

GESTÃO DE CONTEÚDO E VERSIONAMENTO

PT-1



Prof. Alexandre Carlos

profalexandre.jesus@fiap.com.br

Prof. Luís Carlos

lsilva@fiap.com.br

Prof. Wellington Cidade

profwellington.tenqrio@fiap.com.br



Importante!!!



git



github
SOCIAL CODING

O que é Git?



Git : É um sistema de controle de versão distribuído. Ele permite que os desenvolvedores acompanhem as mudanças no código ao longo do tempo, colaborando em projetos de forma eficiente, mesmo sem uma conexão constante com um servidor central.

O Git foi criado por Linus Torvalds (o criador do Linux) em 2005, para substituir sistemas de controle de versão anteriores que não eram tão eficientes para o desenvolvimento distribuído.

O que é Github?



É um serviço web utilizado para compartilhar projetos que utilizam o Git para versionamento, se tornando assim uma rede social para desenvolvedores.

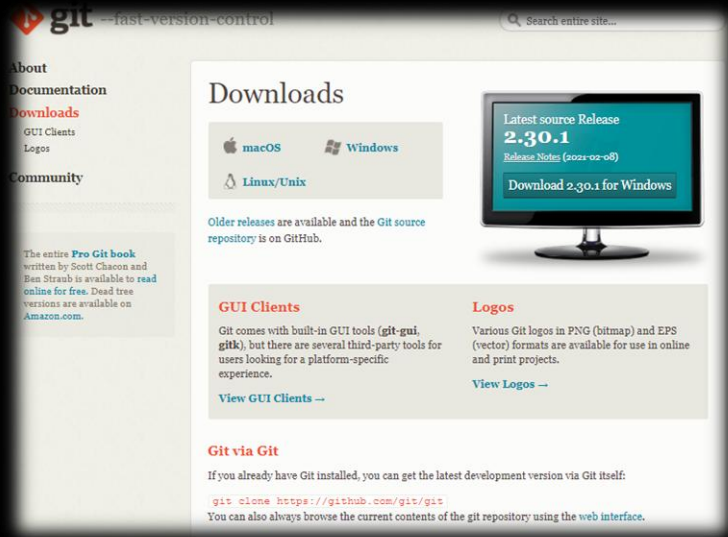
Sua função principal é armazenar projetos na web versionados pelo Git.



Instalação do Git

A instalação do Git é muito simples, basicamente é só ir seguindo as orientações do aplicativo de instalação e ir dando Next.

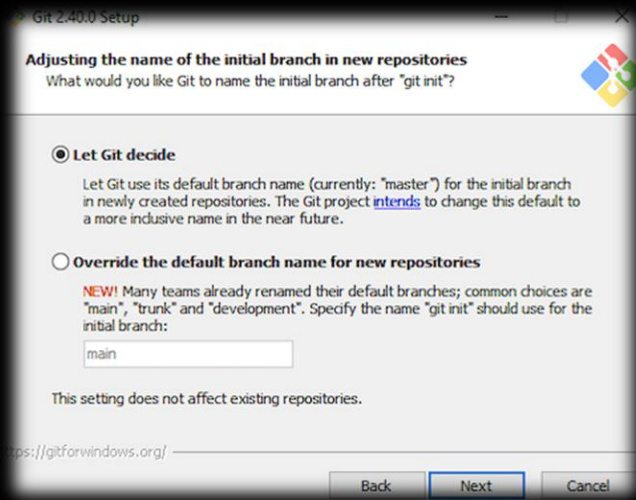
Para fazer o download é só acessar o endereço: <https://git-scm.com/download>





Instalação do Git

Em um certo ponto da instalação se atente para o detalhe abaixo onde vamos modificar o nome da Branch padrão sempre que carregarmos o terminal.



Selecione esta opção e coloque o nome padrão como main.

Dessa maneira, não teremos problemas ao criarmos nossos repositórios no Github, onde o padrão de nome para a Branch principal é main.



Configurações básicas do Git

Quando começamos a utilizar o Git pela primeira vez em nossa máquina ou vamos utilizar uma máquina compartilhada com outras pessoas, é importante fazer algumas configurações básicas como:

Nome do Usuário:

```
Luis@PC-LUIS MINGW64 ~/Desktop (master)
$ git config --global user.name "lcsilva76"
```

E-mail do Usuário:

```
Luis@PC-LUIS MINGW64 ~/Desktop (master)
$ git config --global user.email lcsilva@git.com
```



Configurações básicas do Git

Para verificar as configurações atuais use:

Nome do Usuário: Luis@PC-LUIS MINGW64 ~/Desktop (master)
\$ git config user.name

lcsilva76

Resposta

E-mail do Usuário: Luis@PC-LUIS MINGW64 ~/Desktop (master)
\$ git config user.email

lcsilva@git.com

Resposta



Repositórios

Para que o Git possa fazer o controle dos nossos projetos, precisamos identificar a sua pasta (diretório) como um repositório, assim ele sabe que deve monitorar e fazer o controle dela.

Para vermos como funciona, crie uma pasta no seu desktop chamada “exemplo-git”;

Você pode fazer direto de dentro do git bash (terminal) usando o comando:
`mkdir exemplo-git`

Acesse a pasta com o comando: `cd exemplo-git`

Para inicializar a pasta como repositório digite: `git init`

Para visualizar o conteúdo do repositório digite: `ls -la`



Repositórios

Agora que já temos o nosso repositório configurado vamos criar o seu primeiro arquivo, na pasta Exemplo-Git. Crie um arquivo chamado “primeiro.txt”, você pode utilizar o bloco de notas para isso, escreva nele “Arquivo de teste” e salve o arquivo.

Agora no terminal digite ls, este comando lista do diretório.

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ ls
primeiro.txt
```



Ciclo de vida / Estado dos arquivos

Essa imagem ilustra o ciclo de vida dos arquivos no Git, explicando os diferentes estados que um arquivo pode assumir durante o controle de versão. Aqui está a explicação:



Untracked (Não rastreado):

Arquivos que foram criados, mas ainda não estão sendo rastreados pelo Git. Para começar a rastrear esses arquivos, você deve usar o comando `git add`.

Unmodified (Sem modificações):

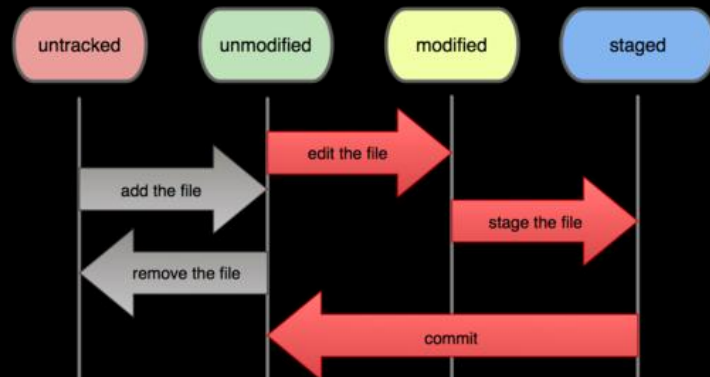
São arquivos já rastreados pelo Git e que não possuem alterações em relação ao último commit. Se você os modificar, eles passarão para o estado Modified.

Modified (Modificado):

Esses arquivos foram alterados, mas as mudanças ainda não foram enviadas para a área de stage (staging area). Para prepará-los para o próximo commit, você deve usar o comando `git add`.

Staged (Preparado):

Arquivos que estão prontos para serem confirmados (commitados). Quando você executa `git commit`, as alterações desses arquivos são salvas no histórico do Git e eles voltam ao estado Unmodified.





Ciclo de vida / Estado dos arquivos

Fluxos principais:

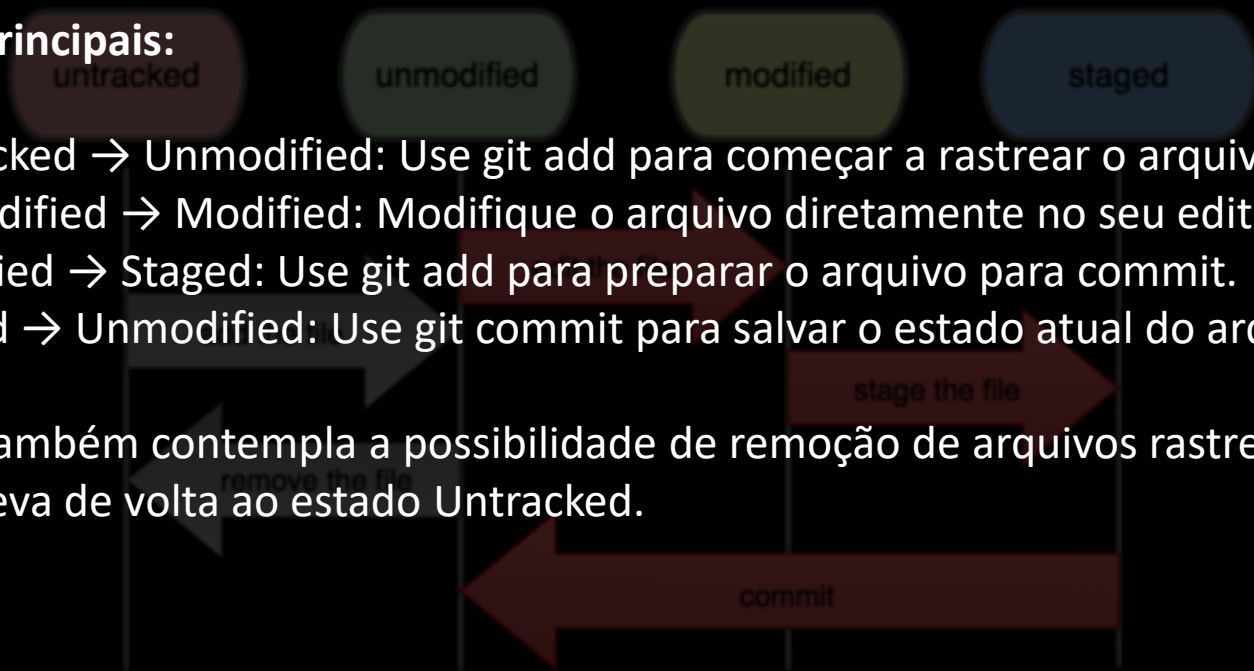
Untracked → Unmodified: Use git add para começar a rastrear o arquivo.

Unmodified → Modified: Modifique o arquivo diretamente no seu editor.

Modified → Staged: Use git add para preparar o arquivo para commit.

Staged → Unmodified: Use git commit para salvar o estado atual do arquivo.

O ciclo também contempla a possibilidade de remoção de arquivos rastreados, que os leva de volta ao estado Untracked.





Estados dos Arquivos

Para sabermos qual o status dos arquivos use o comando: **git status**

```
Luís@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git status
On branch master

No commits yet

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

    primeiro.txt

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

Perceba que ele localizou o nosso arquivo e está avisando que ele ainda não está sendo rastreado pelo git **“UNTRACKED”**.



Estados dos Arquivos

Agora vamos adicionar este arquivo ao controle de rastreamento do Git, digite:

git add primeiro.txt

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git add primeiro.txt
```

Agora: **git status**

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   primeiro.txt
```

Este é o estado de **"UNMODIFIED"** , ele está pronto para ser versionado.



Estados dos Arquivos

Vamos fazer uma alteração no conteúdo do nosso arquivo, insira mais uma linha de texto, escreva por exemplo: Fiz uma alteração. Agora salve o arquivo e digite **git status** no terminal.

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   primeiro.txt

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:   primeiro.txt
```

Nosso arquivo continua sendo rastreado, mas agora ele diz que foi modificado **“MODIFIED”** e precisa ser adicionado novamente para ser versionado.



Git ADD

Para fazer nosso primeiro commit (termo usado para versionamento) adicione novamente no modo STAGE usando o comando **git add primeiro.txt**.

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git add primeiro.txt
```

Agora para realizarmos o commit use o comando: **git commit -m "Add primeiro.txt"**

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git commit -m "Add primeiro.txt"
[master (root-commit) 1a80a56] Add primeiro.txt
1 file changed, 2 insertions(+)
create mode 100644 primeiro.txt
```

Vamos ver se funcionou? Digite **git status** novamente:

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git status
On branch master
nothing to commit, working tree clean
```




Git LOG

Usamos o comando **git log** para visualizar os commits feitos em nosso repositório.

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git log
commit 1a80a56f9c0d5c0b102fb8154330a548144170c7 (HEAD -> master)
Author: lcsilva76 <lcsilva@fiap.com.br>
Date:   Sun Feb 28 12:03:20 2021 -0300
```

Nele podemos visualizar todos os commits feitos e suas informações principais.

Utilizando a hash (código de identificação do commit), podemos ver as alterações feitas naquele commit, digite **git show 1a80a56f9c0d5c0b102fb8154330a548144170c7**

```
diff --git a/primeiro.txt b/primeiro.txt
new file mode 100644
index 0000000..e378cf5
--- /dev/null
+++ b/primeiro.txt
@@ -0,0 +1,2 @@
+Arquivo de teste
+Fiz uma alteração
\ No newline at end of file
```



Git COMMIT

Muitas vezes podemos nos arrepender de alterações que fizemos em algum arquivo e querer se desfazer delas antes de fazer o commit. Para ver o que foi alterado em um arquivo desde o último commit pode usar o comando **git diff**

```
index e378cf5..da43939 100644
--- a/primeiro.txt
+++ b/primeiro.txt
@@ -1,2 +1,3 @@
  Arquivo de teste
- Fiz uma alteração
\ No newline at end of file
+ Fiz uma alteração
+ teste diff e checkout
\ No newline at end of file
```

ATENÇÃO

Este processo só pode ser feito se o commit ainda não foi realizado!!!

Agora que temos certeza de que foi alterado, vamos desfazer usando o comando **git checkout primeiro.txt**

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git checkout primeiro.txt
```

Se fecharmos e abrirmos novamente o nosso arquivo primeiro.txt você verá que as mudanças foram desfeitas



Git CHECKOUT

O comando git checkout tem diferentes usos, dependendo do contexto. Aqui estão os principais:
git checkout

1 - Trocar de Branch: Isso muda para a branch minha-branch. `git checkout minha-branch`

Obs: Atualmente é possível utilizar o comando git switch para trocar de branch: `git switch minha-branch`

2 - Restaurar um arquivo para o estado do último commit: `git checkout -- meu-arquivo.txt`

Isso descarta todas as mudanças não commitadas do arquivo meu-arquivo.txt, restaurando-o para a última versão commitada.

Obs: Atualmente é o comando equivalente seria: `git restore meu-arquivo.txt`

3 - Criar e trocar para uma nova branch (antigo método) : `git checkout -b nova-branch`

Isso cria e já muda para a branch nova-branch.

Obs: Atualmente é o comando equivalente seria: `git switch -c nova-branch`



Git RESET HEAD

Outra coisa que podemos querer desfazer é mandar o arquivo para o stage, para fazermos isso podemos usar o comando **git reset HEAD primeiro.txt**.

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git reset HEAD primeiro.txt
Unstaged changes after reset:
M      primeiro.txt
```

ATENÇÃO

Este processo só pode ser feito se o commit ainda não foi realizado!!!

Então agora, se usarmos o **git diff** novamente as alterações irão aparecer.

```
@@ -1,2 +1,3 @@
  Arquivo de teste
-Fiz uma alteração
\ No newline at end of file
+Fiz uma alteração
+Outra alteração.
\ No newline at end of file
```



Git RESET

E se precisarmos voltar um commit?

Para esse problema temos 3 alternativas:

Git reset --soft (hash) = ele volta o commit, mas os arquivos continuam prontos para serem comitados novamente;

Git reset --mixed (hash) = ele volta o commit, mas os arquivos saem da área de staged;

Git reset --hard (hash) = ele volta o commit, também exclui todas as alterações;

Obs. *O hash deve ser o do commit para que queremos retornar*



Git RESET

(HEAD) - O que é o HEAD no Git?

O HEAD no Git é um ponteiro especial que aponta para o commit atual em que você está trabalhando.

1 - O HEAD aponta para a branch ativa

- ◆ Normalmente, o HEAD aponta para a última versão da branch em que você está.
- ◆ Se você estiver na branch main, HEAD estará apontando para ela:

```
git branch
* main  <- HEAD está aqui
dev
```

◆ Ou seja, HEAD é um ponteiro para um ponteiro (ele aponta para a branch, que por sua vez aponta para um commit).

- ◆ Se você fizer um novo commit, ele será anexado na branch atual e HEAD se moverá junto.



HEAD

(HEAD) - O que é o HEAD no Git?

2 - HEAD no modo "detached" (solto)

- ◆ Se você fizer checkout de um commit específico (e não de uma branch), HEAD fica desanexado (detached).

```
git checkout abc1234
```

- ◆ Agora, HEAD está apontando diretamente para o commit abc1234, e não para uma branch:

```
git branch
  main
  dev
* (detached from abc1234) <- HEAD está solto!
```

👉 **O perigo aqui:**

- ◆ Se você fizer commits nesse estado e depois mudar de branch, pode perder esses commits, pois eles não pertencem a nenhuma branch.

- ◆ Para evitar isso, se quiser salvar suas mudanças, crie uma branch antes de sair do modo detached:

```
git checkout -b nova-branch
```



HEAD

(HEAD) - O que é o HEAD no Git?

3 - Como usar HEAD para navegar no histórico

- ◆ Voltar um commit (HEAD~1):

```
git checkout HEAD~1
```

- ◆ Isso faz HEAD apontar para o commit anterior ou pode voltar vários commits:

```
git checkout HEAD~3 # Volta 3 commits
```

- ◆ Resetar HEAD para um commit específico:

```
git reset --soft HEAD~2
```




HEAD

(HEAD) - O que é o HEAD no Git?

Resumo

HEAD aponta para a branch atual.

HEAD pode ficar "solto" (detached HEAD) se você fizer checkout de um commit.

Você pode usar HEAD~N para navegar pelo histórico.



Git RESET SOFT

git reset --soft (hash)

- O que faz: Restaura o ponteiro do HEAD para um commit específico, mas não altera o índice (staging area) nem o diretório de trabalho.
- Efeito: As mudanças no commit descartado são mantidas no staging area. Ou seja, após o reset, você ainda terá os arquivos prontos para serem adicionados ao próximo commit, sem perder as modificações feitas.
- Quando usar: Quando você deseja reverter o ponteiro do commit, mas quer manter as alterações para continuar trabalhando nelas ou eventualmente fazer um novo commit.
- Exemplo:

```
git reset --soft HEAD~1
```



Git RESET MIXED

git reset --mixed (hash) *é o padrão a ser utilizado*

- O que faz: Restaura o ponteiro do HEAD para um commit específico e altera o índice (staging area), mas não altera o diretório de trabalho.
- Efeito: Após o reset, as mudanças dos commits removidos são movidas para o diretório de trabalho, ou seja, as alterações não são mais rastreadas pelo Git, mas ainda estão nos arquivos. Elas não estão no staging, então você precisa adicionar novamente ao staging (git add) para preparar para um novo commit.
- Quando usar: Quando você quer reverter os commits, mas ainda quer manter as alterações no diretório de trabalho para revisão ou novos ajustes antes de um novo commit.
- Exemplo:

```
git reset --mixed HEAD~2
```



Git RESET HARD

git reset --hard (hash) é o padrão a ser utilizado

- O que faz: Restaura o ponteiro do HEAD para um commit específico e limpa tanto o índice quanto o diretório de trabalho, descartando todas as alterações.
- Efeito: Todas as mudanças locais (modificações no diretório de trabalho e no staging) serão perdidas permanentemente. O repositório é restaurado para o estado do commit especificado.
- Quando usar: Quando você deseja descartar todas as mudanças feitas localmente e voltar para um estado específico do repositório, sem manter nenhuma alteração.

- Exemplo:

```
git reset --hard HEAD~1
```



Git REBASE

Se você apagar um commit antigo com rebase, pode afetar commits futuros ?

- **O que é:** O git rebase é um comando poderoso e útil no Git, utilizado para aplicar mudanças de uma branch em outra, reescrevendo o histórico de commits. Ao invés de fazer uma fusão (merge), o rebase aplica as alterações de uma branch em cima de outra, criando um histórico linear. Isso pode resultar em um histórico mais limpo e organizado, sem os commits de merge, mas também pode ser perigoso se não for utilizado corretamente.
- **Como funciona?** Quando você faz um git rebase, o Git pega os commits da sua branch atual (onde você está trabalhando) e os aplica em cima de outro commit (geralmente o HEAD de outra branch, como a main ou master). Esse processo é uma forma de reescrever o histórico, porque os commits que estavam em uma branch passam a ser "reaplicados" em outro ponto do histórico, criando novos commits com novos hashes.



Git REBASE

Fluxo básico do git rebase

1. **Mude para a branch que você deseja atualizar** (geralmente sua branch de desenvolvimento):

```
git checkout minha-branch
```

2. **Rebase em outra branch** (geralmente a branch principal, como main ou master):

```
git rebase main
```

3. **Esse comando vai pegar os commits da sua branch (minha-branch) e os aplicar em cima do commit mais recente da main.** O Git vai tentar fazer isso de forma automática, mas se houver conflitos, você precisará resolvê-los manualmente.



Git REBASE

Tipos de git rebase

1. **Rebase Interativo (`git rebase -i`)** : O rebase interativo permite que você edite, combine ou até remova commits durante o processo de rebase. É muito útil para limpar o histórico de commits antes de enviar seu código para um repositório remoto.

Exemplo: `git rebase -i HEAD~3`

Isso abre um editor onde você pode interagir com os últimos 3 commits (por exemplo, podendo reordená-los, alterá-los ou combiná-los).

2. **Rebase Automático** : O rebase automático tenta aplicar todos os commits em sequência, criando uma linha de tempo mais limpa, sem a necessidade de fusões (merge).

Exemplo: `git rebase master`



Git REBASE

Porque usar git rebase

1. Benefícios do git rebase :

- Histórico mais limpo: Como não há commits de merge, o histórico do projeto fica linear e mais fácil de ler.
- Facilidade de revisão: Com um histórico linear, é mais fácil para os outros membros da equipe revisarem seu código e entenderem o que foi feito.
- Integração contínua: O rebase ajuda a manter a integração contínua sem a necessidade de fazer merge frequentemente, o que pode resultar em merges complicados e históricos mais confusos.

2. Cuidados ao usar git rebase :

- Rebase em branches compartilhadas: Evite usar git rebase em branches que outras pessoas já estão usando ou trabalhando, porque o rebase reescreve o histórico de commits. Isso pode criar conflitos difíceis de resolver e confusão nos outros colaboradores.
- Preserve o histórico de commits: O rebase pode alterar os hashes dos commits, então ele não deve ser usado em commits que já foram enviados para repositórios remotos, a menos que você tenha certeza de que ninguém mais está usando essa branch.

3. Exemplo de git rebase

Imagine que você tenha a seguinte situação: main tem os commits: A - B - C

- Sua branch feature tem os commits: D - E
- Se você fizer um git rebase main enquanto está na branch feature, o Git vai tentar aplicar os commits D e E em cima de C, resultando em algo assim: main: A - B - C - D' - E'
- Observe que os commits D e E foram aplicados de novo e seus hashes mudaram para D' e E' (porque são novos commits).



Git REBASE

Se você apagar um commit antigo com rebase, pode afetar commits futuros ?

- O que faz: Restaura o ponteiro do HEAD para um commit específico e limpa tanto o índice quanto o diretório de trabalho, descartando todas as alterações.
- Efeito: Todas as mudanças locais (modificações no diretório de trabalho e no staging) serão perdidas permanentemente. O repositório é restaurado para o estado do commit especificado.
- Quando usar: Quando você deseja descartar todas as mudanças feitas localmente e voltar para um estado específico do repositório, sem manter nenhuma alteração.

- Exemplo:

```
git reset --hard HEAD~1
```

Github – Repositório Remoto

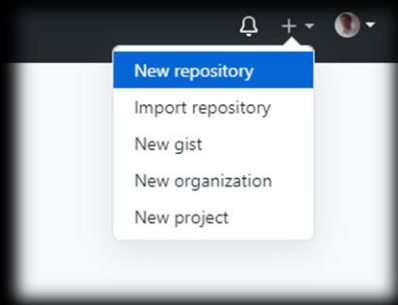




Criando o Primeiro Repositório Remoto

Agora vamos criar um repositório remoto para guardar o nosso repositório local

Na barra de Menu do Github no canto superior direito, ao lado da sua foto, tem um símbolo de mais. Clique nele e escolha a opção **New Repository**.





Criando o Primeiro Repositório Remoto


Agora temos que colocar as seguintes informações:

O nome do repositório remoto;

Uma descrição sobre o repositório, Algo breve, mas que identifique a razão dele;


Se ele será público ou privado, vamos trabalhar com repositórios públicos.


Owner ^{*} Repository name ^{*}

 lcsilva76

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

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.](#)

Create repository



Criando o Primeiro Repositório Remoto

Quick setup — if you've done this kind of thing before

Set up in Desktop or **HTTPS** **SSH** <https://github.com/lcsilva76/Exemplo-Git.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# Exemplo-Git" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/lcsilva76/Exemplo-Git.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/lcsilva76/Exemplo-Git.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Agora que já temos o nosso repositório remoto, vamos subir nosso projeto para ele:

Esta é a opção para subirmos nosso projeto, copie a primeira linha e Execute em seu terminal.

Ainda no terminal digite **git remote** para verificar se já estão ligados.
Deverá aparecer a resposta **origin**.



Criando o Primeiro Repositório Remoto

Para subir nosso projeto digite a última no terminal:

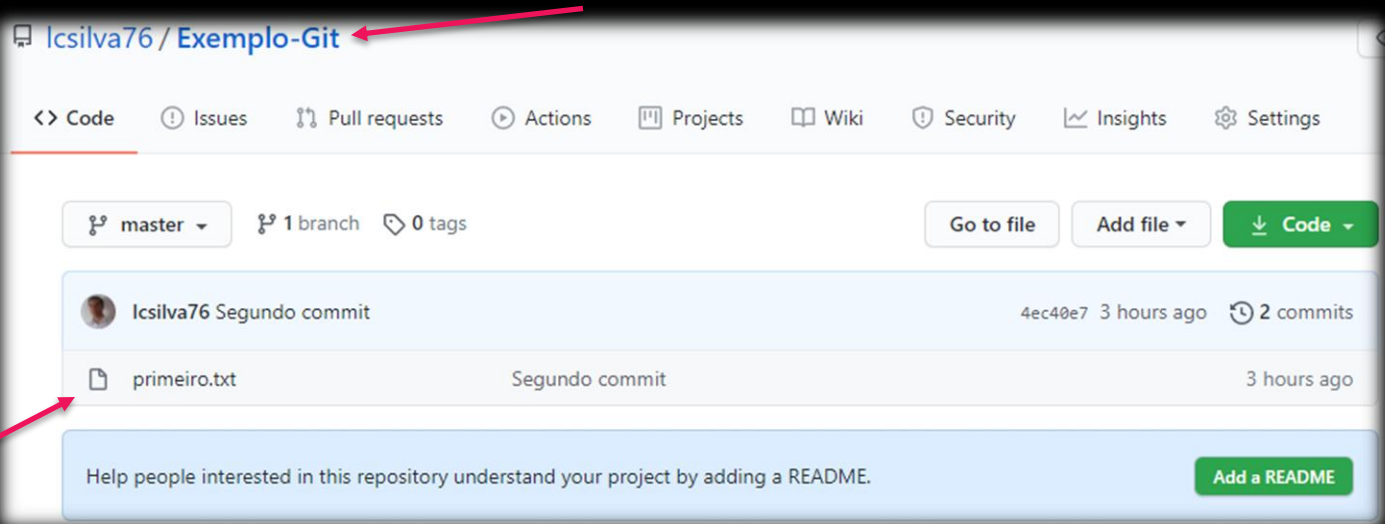
git push -u origin master

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git push -u origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 510 bytes | 255.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/lcsilva76/Exemplo-Git.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```



Criando o Primeiro Repositório Remoto

Pronto, nosso projeto já está em um repositório remoto!!!





Subindo Modificações no Repositório Remoto

Para esta parte, vamos criar um novo arquivo no nosso repositório chamado **segundo.txt**, logo após digite algo dentro e salve.

Agora adicione todos para o stage, digitando **git add *** (o asterisco significa todos)

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git add *
```

Agora faça o commit para atualizar as informações do repositório: **git commit -m "Add segundo.txt"**.

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git commit -m "Add segundo.txt"
[master f20a77a] Add segundo.txt
1 file changed, 1 insertion(+)
create mode 100644 segundo.txt
```




Subindo Modificações no Repositório Remoto

Para esta parte, vamos criar um novo arquivo no nosso repositório chamado **segundo.txt**, logo após digite algo dentro e salve.

Agora adicione todos para o stage, digitando **git add *** (o asterisco significa todos)

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git add *
```

Temos que fazer o commit para atualizar as informações do repositório: **git commit -m "Add segundo.txt"**.

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git commit -m "Add segundo.txt"
[master f20a77a] Add segundo.txt
1 file changed, 1 insertion(+)
create mode 100644 segundo.txt
```

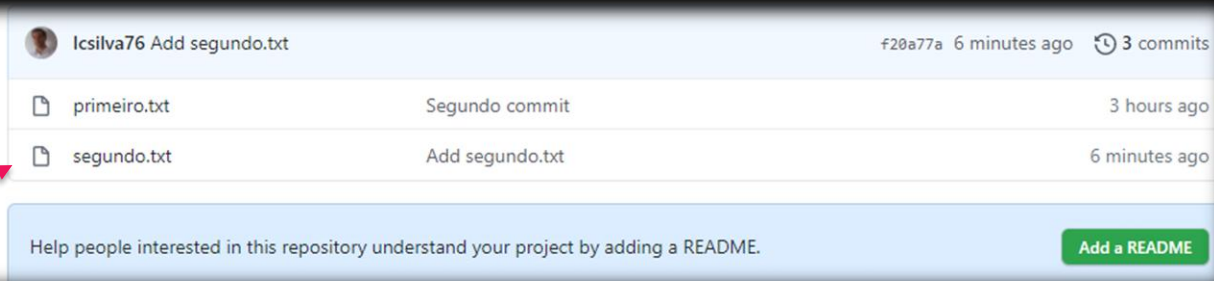


Subindo Modificações no Repositório Remoto

Agora para subir use o comando: `git push origin master`

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 309 bytes | 154.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/lcsilva76/Exemplo-Git.git
4ec40e7..f20a77a master -> master
```

Pronto, é só dar um refresh na página e ver o nosso arquivo `segundo.txt` adicionado!



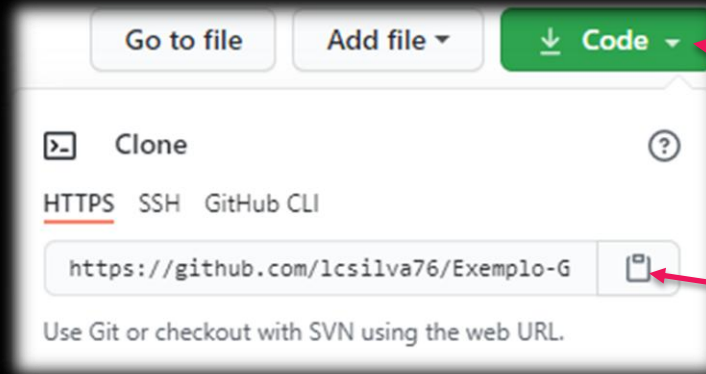


Clonando um Repositório Remoto

É muito comum termos que trabalhar em projetos com várias pessoas para isso precisamos clonar (fazer uma cópia) do projeto em nossa máquina.

Vamos fazer um teste em nosso próprio projeto, siga os passo:

1 – No nosso repositório do github clique no botão **Code** e em seguida, **copie a URI** que está nele, é a URI do seu projeto.





Clonando um Repositório Remoto

2 – Voltando para o nosso terminal, vamos sair da pasta atual e criar outra para nosso clone.

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ cd ..
```

← Sai da pasta atual

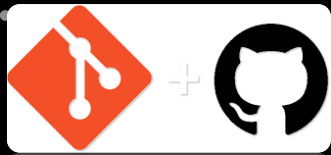
3 – Para fazer o clone na nova pasta digite: **git clone (URI copiada) Exemplo-Github-Clone**

```
Luis@PC-LUIS MINGW64 ~/Desktop (master)
$ git clone https://github.com/lcsilva76/Exemplo-Git.git Exemplo-Github-Clone
Cloning into 'Exemplo-Github-Clone'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.
```

4 – Agora abra a pasta com o comando **cd Exemplo-Github-Clone** e use o comando **ls** para conferir se os arquivos foram clonados.

```
Luis@PC-LUIS MINGW64 ~/Desktop (master)
$ cd Exemplo-Github-Clone

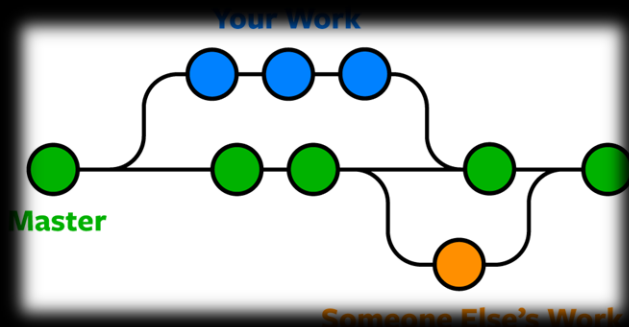
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Github-Clone (master)
$ ls
primeiro.txt  segundo.txt
```



O que é uma Branch?

É uma duplicação do projeto, gerando uma ramificação do projeto principal, permitindo ter um ou mais pessoas trabalhando nesses ramos. Assim no momento oportuno podem juntar suas contribuições ao projeto principal.

- Assim podemos:
- Trabalhar no projeto sem afetar o principal;
- Você pode apagar sem comprometer o projeto;
- Várias pessoas trabalhando
- Gerenciar os conflitos





Como criar uma Branch?

Vamos para nosso projeto Exemplo-Github e criar nossa primeira branch:

- Para sair do projeto atual: **cd ..** (voltamos para o Desktop)

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Github-Clone (master)
$ cd ..
```

- Para entrar no Exemplo-Github: **cd Exemplo-Git**

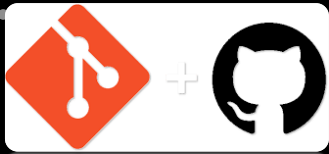
```
Luis@PC-LUIS MINGW64 ~/Desktop (master)
$ cd Exemplo-Git
```

- Criando a branch: **git checkout -b novaBranch**

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git checkout -b novaBranch
Switched to a new branch 'novaBranch'
```

- Digite **git branch**, ele irá mostrar as branches que você possui. A que está com asterisco é a atual.

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (novaBranch)
$ git branch
master
* novaBranch
```



Deletando ou mudando de Branch

Temos agora a **branch master** e uma ramificação a **novaBranch**, para navegarmos entre elas usamos o comando: **git checkout master** (repare que para navegar não usamos o **-b**)

Para apagarmos uma branch usamos o comando: **git branch -D novaBranch** (você não pode estar na pasta dela no momento).

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git branch -D novaBranch
Deleted branch novaBranch (was f20a77a).
```



MERGE – Juntando branches

Com os projetos sendo trabalhados paralelamente, chega a hora de juntar trabalhos feitos no projeto, para isso usamos o “Merge”.

1 - Vamos criar uma nova branch chamada branch2: **git checkout -b branch2**

2 - Agora vamos fazer uma alteração no arquivo primeiro.txt

3 - Quando adicionarmos e comitarmos essa alteração nossa branch2 terá conteúdo diferente da master: **git commit -am “Atualizacao da branch2”** (é um atalho para adicionar e comitar ao mesmo tempo).

4 – Vamos voltar para master: **git checkout master** e agora altere o arquivo segundo.txt e comite as alterações: **git commit -am “Atualizacao da master”**.



MERGE – Juntando branches

- 5 – Ainda dentro da master digite o comando: **git merge branch2**

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git merge branch2
Merge made by the 'recursive' strategy.
primeiro.txt | 3 ++-
1 file changed, 2 insertions(+), 1 deletion(-)
```

- 6 – Agora use o comando: **git log --graph** (vai mostrar as ramificações)

```
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Git (master)
$ git log --graph
*   commit 414a9caed9069c964b03877bf015fcd0bbe2561 (HEAD -> master)
    \
     | Merge: a65475d 3d3e721
     | Author: lcsilva76 <lcsilva@fiap.com.br>
     | Date:   Sun Feb 28 20:41:24 2021 -0300
     |
     |     Merge branch 'branch2'
     |
    *   commit 3d3e7218bab972904e0caa8393a7c22890f21e1d (branch2)
        \
         | Author: lcsilva76 <lcsilva@fiap.com.br>
         | Date:   Sun Feb 28 20:39:30 2021 -0300
         |
         |     Atualizacao da branch2
        *   commit a65475d01c90beee3ebf4498c21c139aff8852a0
            \
             | Author: lcsilva76 <lcsilva@fiap.com.br>
             | Date:   Sun Feb 28 20:40:50 2021 -0300
```



Praticando!!!

- Agora que conhecemos um pouco de Git e Github vamos praticar fazendo os seguintes exercícios:
 - 1 – Crie uma pasta no seu Desktop chamada **Exercicio-Git** e inicie ela no git;
 - 2 – Crie um arquivo chamado `exercicio1.txt`, escreva uma frase nele e salve.
 - 3 – Adicione o arquivo e faça o commit dele.
 - 4 – Crie uma branch chamada **branchExercicio**, crie um novo arquivo chamado `exercicio2.txt`, escreva uma frase, salve, adicione ele e faça o commit.
 - 5 – Volte para branch **master**, crie um arquivo chamado `exercicio3.txt`, escreva uma frase, salve, adicione ele e faça o commit.
 - 6 – Faça um merge da branch **branchExercicio** e visualize a junção com o **git log --graph**.
 - 7 – Crie um repositório remoto no github e faça um push da branch master do projeto.

GESTÃO DE CONTEÚDO E VERSIONAMENTO

PT-2



Prof. Alexandre Carlos

profalexandre.jesus@fiap.com.br

Prof. Luís Carlos

lsilva@fiap.com.br

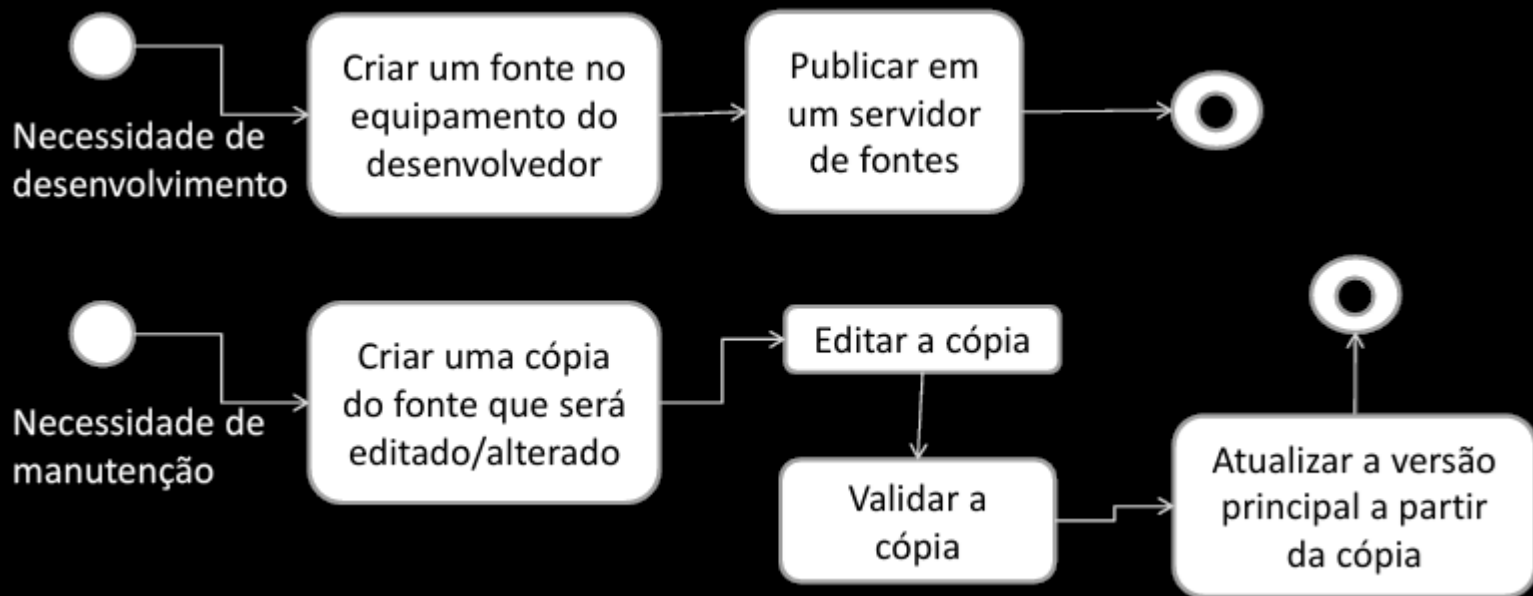
Prof. Wellington Cidade

profwellington.tenqrio@fiap.com.br



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

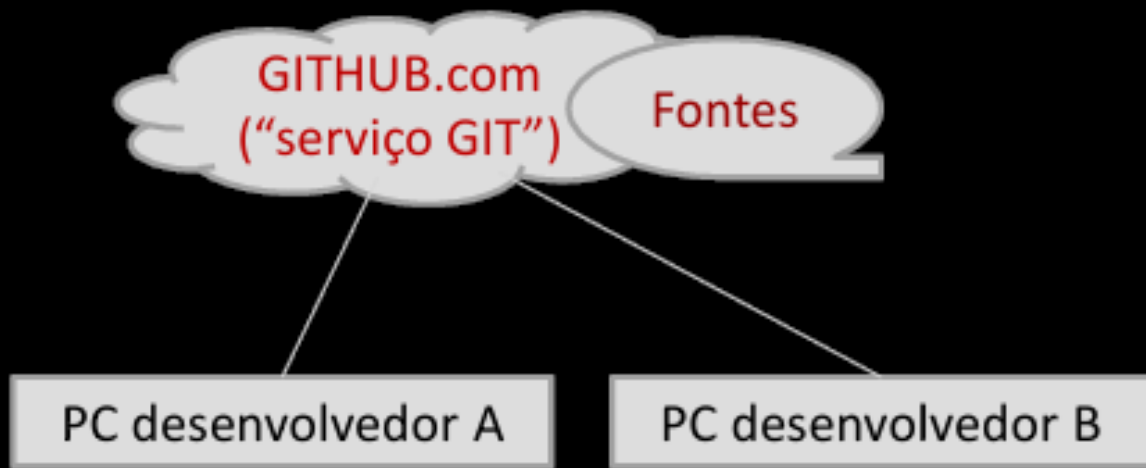
Vamos trabalhar com a seguinte proposta de processo de gestão de mudança no desenvolvimento de software:





ARQUITETURAS GIT

1ª - GITHUB como repositório único dos códigos fontes

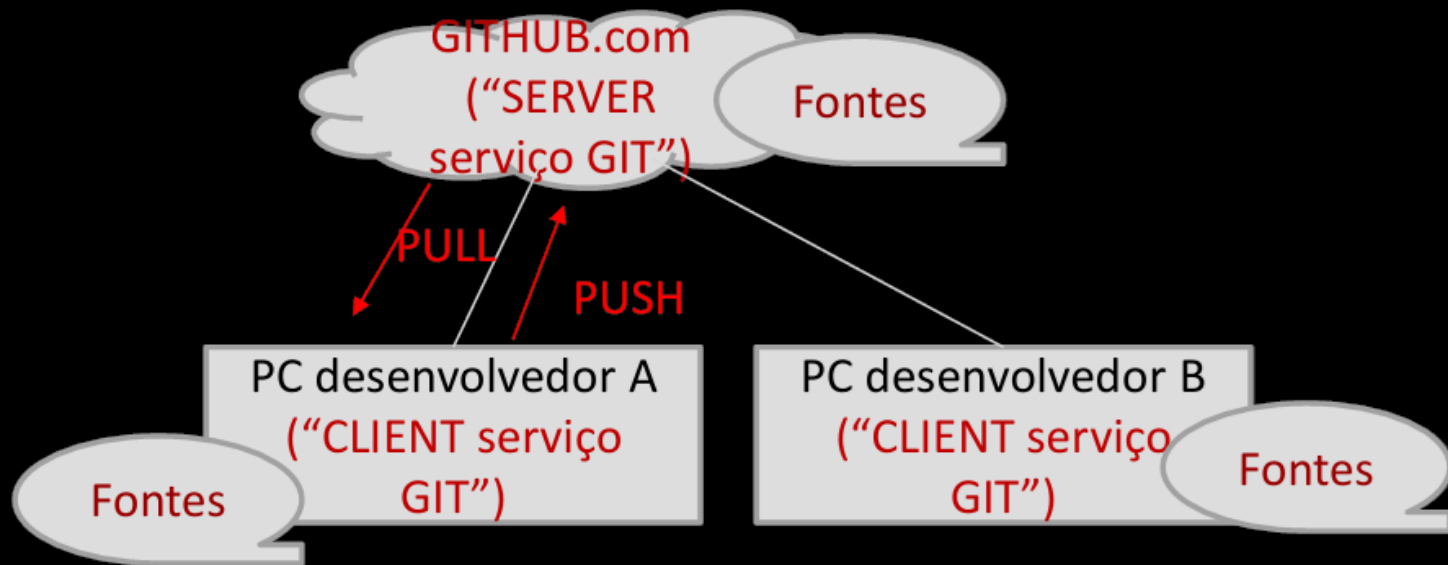


Aqui estamos usando o Github diretamente...



ARQUITETURAS GIT

2ª - GITHUB como repositório central dos fontes com clones do repositório de fontes nos PCs dos desenvolvedores.

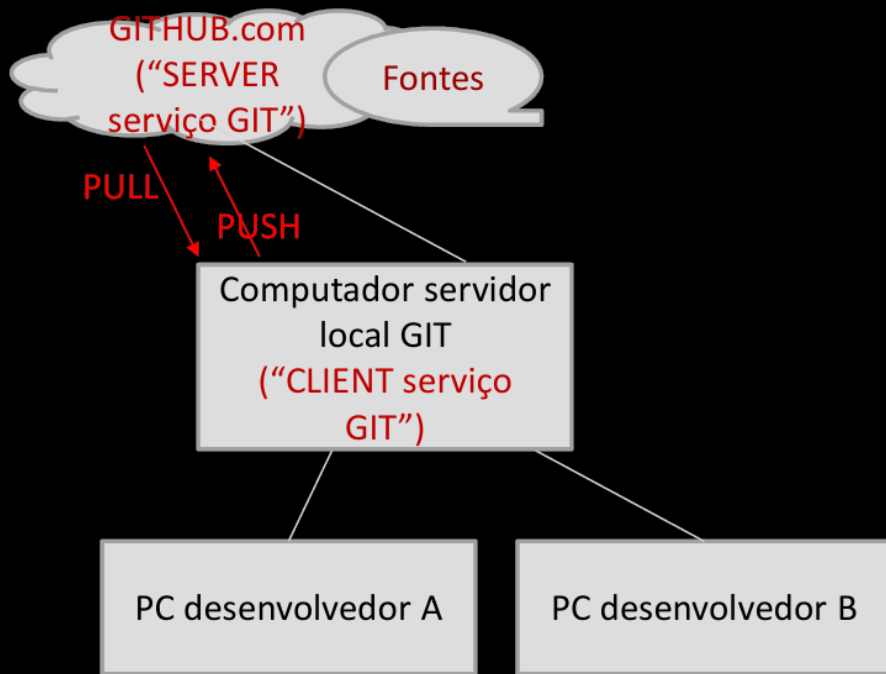


Aqui estamos usando o Github indiretamente...



ARQUITETURAS GIT

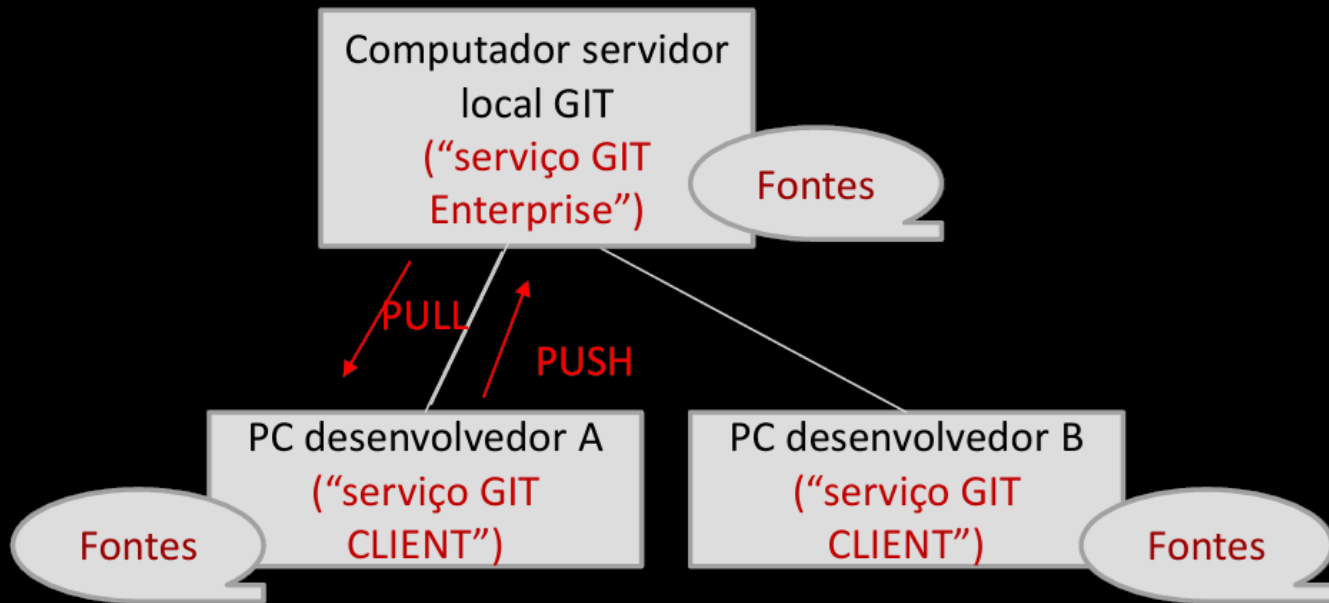
3ª - GITHUB como repositório central dos fontes com clone do repositório de fontes em um servidor corporativo





ARQUITETURAS GIT

4ª - Um servidor GIT Enterprise (proprietário da empresa) como o repositório de fontes e as estações de trabalho têm ou não cópias clonadas para trabalhar.





GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Através da ferramenta de controle de versões de códigos e/ou documentos de software, é possível evitar os problemas de se trabalhar com a fonte errada em um determinado ponto do projeto, facilitando a colaboração no projeto.



Desenvolvedor

Check out para fazer a manutenção

Check in após a manutenção

Repositório de Fontes

```
socket.error, (errno, strerror)
print "ncfiles: Socket error (%s) for host %s (%s)" % (errno, host, ip)

for h3 in page.findAll("h3"):
    value = (h3.contents[0])
    if value != "Afdeling":
        print >> txt, value
        import codecs
        f = codecs.open("alle.txt", "r", encoding="utf-8")
        text = f.read()
        f.close()
        # open the file again for writing
        f = codecs.open("alle.txt", "w", encoding="utf-8")
        f.write(value+"\n")
```



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Trabalhando com GIT, os arquivos fontes serão organizados em:

Cópia Master

Contém os arquivos na versão estável, que podem ser usados por outros desenvolvedores na integração de componentes ou com outros sistemas, ou podem ser usados para gerar um pacote de versão final do produto.

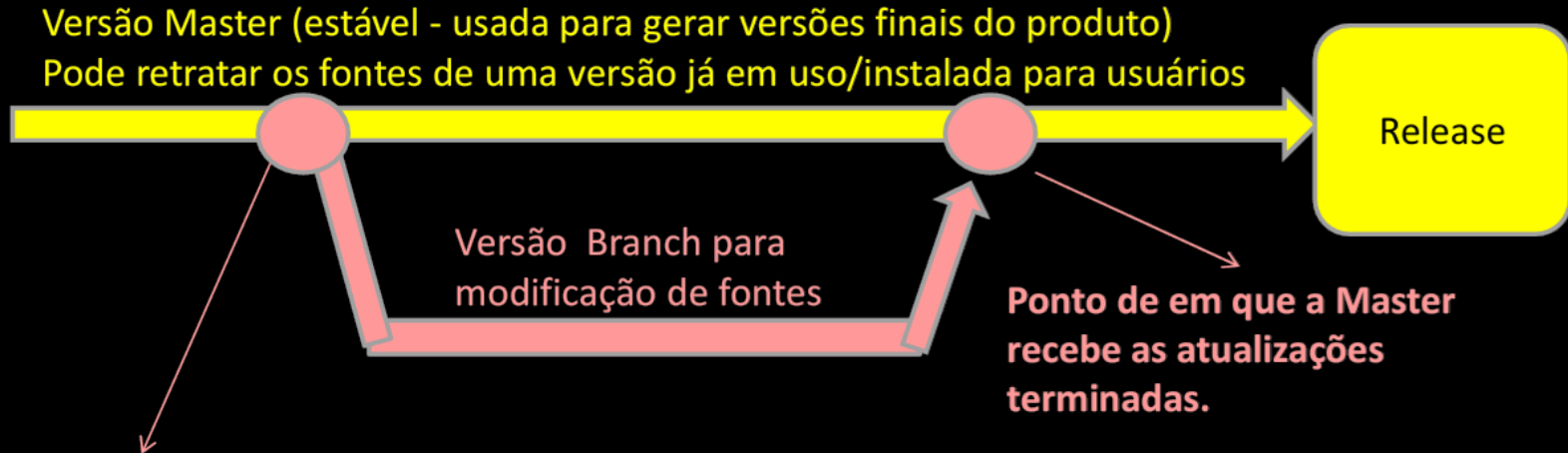
Cópia Branch

Contém os arquivos em uma versão de manutenção/atualização que não estão estáveis e não podem ser usados para gerar uma versão final do produto.



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

O funcionamento da Branch.



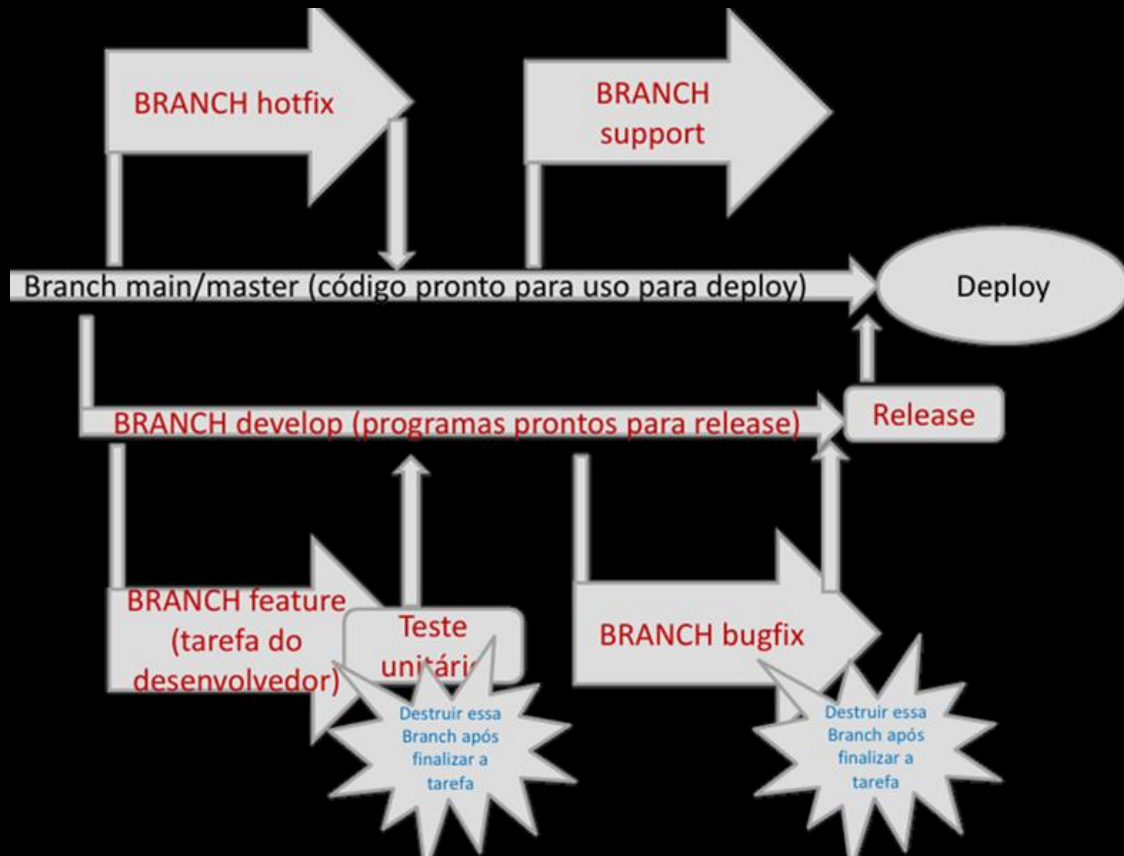
Ponto de necessidade de manutenção para:

- Corrigir o software/eliminar BUGs (manutenção CORRETIVA);
- Adaptar o software para novas regras de negócio (manutenção ADAPTATIVA);;
- Prevenir contra possíveis problemas futuros (manutenção PREVENTIVA);;
- Alcançar a perfeição na experiência do usuário (manutenção PREFECTIVA)



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

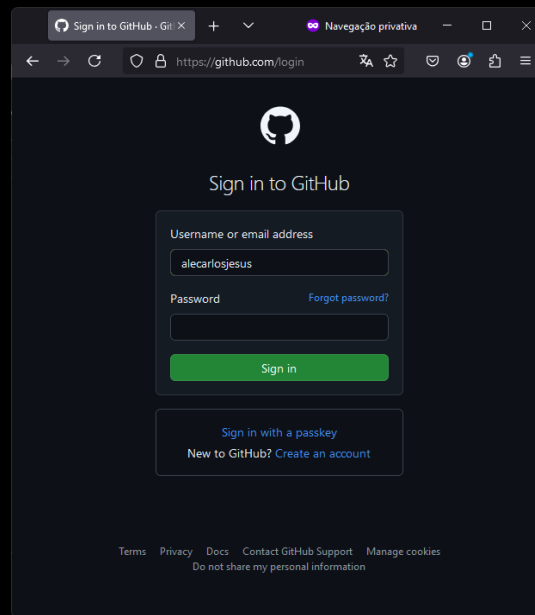
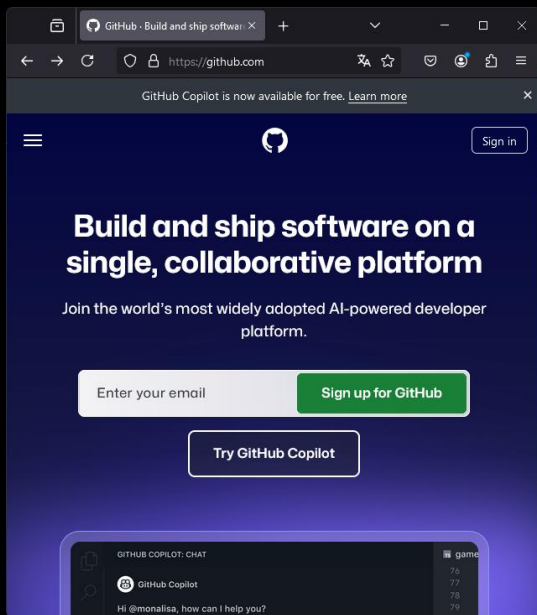
GIT FLOW





GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

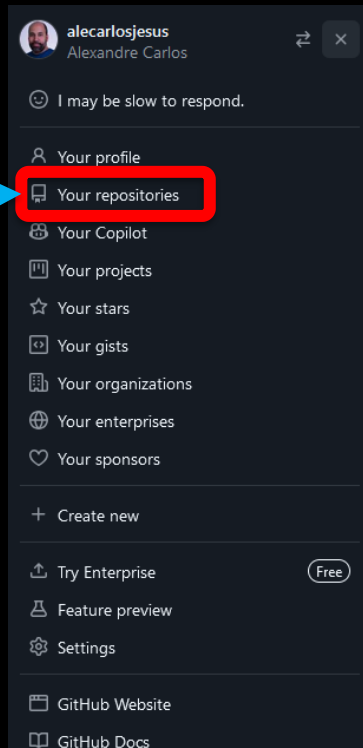
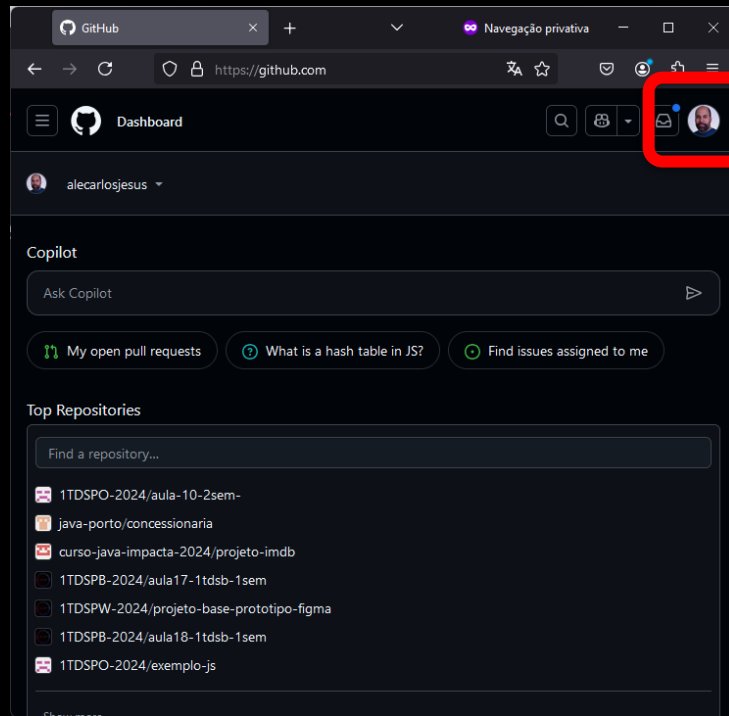
Faça login ou crie sua conta no site do Github: <https://github.com>





GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

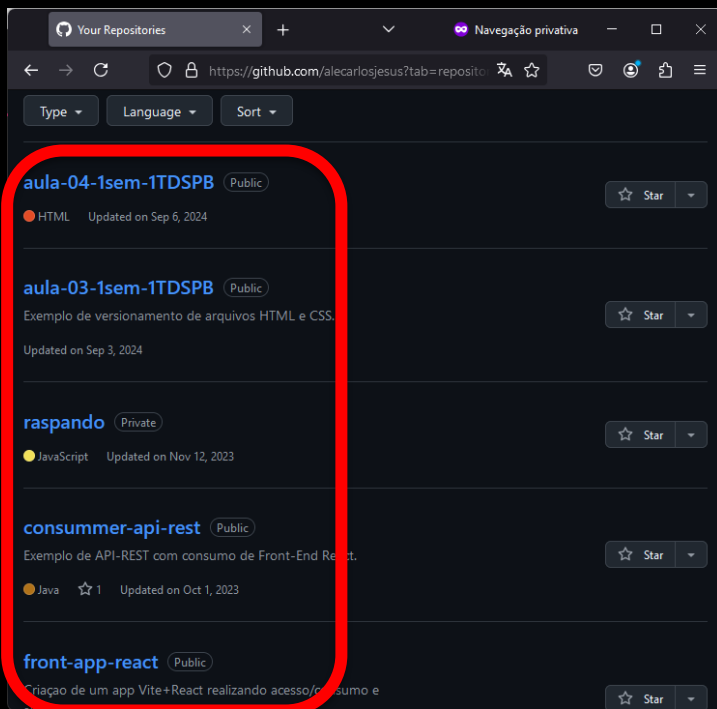
Logado no GITHUB, acesse a área de repositórios.





GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

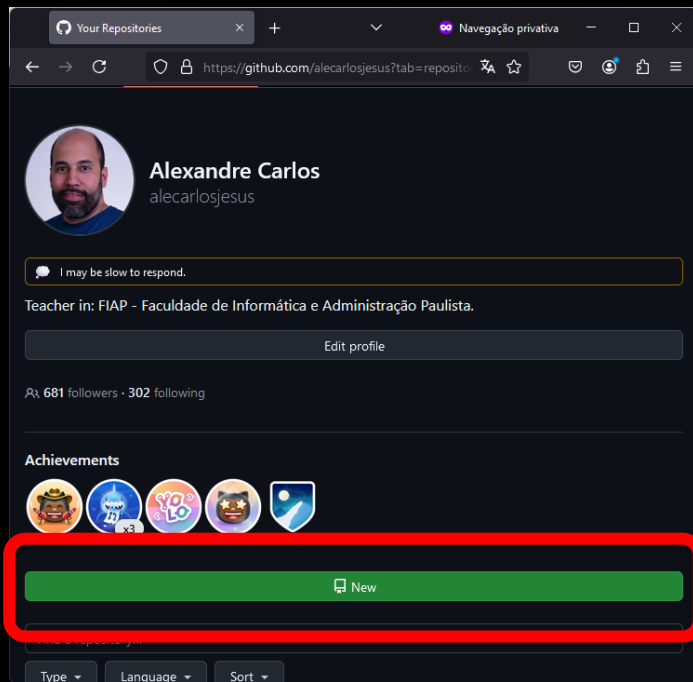
Seus repositórios de projetos de software aparecerão, caso você já tenha criado algum.





GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Para entendermos melhor o processo vamos criar um novo repositório:





GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Para entendermos melhor o processo vamos criar um novo repositório:

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

Required fields are marked with an asterisk (*).

Owner *
alecartosjesus

Repository name *
TesteGitHubHub
TesteGitHubHub is available.

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

Description (optional)
Respositório de demonstração...

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:
☐ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Annotations:
- nome do repositório (points to Repository name)
- Derscrição - opcional (points to Description)
- Visibilidade do repositório (points to Public/Private options)
- Se vai ter um arquivo Readme inicial. (points to Add a README file)

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

Initialize this repository with:
☐ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: None
Choose which files not to track from a list of templates. [Learn more about .gitignore files.](#)

Choose a license
License: None
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

① You are creating a public repository in your personal account.

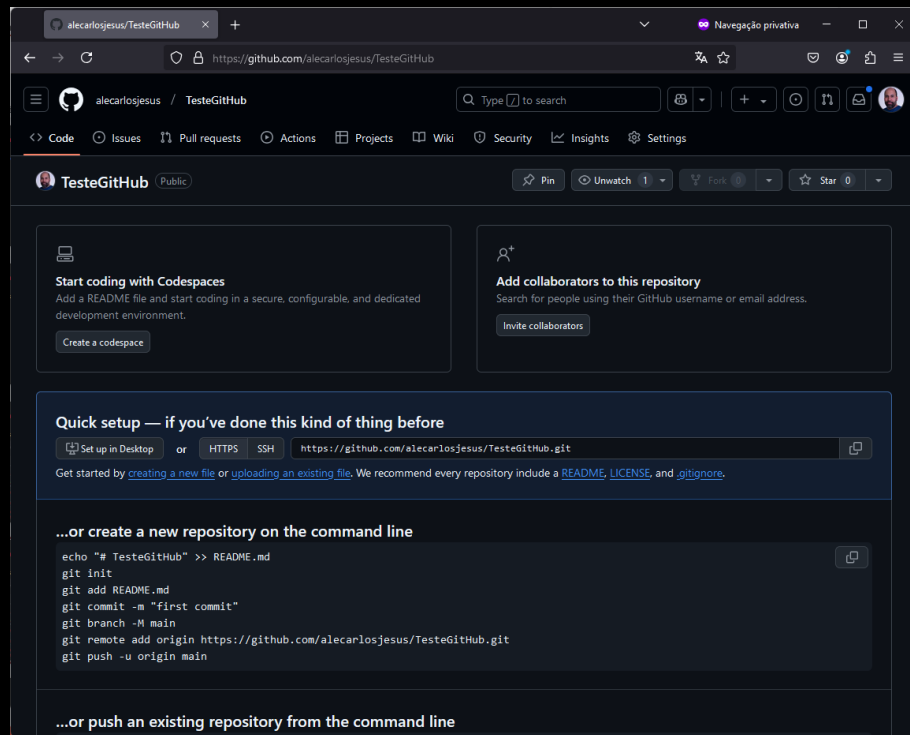
Create repository

Annotations:
- Adiciona um arquivo gitignore ao repositório (points to Add .gitignore)
- Adiciona um arquivo de licença. (points to Choose a license)
- Cria o repositório (points to Create repository button)



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Veja que ao criarmos um repositório vazio o Github nos dá a opção de criarmos um repositório local ou já utilizarmos um existente para conectarmos com o nosso repositório remoto. Por isso que se criarmos um repositório no Github devemos ter em mente o que vai ser feito deste.





GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Vamos criar um repositório inicializando ele com um arquivo README.MD, para podermos trabalhar com ele no github.

New repository

https://github.com/new

Create a new repository

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

Required fields are marked with an asterisk (*).

Owner * Repository name *

alecarlosjesus / TesteGitHub

TesteGitHub is available.

Great repository names are short and memorable. Need inspiration? How about [automatic-octo-doodle](#)?

Description (optional)

Repositório de exemplo

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:

☒ Add a README file

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

Add .gitignore

.gitignore template: None

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

Choose a license

License: None

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



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

A tela do repositório no Github:

alecarlosjesus/TesteGitHub: Rej x +

https://github.com/alecarlosjesus/TesteGitHub

alecarlosjesus / TesteGitHub

Type to search

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

TesteGitHub Public

Pin Unwatch 1 Fork 0 Star 0

main 1 Branch 0 Tags

Go to file

+ <> Code

+ Create new file

Upload files

alecarlosjesus Initial commit 6465016 · 2 minutes ago

README.md Initial commit

README 2 minutes ago

TesteGitHub

Repositório de exemplo

Carrega arquivos manualmente no repo.

Seleciona a área/cópia de fontes para trabalhar

Nomes dos arquivos que constam no repositório

Releases

No releases published

Published your package

Packages

No packages published

Published your first package

© 2025 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

A tela do repositório no Github:

alecarlosjesus/TesteGitHub: Rej x +

https://github.com/alecarlosjesus/TesteGitHub

alecarlosjesus / TesteGitHub

Type to search

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

TesteGitHub Public

Pin Unwatch 1 Fork 0 Star 0

main 1 Branch 0 Tags

Go to file

+ <> Code

+ Create new file

Upload files

alecarlosjesus Initial commit 6465016 · 2 minutes ago

README.md Initial commit 2 minutes ago

README

TesteGitHub

Repositório de exemplo

Carrega arquivos manualmente no repo.

Seleciona a área/cópia de fontes para trabalhar

Nomes dos arquivos que constam no repositório

Releases

No releases published

Package your first release

Packages

No packages published

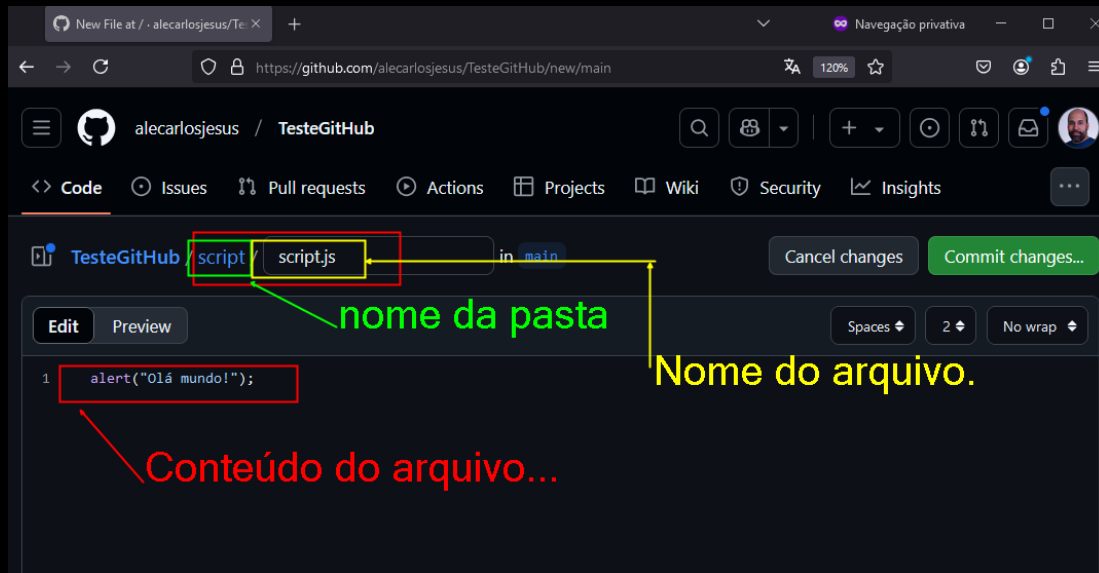
Publish your first package

© 2025 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Você pode criar pastas para separar tipos de arquivos no repositório. Vamos criar algumas pastas:



Obs: Estamos realizando este processo manualmente através do site do Github e normalmente esse processo se dá diretamente dentro dos projetos, localmente em nossas máquinas.



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Quando criamos, alteramos um determinado conteúdo ou um conjunto de alterações, por exemplo, editar muitos arquivos, pastas, apagar arquivos, criar novos em nosso repositório, devemos registrar estas alterações, a esse processo damos o nome de commit. E neste caso agora não será diferente, vamos realizar um commit por termos criado um arquivo e uma pasta.

Ao clicar em commit changes Esta janela aparece!!

Commit message
Create script.js
Aqui devemos colocar a msg referente a(s) alterações realizadas.

Extended description
Add an optional extended description..
Aqui são para comentários suplementares.

Commit Email
O local/branch onde os dados serão salvos.

☒ Commit directly to the main branch
☐ Create a new branch for this commit and start a pull request [Learn more about pull requests](#)

Cancel Commit changes Botão para confirmar.



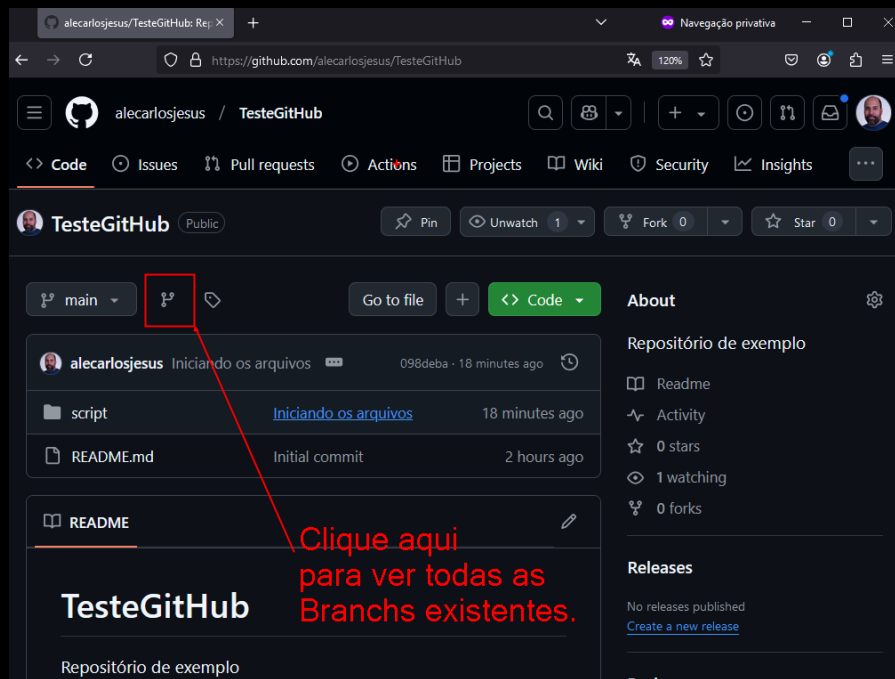
GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Para incluir novos fontes ou alterar fontes publicados no GIT, crie uma Branch (cópia de desvio) da área Main.

A Main deve conter apenas os códigos fonte estáveis, que podem ser usados por outros desenvolvedores.

A Branch que será criada é uma réplica da Main para que um ou mais programadores façam alterações nos programas e depois republiquem os arquivos atualizados na Main.

Crie uma Branch a partir da Main com o nome que desejar.





GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Crie uma Branch a partir da Main com o nome que desejar.

Clique aqui para abrir a janela abaixo:

New branch

Overview Yours Active Stale All

Search branches...

Create a branch

New branch name

nova-branch

Source

main

Cancel Create new branch

Botão para gerar...

Aqui estamos dizendo que o conteúdo desta nova Branch está sendo copiado a partir da main.

Branches

New branch

Overview Yours Active Stale All

Search branches...

Default

Branch	Updated	Check status	Behind	Ahead	Pull request
main	33 minutes ago			Default	

Your branches

Branch	Updated	Check status	Behind	Ahead	Pull request
nova-branch	now		0	0	

Active branches

Branch	Updated	Check status	Behind	Ahead	Pull request
nova-branch	now		0	0	



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Carregamento de Arquivo Manual:

Suba um arquivo .JS ou .HTML para experimentar, vamos utilizar a ferramenta de upload do Github.

Como alternativa, você pode abrir a pasta com o seu arquivo no Windows Explorer e arrastá-lo para a página do GITHUB.



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Carregamento de Arquivo Manual:

The screenshot shows the GitHub 'Upload files' page for the repository 'alecarlosjesus/TesteGitHub'. The page has a dark theme. At the top, there's a navigation bar with the repository name and various icons. Below that, a tabbed interface shows 'Code' as the active tab. The main content area is titled 'TesteGitHub /' and contains a large box with the text 'Arrastando arquivos para cá, você pode fazer download para seu repositório.' and 'Drag files here to add them to your repository'. Below this, there's a button labeled 'Or choose your files' which is highlighted with a red box. A red arrow points from the text 'Assim como clicando aqui!!' to this button. Below the main content area, there's a 'Commit changes' section with a green border, containing two input fields: 'Add files via upload' and 'Add an optional extended description...'. At the bottom, there's a green text overlay that reads 'Não podemos esquecer do nosso COMMIT ao final desta operação!!'.

Upload files · alecarlosjesus/TesteGitHub

alecarlosjesus / TesteGitHub

Code Issues Pull requests Actions Projects Wiki Security Insights

TesteGitHub /

Arrastando arquivos para cá, você pode fazer download para seu repositório.

Drag files here to add them to your repository

Assim como clicando aqui!!

Or choose your files

Commit changes

Add files via upload

Add an optional extended description...

Não podemos esquecer do nosso COMMIT ao final desta operação!!



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Carregamento de Arquivo Manual:

Veja que após realizarmos o carregamento do nosso arquivo em nossa Branch paralela é mostrado um botão de pull request(merge).

Isso quer dizer que se atuarmos sobre esse botão vamos atualizar(merge) a Branch Main com o conteúdo a partir de nossa Branch(nova-branch).

The screenshot shows the GitHub interface for the repository 'alecarlosjesus / TesteGitHub'. The 'nova-branch' is selected, and a green button labeled 'Compare & pull request' is highlighted with a red box. A red arrow points from the text 'Criar o merge!!' to this button. The interface also shows the repository's commit history and a list of files.

File	Commit Message	Time Ago
script	Iniciando os arquivos	5 hours ago
README.md	Initial commit	6 hours ago
index.html	Adicionei index.html.	8 minutes ago



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Carregamento de Arquivo Manual:

Depois que estiver certo que a mudança está completa e correta, publique a modificação na cópia Main, tornando-a disponível para todos os desenvolvedores usarem!

Comparing main...nova-branch

alecarlosjesus / TesteGitHub

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).

base: main ← compare: nova-branch ✓ Able to merge. These branches can be automatically merged.

Add a title

Adicionei index.html. ← Msg do commit

Add a description

Write Preview

Estou certo de realizar o pull request e atualizar a main. ← Msg descritiva do commit. Não obrigatória.

Create pull request



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Carregamento de Arquivo Manual:

The screenshot shows a GitHub pull request page for 'Adicionei index.html. #1'. The pull request is open, and the 'Merge pull request' button is highlighted with a red box. A red arrow points to the button with the text 'Confirmação do merge!'. Another red box highlights the message 'This branch has no conflicts with the base branch' with the text 'A confirmação da não existência de conflitos...'. The page also shows a comment from 'alecarlosjesus' and a 'Verified' commit.

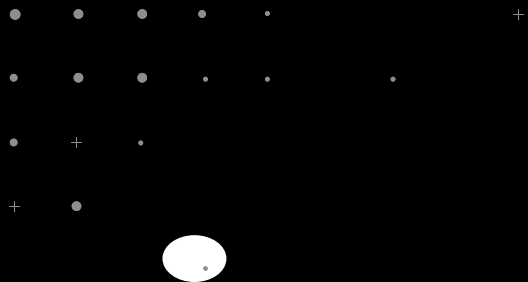
The screenshot shows the same GitHub pull request page, but now the pull request is merged. The status is 'Merged', and the 'Merge pull request' button is replaced by a 'Delete branch' button. A red box highlights the message 'Pull request successfully merged and closed' with the text 'Merge realizado com SUCESSO!!'. The page also shows a comment from 'alecarlosjesus' and a 'Verified' commit.



GERENCIAMENTO DA PRODUÇÃO E ATUALIZAÇÃO DE SOFTWARE

Exercitando...

- Experimente criar uma nova Branch para fazer alterações!!
- Faça as modificações e execute o Commit.
- Crie uma Pull request e execute a atualização/Pull-Request/Merge da Main.



|
+

FIAP

