

# Controle de Versão, Git e GitHub

---

# Controle de versão

Controles de versão são sistemas que controlam o código gerado em projetos.

Permite que você registre as mudanças feitas em um ou vários arquivos ao longo do tempo de forma que se possa recuperar versões específicas.

Permite reverter um arquivo para um estados anterior.

Permite identificar quem fez as alterações no arquivo, quando fez e o que foi feito.

São exemplos de Sistemas de Controle de Versão: Subversion, Mercurial, Microsoft Visual Studio Team Foundation Server, CVS (Concurrent Version System) e **Git**.

---

# Git

Git é um **sistema de controle de versão distribuído**, de forma que todas as máquinas que tem um repositório de git podem ser utilizadas como servidor pois elas tem o histórico completo do repositório.

O Git foi inicialmente projetado e desenvolvido por Linus Torvalds para o desenvolvimento do kernel Linux. – Wikipedia, GIT.

GIT é um software, disponível em várias plataformas, com o qual podemos manipular os arquivos que estarão no repositório.

---

# Instalação do Git

## No Linux:

Debian e Ubuntu: **sudo apt-get install git**

Fedora: **yum install git-core**

## No Windows:

**msysgit:** <http://msysgit.github.io/>

**windows.github.com**

---

# Ferramentas

GitKraken, <https://www.gitkraken.com/download>. Software pago, com versão de avaliação.

TortoiseGit

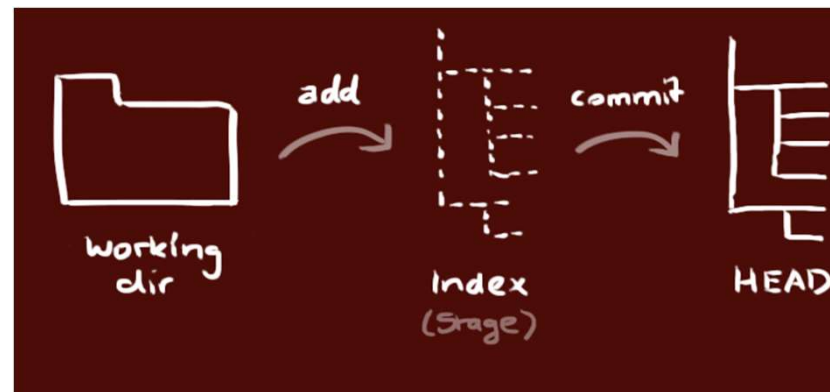
Git-Cola

---

# Áreas de operação

Os locais de operação são as áreas onde os arquivos irão transitar enquanto estão sendo editados e modificados. São 3 (três):

- Working Directory,
- Stage Area (também chamada de *Index*),
- Git directory (repositório ou *committed* ou *HEAD*).



## Áreas de operação

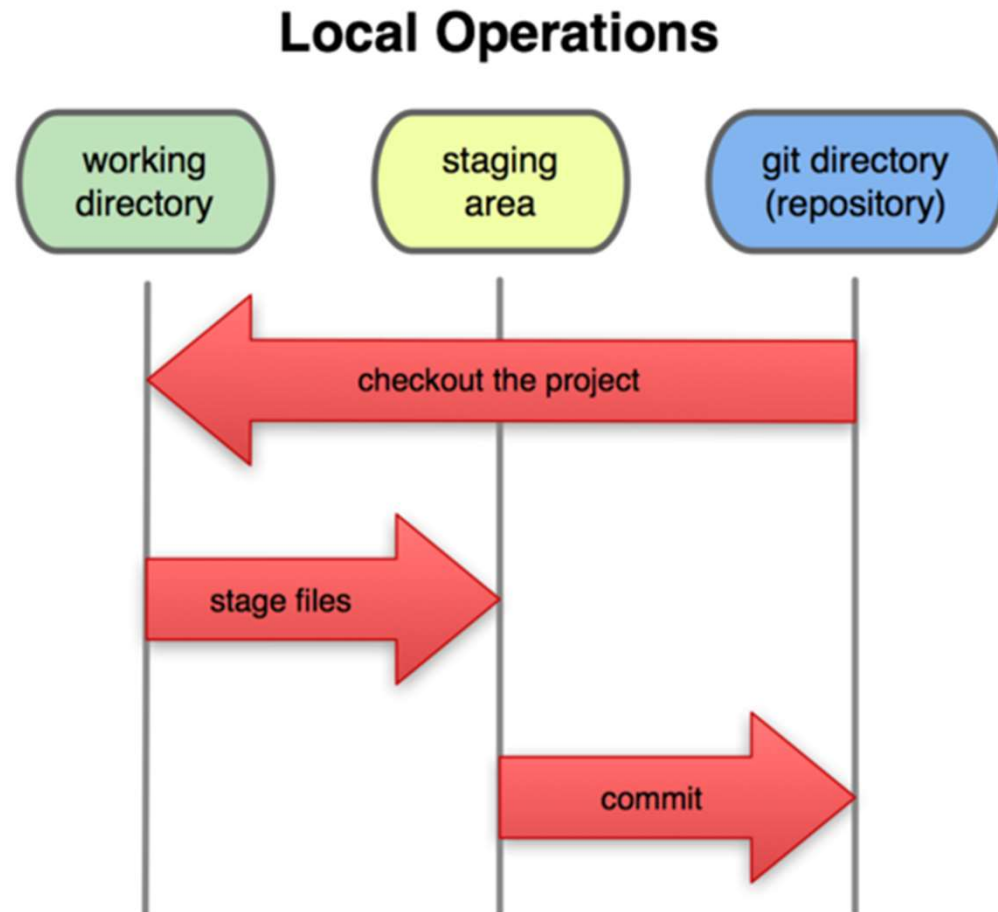
O **Git Directory (committed ou HEAD)** é onde o Git guarda os dados e objetos do seu projeto. É o diretório mais importante do Git e é ele que será copiado quando alguém *clonar* o projeto.

O **Work Directory** é onde você vai trabalhar. Os arquivos ficam aí para poderem ser usados e alterados quantas vezes quiser para você. É basicamente sua pasta de arquivos do projeto.

Quando você faz uma alteração em algum arquivo, ele vai para o **Staging Area**, que é uma área intermediária. Basicamente o Staging Area contém o Git Directory com os arquivos modificados, onde ele guarda as informações sobre o que vai no seu próximo commit.

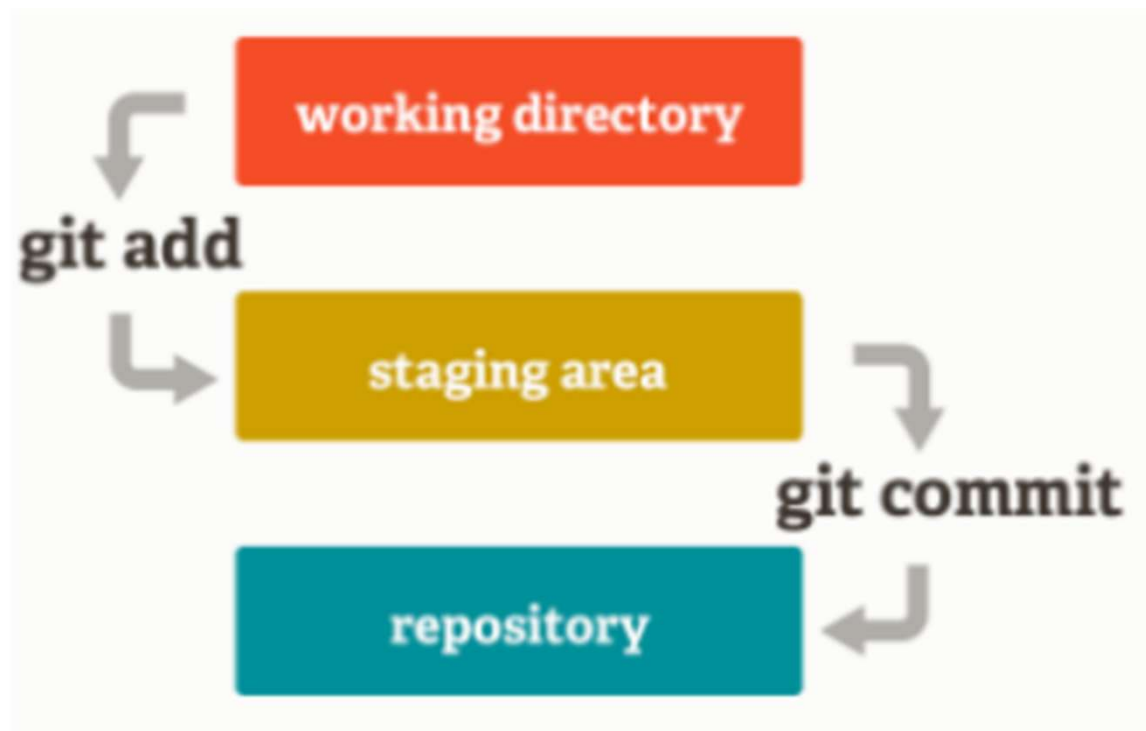
---

## Áreas de operação





## Fluxo de Trabalho



## Fluxo de operação básico

O workflow básico, após *criar/ativar* o repositório do **Git** é descrito da seguinte forma:

Você cria/modifica arquivos no seu diretório de trabalho.

Então, *seleciona* os arquivos modificados que vão ser adicionados a área de preparação.

```
git add nomeDoArquivo
```

Faz o *commit*: este comando move os arquivos da área de preparação para o Diretório Git (repositório local).

```
git commit -m "Uma mensagem de mudança"
```

---

## Configurando suas informações no git

A primeira coisa que você deve fazer depois de instalar o *Git* é definir seu **nome de usuário** e **e-mail**. À partir dessas informações é possível identificar o autor das mudanças realizadas.

```
$ git config --global user.name "Seu nome"
```

```
$ git config --global user.email "seu e-mail"
```



## git init

Cria um repositório do *git*.

Para iniciar um repositório utilize o “git init”, e pronto.

*Obs: Também temos o comando “git clone” que faz uma cópia de um repositório remoto.*

---

## git init

Indica que este projeto (pasta) será controlado pelo git, ou seja, transforma a pasta em um repositório do **git**. Passos:

1. Crie uma nova pasta e acesse a mesma (ou entre numa pasta que queira controlar)
2. Digite o comando abaixo para criar um repositório

```
git init
```

*Esse comando vai criar um diretório invisível dentro do projeto chamado **.git**, que contém todos os arquivos necessários do seu repositório.*

---

## git status

Este comando exibe informações sobre os arquivos criados e/ou modificados dentro de um repositório do git.

Com ele é possível ver os arquivos que estão fora do controle do *git*, os que foram alterados e quais estão na área de *stage*:

```
git status
```

---

## git ls-files

Mostra os arquivos que estão sob o controle (indexados) do git.

```
git ls-files
```



## git add

Adiciona o(s) arquivo(s) no estágio (*stage ou index*) de controle, ou seja, inclui o(s) arquivo(s) para o ambiente de controle do git.

```
git add .
```

```
git add nomeDoarquivo
```

---



## git commit

Gera uma versão das modificações que estão na área de *Stage*. Ou seja, neste caso o arquivo é enviado para o HEAD da sua cópia de trabalho atual, mas ainda não para o repositório remoto.

```
git commit -m "mensagem"
```

*Obs: o commit não envia as modificações para o **repositório remoto** (ver definição adiante). O envio para o repositório remoto é feito pelo comando **git push** (ver **git remote** e **git push** mais a frente).*

---

## git log

Verificando o histórico de commits.

```
git log
```

Para verificar o que foi mudado, diferença entre um arquivo e outro:

```
git log -p
```

Para visualizar um *commit* por linha

```
git log --oneline
```

---

# git whatchanged

Semelhante ao comando **git log**.



## git diff

Mostra as diferenças entre versões dos arquivos.

```
git diff
```



```
git checkout -- <arquivo>
```

Reconstrói as alterações locais usando o último commit.



## git reset

Volta ao estágio anterior

```
git reset HEAD nome-do-arquivo
```

Voltar um commit:

```
git reset HEAD~1
```

Voltar dois commits:

```
git reset HEAD~2
```

\* HEAD: representa o último commit no *ramo* atual em que se está trabalhando.

---

# Tag (rotulando)

Criar rótulos para identificar um release de software.

Antes de lançar atualizações/alterações de software, é sempre recomendado criar tags.



## Uso de Tags para marcar uma versão

As tags servem para marcar uma etapa. Imagine que você vai lançar uma versão, que resolve uma série de problemas. Você pode marcar aquela etapa criando uma tag.

Assim fica simples de fazer qualquer rollback do projeto para uma tag específica em vez de voltar para um commit. Você sabe que tudo o que foi feito até aquela tag está funcionando.





# git tag

Criando tags

```
git tag versão-da-tag
```

Listando tags:

```
git tag -l
```

Removendo as tags criadas localmente:

```
git tag -d versão-da-tag
```

---

# Branch (ramificação)

utilizados para desenvolver funcionalidades isoladas umas das outras.

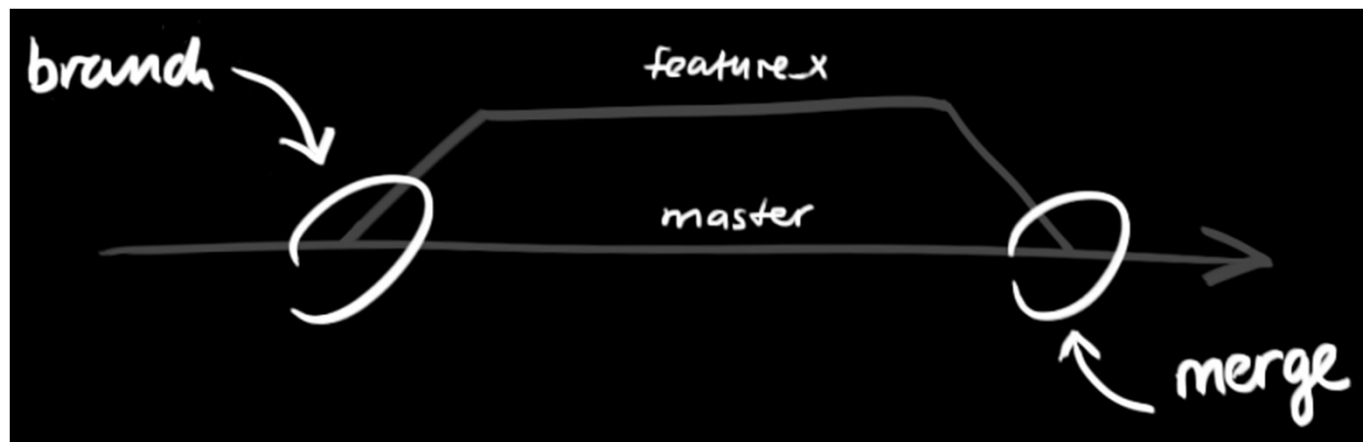
Branches ("ramos") são utilizados para desenvolver funcionalidades isoladas umas das outras. O branch master é o branch "padrão" quando você cria um repositório. Use outros *branches* para desenvolver e mescle-os (merge) ao branch master (*main*) após a conclusão.

---

## Branch (Ramificação)

Representa uma seção separada no projeto, que pode ser desenvolvida paralelamente com outras seções.

Uma seção (branch) não interfere em outra, exceto quando houver um merge.



## git branch

Permite criar um novo branch:

```
git branch nomeBranch
```

Remover um branch:

```
git branch -d nomeBranch
```

Entrar em um branch

```
git checkout nomeBranch
```

---

## git branch

Altera o nome do branch atual

```
git branch -m novonome
```



## git branch

O objetivo principal de um branch é desenvolver novas funcionalidades, mantendo-os isolados uns dos outros. O branch padrão em qualquer projeto é sempre o **master** branch.

Tantos branches quanto necessários podem ser criados e eventualmente *mesclados* ao **master** branch.

```
git merge nomeBranch
```

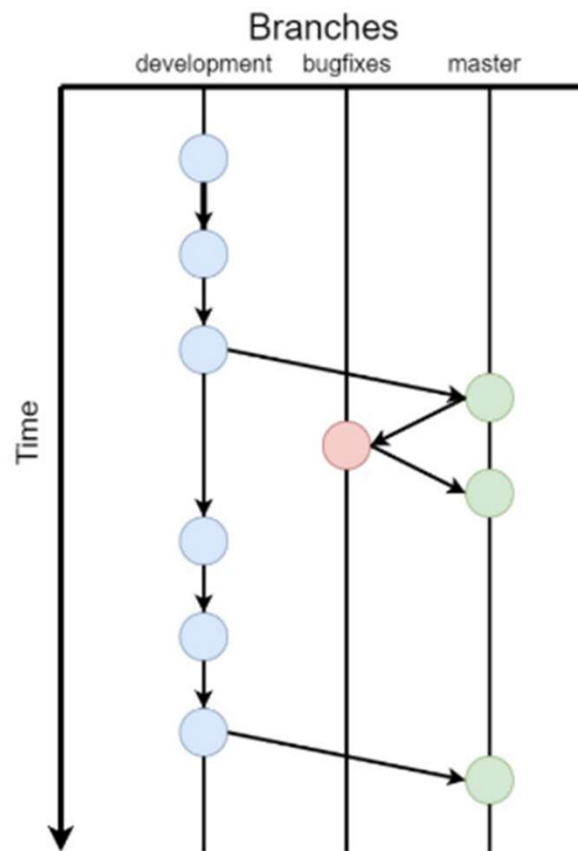
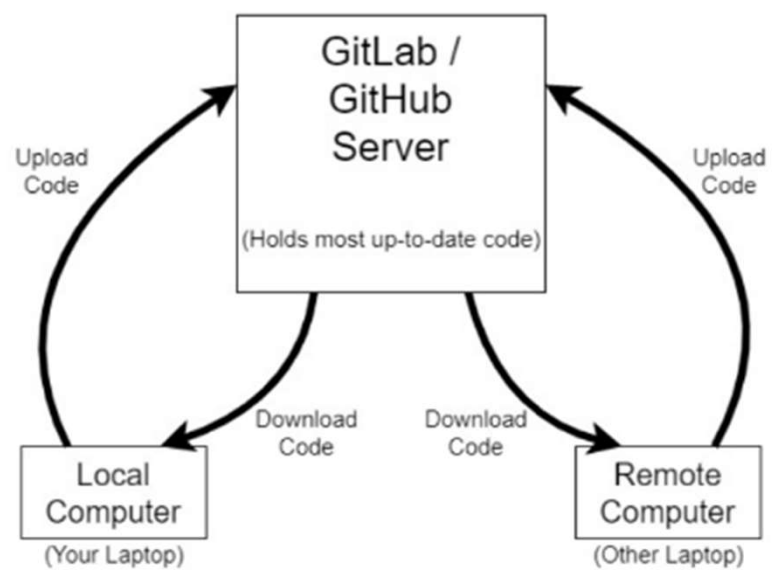


# GitHub

[www.github.com](https://www.github.com)

É uma plataforma de  
desenvolvimento colaborativo de  
software para hospedar projetos







# GitHub

O GitHub é um serviço web que provê, através de diversos planos, a possibilidade de hospedagem em um **repositório** de projetos.

**Repositório** é o nome dado a um local propício para que o projeto seja armazenado com segurança por todos os membros da equipe.



# GitHub

- Gerência usuários e permissões
- Serviço utilizado para hospedagem de repositórios remotos
- Permite criar repositórios para os quais podemos submeter nossos arquivos e outras pessoas de uma equipe possam acesso e permissão para manipulá-los de maneira colaborativa.

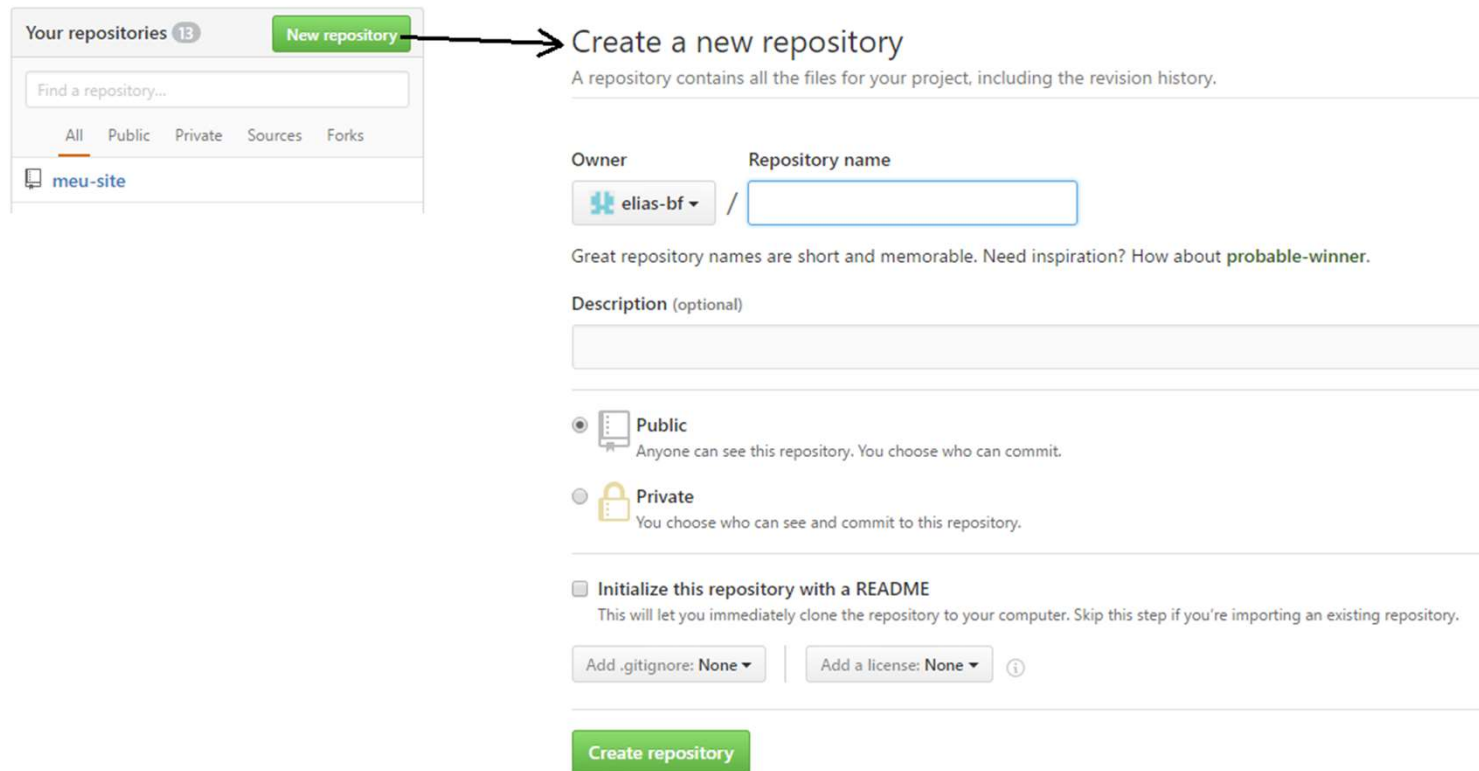


# Repositório

- Local remoto onde os arquivos ficam disponíveis para acesso aos diversos desenvolvedores
- Todo projeto deve ter seu repositório próprio para armazenar os arquivos que o compõem.
- Através do Github é possível criar ilimitados repositórios, porém todos os projetos são públicos
  - É possível ter repositórios privados, que são acessíveis por planos pagos.



# Criar um repositório (é necessário ter um conta)



**Your repositories** 13 **New repository**


Find a repository...

All Public Private Sources Forks

meu-site

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner:  elias-bf /

Repository name

Great repository names are short and memorable. Need inspiration? How about **probable-winner**.

Description (optional)

☒ **Public**  
Anyone can see this repository. You choose who can commit.

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

☐ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** ⓘ

**Create repository**

## git remote add

```
git remote add [alias_do_repositorio] [uri_do_repositorio]
```

É convenção utilizar o alias\_do\_repositório como “**origin**”

A *uri\_do\_repositório* você poderá copiá-la do próprio **github**, por exemplo:

```
git@github.com:[nome_do_usuario]/[nome_do_repositorio].git
```

---

## `git remote -v`

Mostra os repositórios remotos que foram adicionados ao projeto.



## git push

Envia os commits locais para o repositório remoto. Sua sintaxe é:

```
git push origin master
```

Onde:

- **origin**: nome do repositório remoto;
- **master** (ou **main**): nome do *branch* que será enviado ao repositório remoto.

Se usar o **-u** como argumento, no próximo *push* o nome do *branch* e do *repositório* não precisa ser especificado.

```
git -u push origin master
```

---

## git clone

Permite que outros desenvolvedores obtenham cópias de projetos (repositórios) que estão no github para a máquina local:

```
git clone [uri_do_repositorio]
```

quando usar um servidor remoto, seu comando será

```
git clone usuário@servidor:[uri_do_repositorio]
```

---



## git pull

Para atualizar seu repositório local com a mais nova versão do repositório remoto utilize o comando git pull:

```
git pull
```

ou

```
git pull origin master
```



## git fetch e git merge

O *git fetch* faz o download de todos o histórico de alterações do repositório remoto, em seguida é necessário executar o *git merge* para juntar os dados ao repositório local.

$\$git\ pull = \$git\ fetch + \$git\ merge$

---

# Permitindo contribuição de terceiros

The screenshot shows the GitHub interface for a repository named 'meu-site' by user 'elias-bf'. The top navigation bar includes links for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Pulse', 'Graphs', and 'Settings' (circled in yellow). On the right, there are buttons for 'Unwatch', 'Star', and 'Fork'. The left sidebar contains a list of settings: 'Options', 'Collaborators' (circled in yellow), 'Branches', 'Webhooks', 'Integrations & services', and 'Deploy keys'. The main content area is titled 'Collaborators' and includes a sub-header 'Push access to the repository'. It states: 'This repository doesn't have any collaborators yet. Use the form below to add a collaborator.' Below this is a search bar with the placeholder text 'Search by username, full name or email address'. The search bar contains the text 'James Gosling' (circled in yellow). To the right of the search bar is a button labeled 'Add collaborator' (circled in yellow). Below the search bar, a search result is displayed: a blue bar with a GitHub logo, the username 'jgosling', and the full name 'James Gosling (acn)'.

elias-bf / meu-site

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs **Settings**

Options

**Collaborators**

Branches

Webhooks

Integrations & services

Deploy keys

Collaborators Push access to the repository

This repository doesn't have any collaborators yet. Use the form below to add a collaborator.

Search by username, full name or email address

James Gosling

**Add collaborator**

**jgosling** James Gosling (acn)

try.github.io



## Referências

Iniciando no Git, <https://tableless.com.br/iniciando-no-git-parte-1/>.

Comandos iniciais do git: <https://tableless.com.br/alguns-comandos-git/>.

git - guia prático: [https://rogerdudler.github.io/git-guide/index.pt\\_BR.html](https://rogerdudler.github.io/git-guide/index.pt_BR.html)

GIT Tutorial Para Iniciantes: <https://www.hostinger.com.br/tutoriais/tutorial-do-git-basics-introducao>

Guia rápido: <https://dev.to/womakerscode/git-e-github-guia-rapido-e-comandos-basicos-para-iniciantes-4ile>

---