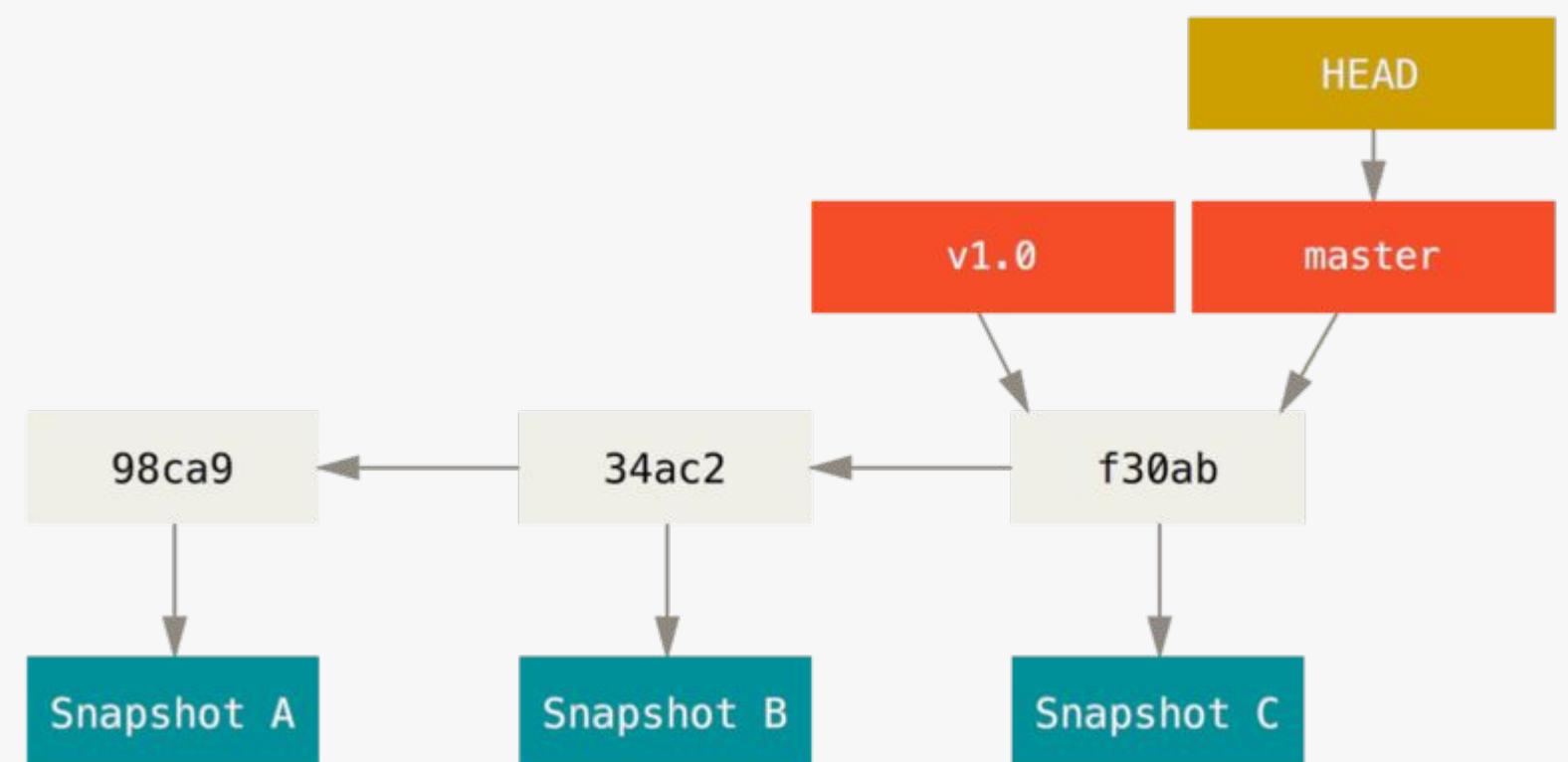
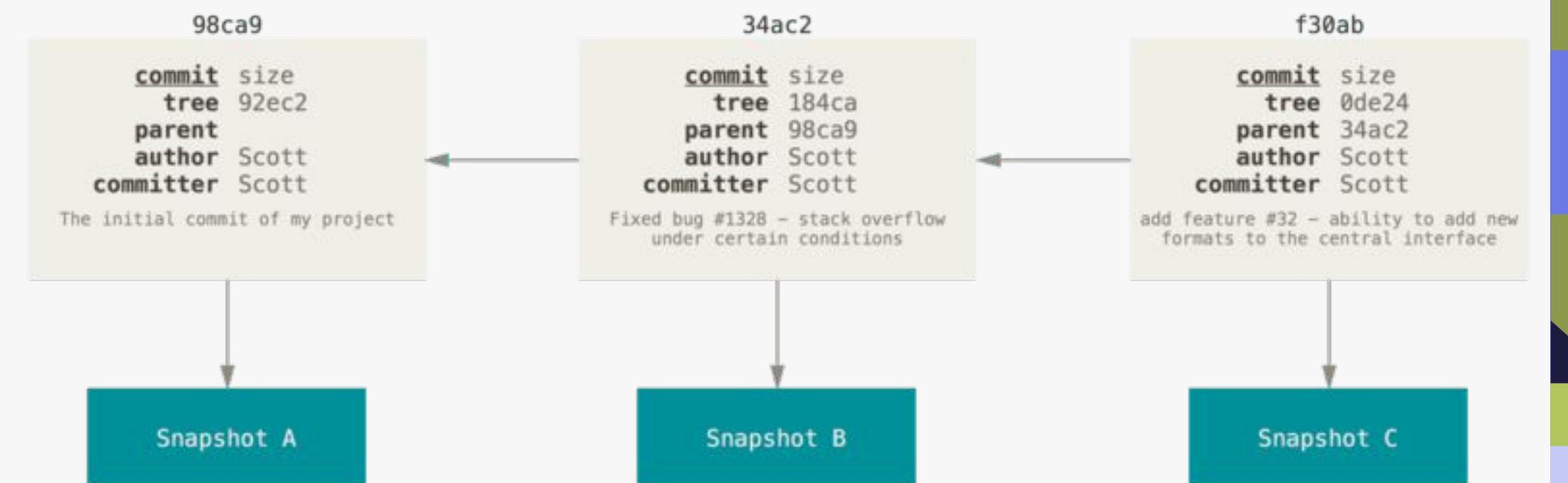
Vamos entender na prática!

```
$ git checkout -b teste
```

```
$ git branch
```

```
$ git checkout master
```

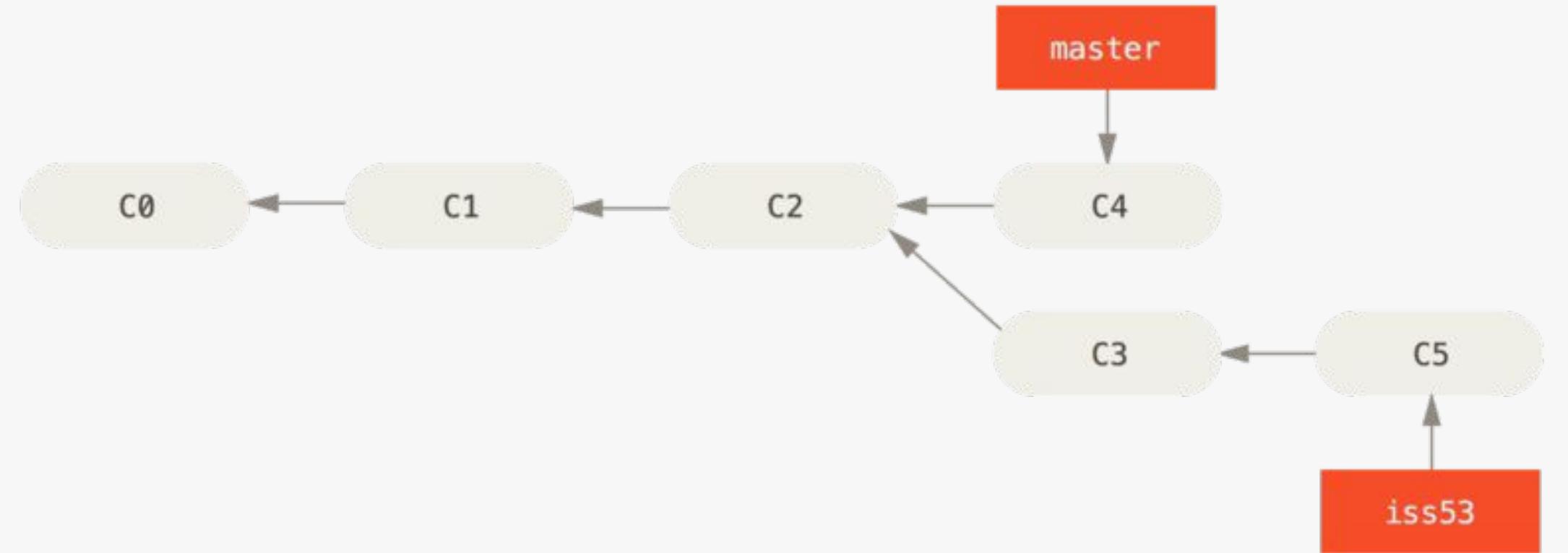
```
$ git branch -D teste
```



Git Merge

Vamos entender na prática!

```
$ git checkout -b iss53
$ git branch
$ echo "Arquivo iss53" >> iss53.txt
$ git add iss53.txt
$ git commit -m "iss53"
$ echo "Arquivo 2 iss53" >> iss53_2.txt
$ git add iss53_2.txt
$ git commit -m "file 2 iss53"
$ git log
```

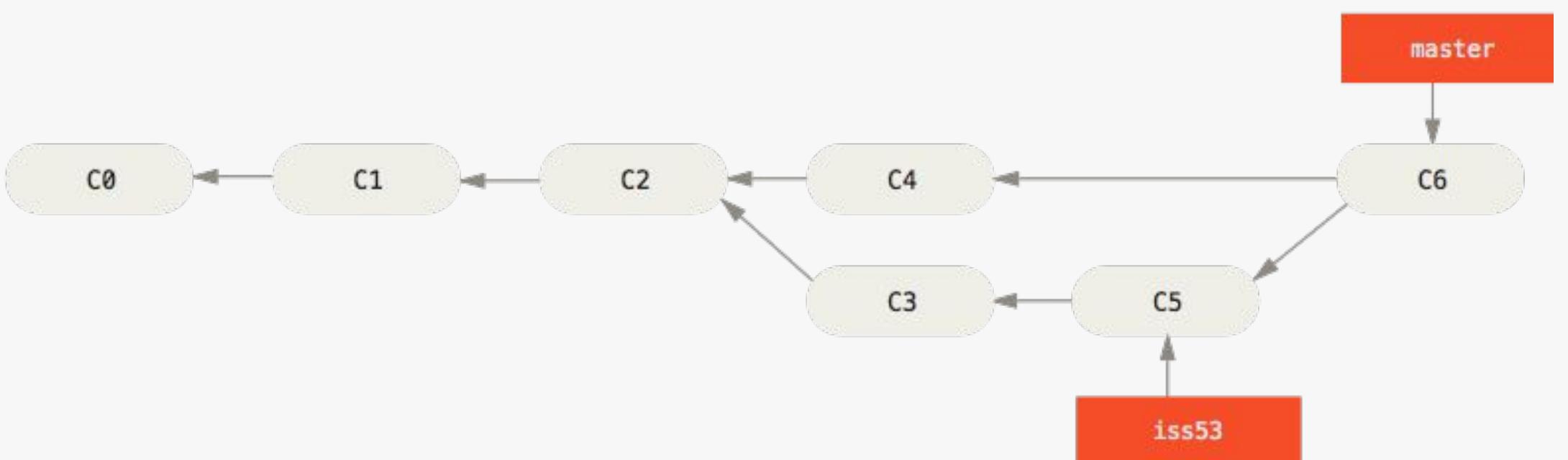
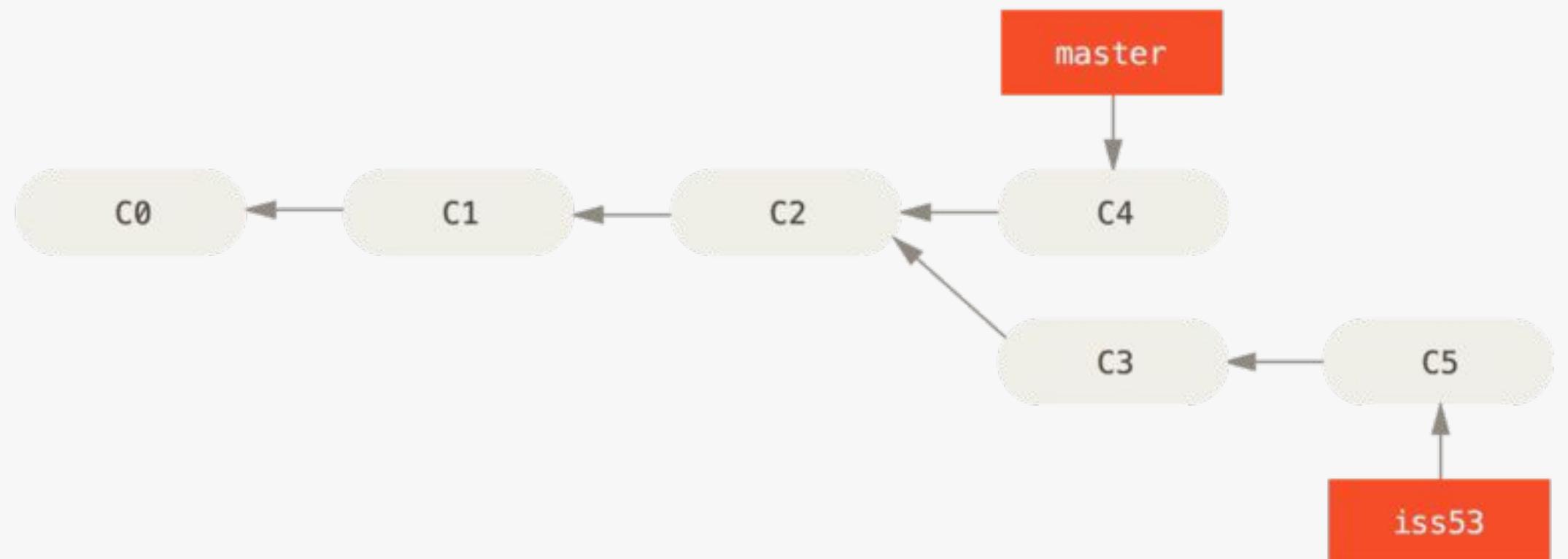


<https://git-scm.com/book/pt-br/v2/Branches-no-Git-O-b%C3%A1sico-de-Ramifica%C3%A7%C3%A3o-Branch-e-Mesclagem-Merge>

Git Merge

Vamos entender na prática!

```
$ git checkout master  
$ echo "Arquivo hotfix" >> hotfix.txt  
$ git add hotfix.txt  
$ git commit -m "hotfix"  
$ git log  
$ git merge iss53  
$ git log --graph
```



<https://git-scm.com/book/pt-br/v2/Branches-no-Git-O-b%C3%A1sico-de-Ramifica%C3%A7%C3%A3o-Branch-e-Mesclagem-Merge>

Git Merge

Vantagens

- Operação não destrutiva

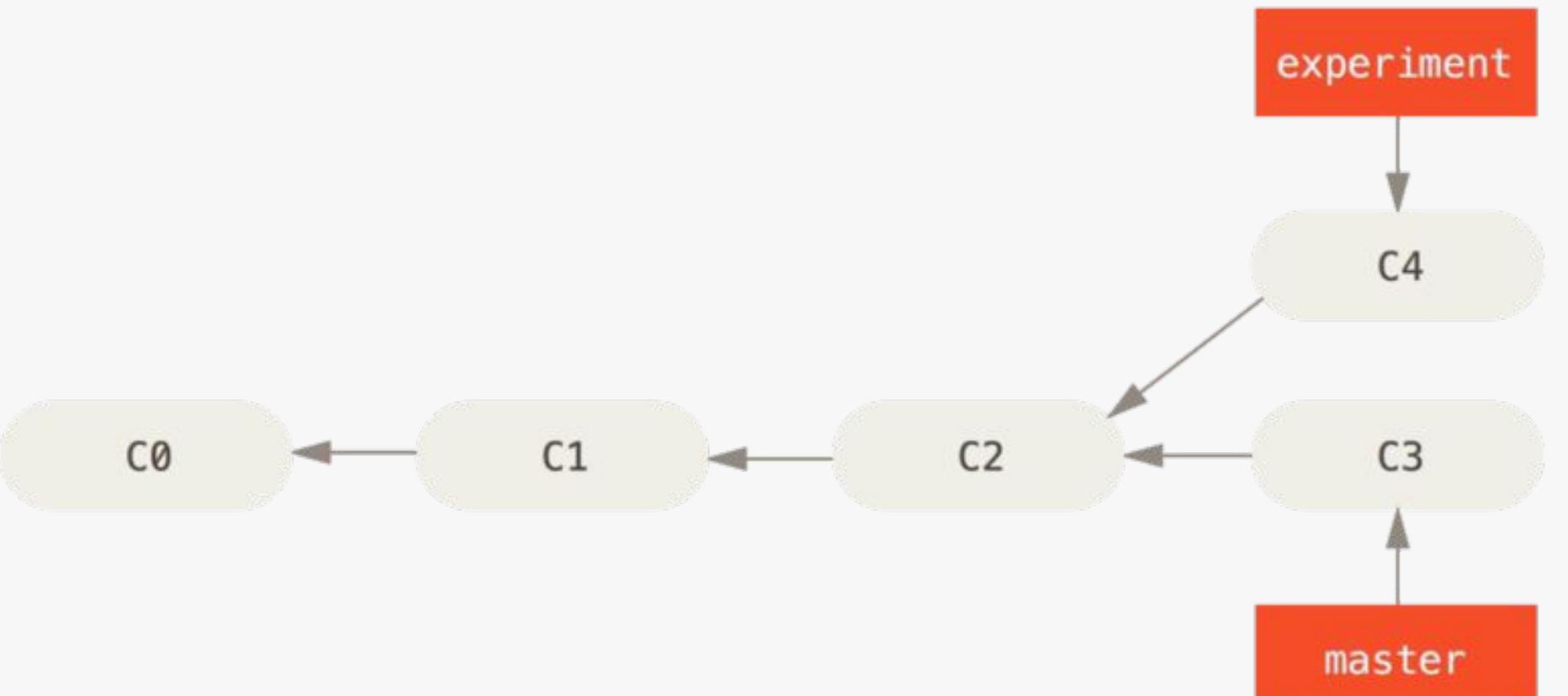
Desvantagens

- Commit extra
- Histórico poluído

Git Rebase

Vamos entender na prática!

```
$ git checkout -b experiment  
$ git branch  
$ echo "Arquivo rebase" >> rebase.txt  
$ git add rebase.txt  
$ git commit -m "rebase"  
$ git log
```



<https://git-scm.com/book/pt-br/v2/Branches-no-Git-Rebase>

Git Rebase

Vamos entender na prática!

```
$ git checkout main
```

```
$ echo "Arquivo rebase hotfix" >> rebase_hotfix.txt
```

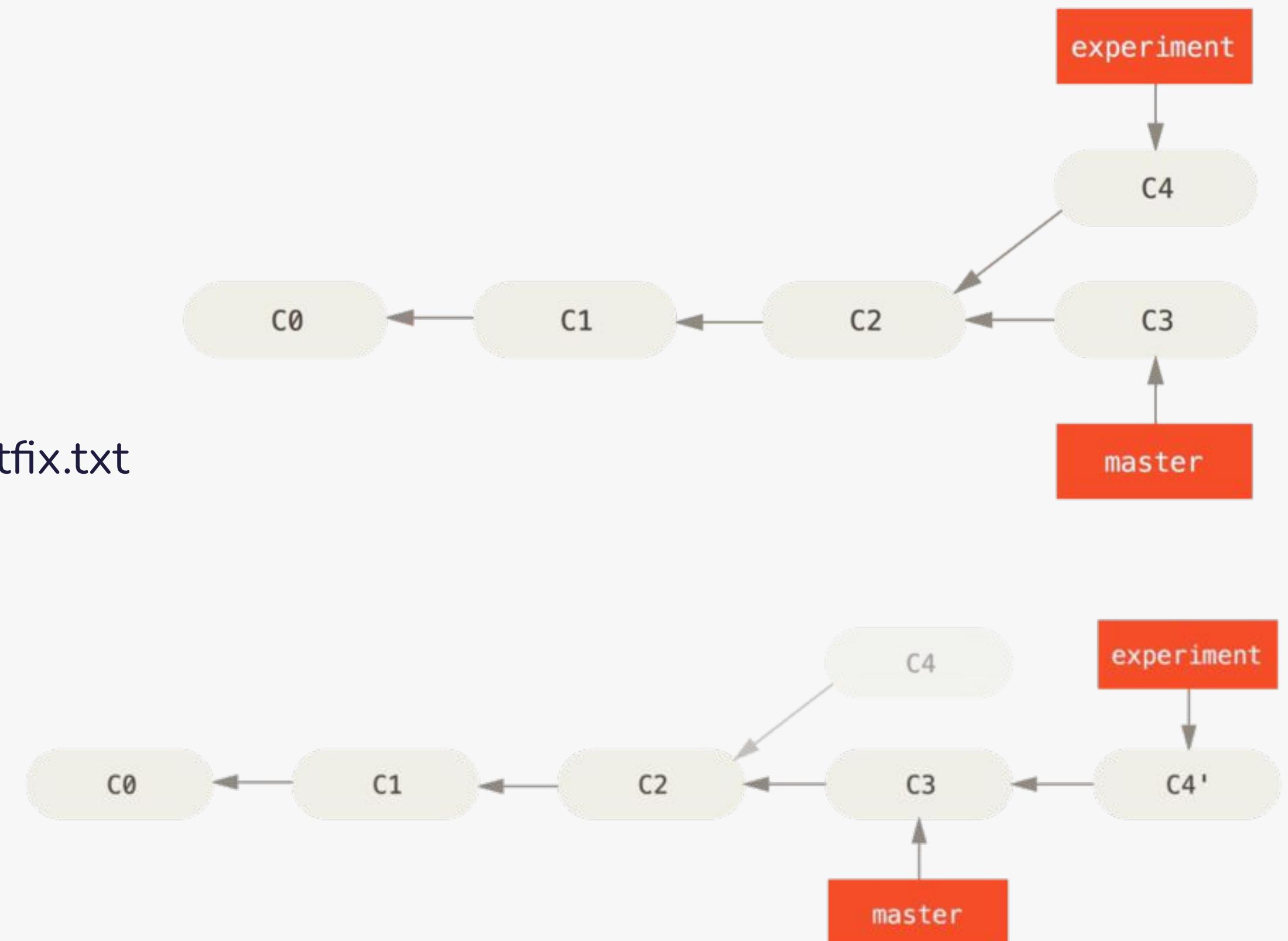
```
$ git add rebase_hotfix.txt
```

```
$ git commit -m "rebase hotfix"
```

```
$ git log
```

```
$ git checkout experiment
```

```
$ git rebase experiment
```



Git Rebase

Vantagens

- Evita commits extras
- Histórico dos commits fica linear

Desvantagens

- Reescreve commits do histórico

Git conflitos!!

- Vamos entender na prática!
- \$ echo "Teste conflito" >> teste.txt
- \$ git commit -am "teste conflito"
- \$ git checkout -b teste_conflito
- \$ git checkout main
- \$ vi teste.txt
- Modifica a linha do teste conflito
- \$ git commit -am "Teste conflito main"
- \$ git checkout teste_conflito
- \$ vi teste.txt
- Modifica a linha do teste conflito

```
[→ project-1 git:(main) echo "Teste conflito" >> teste.txt
[→ project-1 git:(main) ✘ git commit -am "teste conflito"
[main 02588aa] teste conflito
 1 file changed, 1 insertion(+)
[→ project-1 git:(main) git checkout -b teste_conflito
Switched to a new branch 'teste_conflito'
[→ project-1 git:(teste_conflito) git checkout main
Switched to branch 'main'
[→ project-1 git:(main) vi teste.txt
[→ project-1 git:(main) ✘ git commit -am "Teste conflito main"
[main 6e32e13] Teste conflito main
 1 file changed, 1 insertion(+), 1 deletion(-)
[→ project-1 git:(main) git checkout teste_conflito
Switched to branch 'teste_conflito'
[→ project-1 git:(teste_conflito) vi teste.txt
[→ project-1 git:(teste_conflito) ✘ git commit -am "Teste conflito branch teste"
[teste_conflito a599c07] Teste conflito branch teste
 1 file changed, 1 insertion(+), 1 deletion(-)
[→ project-1 git:(teste_conflito) git checkout main
Switched to branch 'main'
[→ project-1 git:(main) git merge teste_conflito
merge: teste_conflito - not something we can merge
[→ project-1 git:(main) git merge teste_conflito
Auto-merging teste.txt
CONFLICT (content): Merge conflict in teste.txt
Automatic merge failed; fix conflicts and then commit the result.
[→ project-1 git:(main) ✘ vi teste.txt
[→ project-1 git:(main) ✘ git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
    (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:  teste.txt

no changes added to commit (use "git add" and/or "git commit -a")
[→ project-1 git:(main) ✘ git commit -am "merge"
[main 1085c6e] merge
```

```
<<<<< HEAD
Teste conflito
=====
Teste conflito
>>>>> teste_conflito
```

Git conflitos!!

- Vamos entender na prática!
- \$ git commit -am "Teste conflito branch teste"
- \$ git checkout main
- \$ git merge teste_conflito
- \$ vi teste.txt
- \$ git commit -am "Merge resolvido"

```
[→ project-1 git:(main) echo "Teste conflito" >> teste.txt
[→ project-1 git:(main) ✘ git commit -am "teste conflito"
[main 02588aa] teste conflito
 1 file changed, 1 insertion(+)
[→ project-1 git:(main) git checkout -b teste_conflito
Switched to a new branch 'teste_conflito'
[→ project-1 git:(teste_conflito) git checkout main
Switched to branch 'main'
[→ project-1 git:(main) vi teste.txt
[→ project-1 git:(main) ✘ git commit -am "Teste conflito main"
[main 6e32e13] Teste conflito main
 1 file changed, 1 insertion(+), 1 deletion(-)
[→ project-1 git:(main) git checkout teste_conflito
Switched to branch 'teste_conflito'
[→ project-1 git:(teste_conflito) vi teste.txt
[→ project-1 git:(teste_conflito) ✘ git commit -am "Teste conflito branch teste"
[teste_conflito a599c07] Teste conflito branch teste
 1 file changed, 1 insertion(+), 1 deletion(-)
[→ project-1 git:(teste_conflito) git checkout main
Switched to branch 'main'
[→ project-1 git:(main) git merge teste_conflito
merge: teste_conflito - not something we can merge
[→ project-1 git:(main) git merge teste_conflito
Auto-merging teste.txt
CONFLICT (content): Merge conflict in teste.txt
Automatic merge failed; fix conflicts and then commit the result.
[→ project-1 git:(main) ✘ vi teste.txt
[→ project-1 git:(main) ✘ git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
    (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:  teste.txt

no changes added to commit (use "git add" and/or "git commit -a")
[→ project-1 git:(main) ✘ git commit -am "merge"
[main 1085c6e] merge
```

```
<<<<< HEAD
Teste conflito
=====
Teste conflito
>>>>> teste_conflito
```

Git .gitignore

- Vamos entender na prática!
- \$ echo "{ \"hello\": \"world\" }" >> hello.json
- \$ echo "Novo teste" >> teste.txt
- \$ git status
- \$ echo "*.json" >> .gitignore
- \$ git status

```
[→ project-1 git:(main) echo "{ \"hello\": \"world\" }" >> hello.json
[→ project-1 git:(main) ✘ cat hello.json
{ "hello": "world" }
[→ project-1 git:(main) ✘ echo "Novo teste" >> teste.txt
[→ project-1 git:(main) ✘ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello.json
    teste.txt

nothing added to commit but untracked files present (use "git add" to track)
[→ project-1 git:(main) ✘ echo "*.json" >> .gitignore
[→ project-1 git:(main) ✘ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    teste.txt

nothing added to commit but untracked files present (use "git add" to track)
[→ project-1 git:(main) ✘ code .gitignore
[→ project-1 git:(main) ✘ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello.json
    teste.txt

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

Git

git stash

- Vamos entender na prática!
- \$ git add .
- \$ git commit -m "files example"
- \$ echo "Teste stash" >> teste.txt
- \$ git checkout -b stash
- \$ git status
- \$ git stash
- \$ git stash list
- \$ git checkout master
- \$ git stash apply

```
[→ project-1 git:(main) echo "{\"hello\": \"world\" }" >> hello.json
[→ project-1 git:(main) ✘ cat hello.json
{
  "hello": "world"
}
[→ project-1 git:(main) ✘ echo "Novo teste" >> teste.txt
[→ project-1 git:(main) ✘ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello.json
    teste.txt

nothing added to commit but untracked files present (use "git add" to track)
[→ project-1 git:(main) ✘ echo "*.json" >> .gitignore
[→ project-1 git:(main) ✘ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    teste.txt

nothing added to commit but untracked files present (use "git add" to track)
[→ project-1 git:(main) ✘ code .gitignore
[→ project-1 git:(main) ✘ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello.json
    teste.txt

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

Git

git revert

- Vamos entender na prática!
- \$ echo "{ \"quebrou\": \"o codigo\" }" >> hello.json
- \$ git add hello.json
- \$ git commit -m "feat broken"
- \$ git log
- \$ git revert <HASH>
- \$ git log
- \$ git show <HASH>

```
[→ project-1 git:(main) echo "{\"quebrou\": \"o codigo\" }" >> hello.json
[→ project-1 git:(main) ✘ cat hello.json
{
  "quebrou": "o codigo"
}
[→ project-1 git:(main) ✘ echo "Novo teste" >> teste.txt
[→ project-1 git:(main) ✘ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello.json
    teste.txt

nothing added to commit but untracked files present (use "git add" to track)
[→ project-1 git:(main) ✘ echo "*.json" >> .gitignore
[→ project-1 git:(main) ✘ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    teste.txt

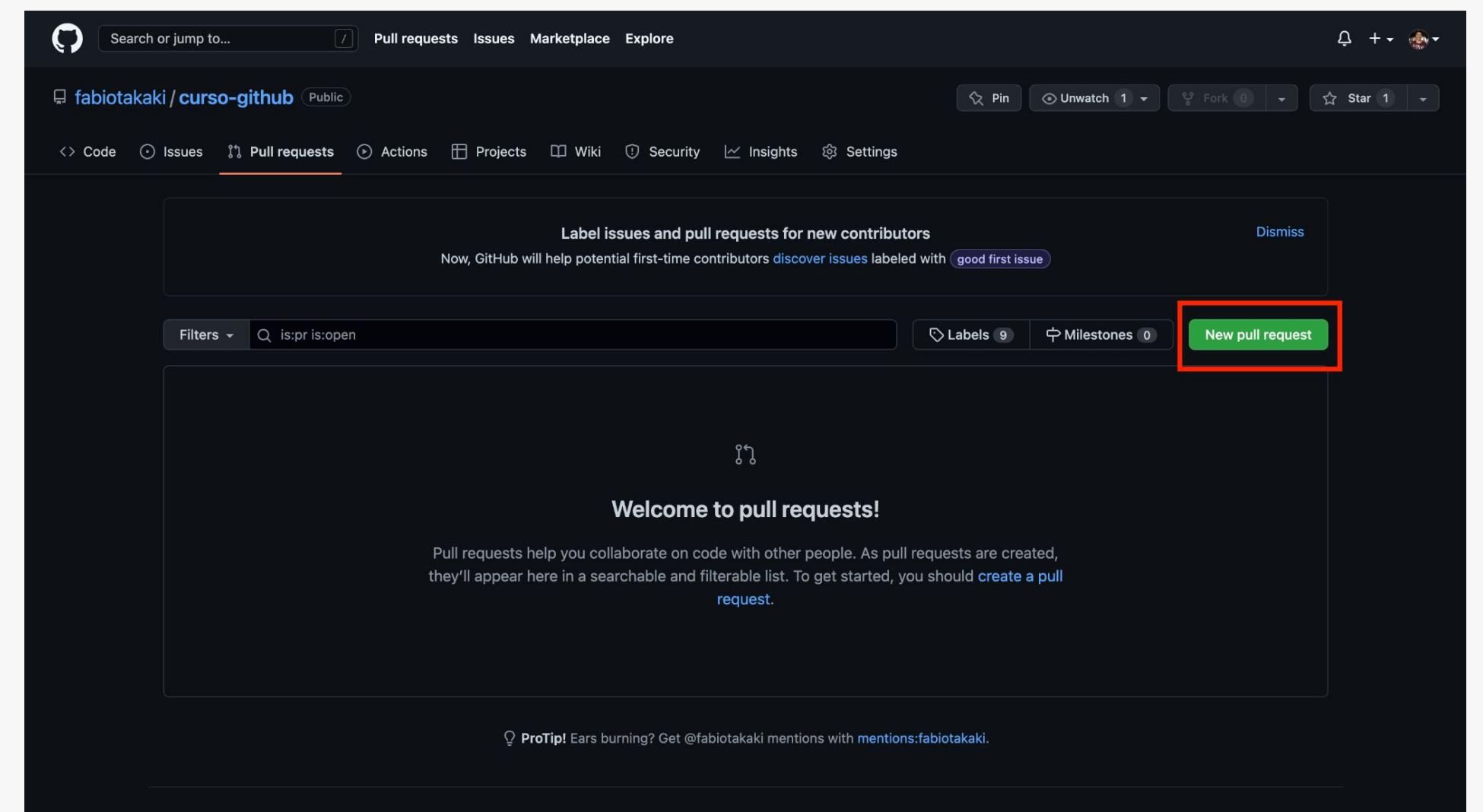
nothing added to commit but untracked files present (use "git add" to track)
[→ project-1 git:(main) ✘ code .gitignore
[→ project-1 git:(main) ✘ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello.json
    teste.txt

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

Github

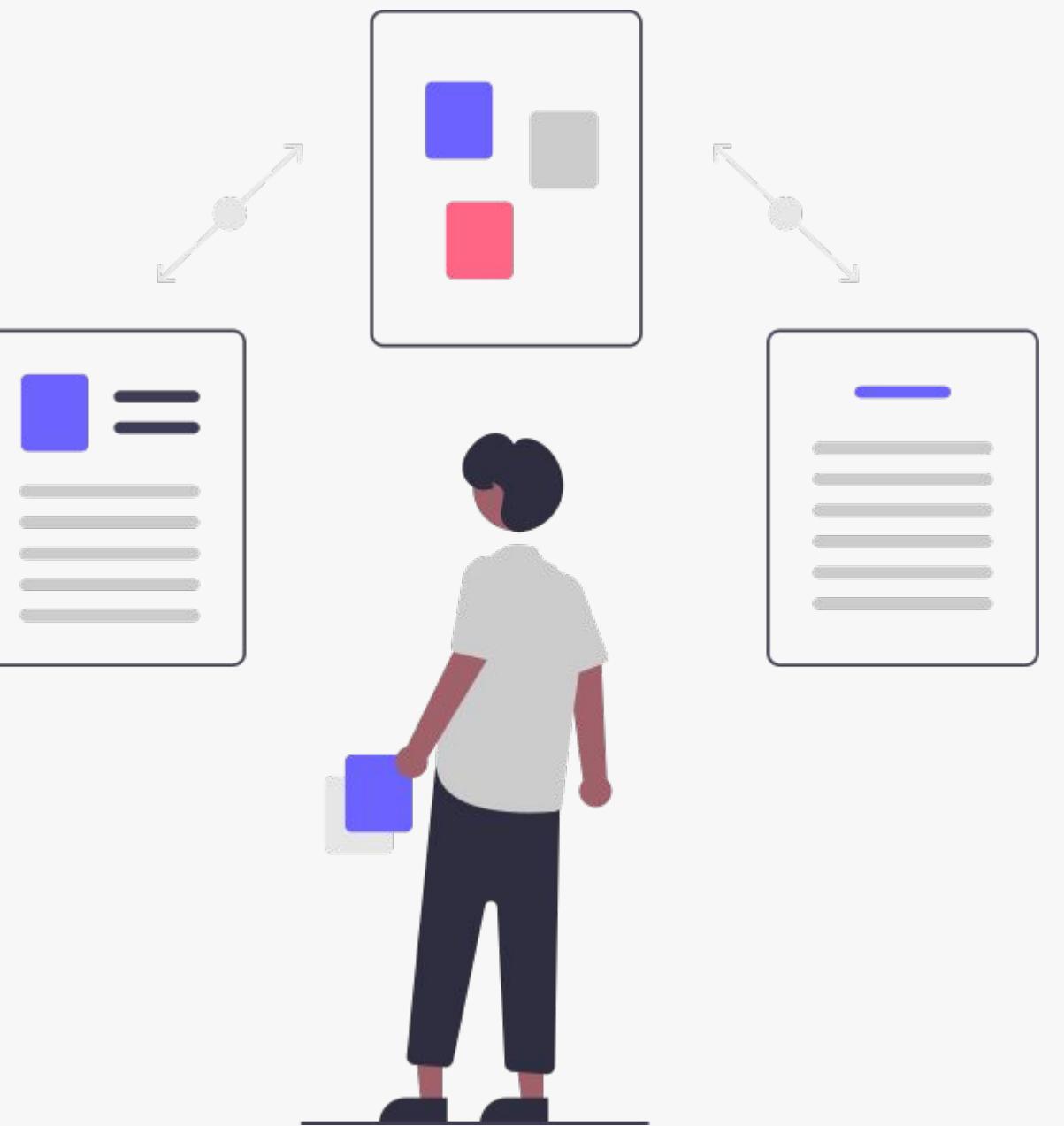
Criando um pull request

- Vamos entender na prática!
- \$ git checkout -b pull_request_test
- \$ echo "{ \"pull request\": \"atualiza\" }" >> hello.json
- \$ git add hello.json
- \$ git commit -m "pull request #1"
- \$ git push origin pull_request_test



Material de Referência

1. <https://git-scm.com/book/pt-br/v2/Fundamentos-de-Git-Gravando-Altera%C3%A7%C3%B5es-em-Seu-Reposit%C3%B3rio>
2. <https://github.com/>
3. <https://www.atlassian.com/br/git/tutorials/rewriting-history#:~:text=O%20comando%20git%20commit%20%2D%2D,anterior%20sem%20alterar%20seu%20instant%C3%A2neo.>
4. <https://git-scm.com/docs/git-reset>
5. <https://git-scm.com/book/pt-br/v2/Branches-no-Git-O-b%C3%A1sico-de-Ramifica%C3%A7%C3%A3o-Branch-e-Mesclagem-Merge>
6. <https://www.udemy.com/course/git-e-github-para-iniciantes>





Obrigado >)

contato@rubcube.com

Rua João Gonçalves Fóz, Sala 01, Jd.
Maripiara - Presidente Prudente - SP,
Cep: 19060-050.