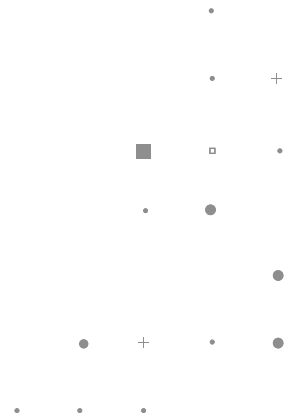




FIAP

GRADUAÇÃO



TDS

Responsive Web Development

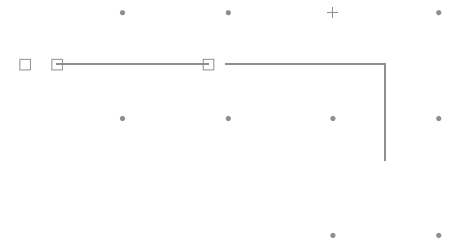
Prof. Alexandre Carlos - profalexandre.jesus@fiap.com.br

Prof. Luís Carlos - lsilva@fiap.com.br

Prof. Alexander Gobbato - profalexander.albuquerque@fiap.com.br

Apresentação Professor

- Tecnólogo em Telecomunicações (Unicid– 2001)
- Especialização em Engenharia de Web Sites (Cruzeiro do Sul - 2004)
- □ Mestre em Ensino de Ciências e Matemática (Cruzeiro do Sul - 2013)
- Docente desde 2004 nos cursos de Ciência da Computação, Análise e Desenvolvimento de Sistemas e Jogos Digitais



Apresentação Professor

- Profissional com mais 20 anos trabalhando na área de tecnologia, com vasta experiência em análise, execução e suporte a sistemas web e cliente-servidor.

- Atuei em projetos webs, exercendo funções de Análise, desenvolvimento e implantação para Help Desk, Integração Financeira, Estoque, Venda de ingressos e Relatório de Despesas.

- Como desenvolvedor já atuei com as linguagens Visual Basic 6.0, Asp e Asp.net com C# criando serviços de integrações com Oracle SOA e banco de dados Sql Server e Oracle.

- Atuo com análise de requisitos, priorização das tarefas de desenvolvimento, análise funcional, refinamento das especificações técnicas, utilizando a ferramenta Postman e SoapUi para criação de testes e validação de serviço.

Leitura e compreensão de log utilizando o MobaXterm.

Atualmente exerço a função de analista de integração pela Builders, utilizando a ferramenta Sensedia e como Líder Técnico a frente do grupo Fleury responsável por projetos de Ficha Eletrônica e Padrões TISS.

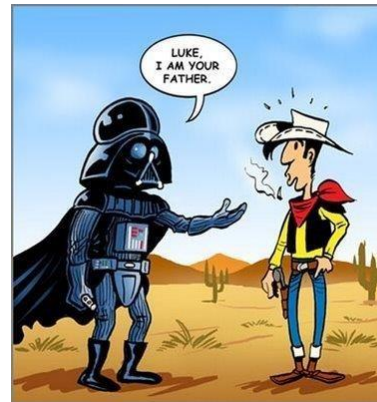
Quem são vocês?

Conhecendo a turma

Trabalha ou dedicação exclusiva aos estudos?

Já conhece alguma linguagem de programação?

Qual?





O que irei aprender?

Como irei aprender?

Objetivo da disciplina

Estudos sobre conceitos e aplicações de Interfaces Gráficas e utilização de mecanismos de acesso de banco de dados

Descrição Aula
Aula apresentação
Introdução ao Git
Introdução GitHub
Introdução Javascript + exercícios + github
Javascript + eventos + exercícios + github
Projeto 1
Padrões Web
Introdução Html5
Introdução Figma
Introdução Html Semântico
Projeto 2
Introdução CSS3 + exercícios
Introdução CSS3 + exercícios
Projeto 3
Hmtl5 - Tabelas e Formulários + exercícios
Projeto 4
Html5 + CSS3 + Responsividade + exercícios
Html5 + CSS3 + Flexbox + exercícios
Projeto 5

-
- A blank coordinate grid with x and y axes. The x-axis is horizontal and the y-axis is vertical. They intersect at the origin, which is marked with a small square. There are tick marks on both axes, but no numerical labels.

Git e Github

- O que é controle de versão
- Git – principais comandos
- Github – repositório remoto



O que é controle de versão?

O controle de versão é um sistema que é utilizado para gerenciar alterações em programas de computador, documentos, grandes aplicações web ou outros projetos.

Git é considerado o arroz com feijão do desenvolvedor. Não importa qual sua especialidade, você vai precisar dele e é importante que saiba utilizar do jeito certo.

Pela documentação oficial, Git é um sistema de controle de versão distribuído de código aberto e gratuito, projetado para lidar com tudo, de projetos pequenos a grandes. O que isso significa? Significa que com o Git é possível manter um histórico das alterações dos seus arquivos, sabendo quem, por que e quando um arquivo foi editado.



O que é controle de versão?

Ele resolve problemas como:

- 1 – A necessidade de fazer cópias e mais cópias de um mesmo projeto** – Ter que fazer cópias do projeto a cada alteração importante, temendo ter a necessidade de voltar ao estado anterior.
- 2 – Acidentes durante o desenvolvimento** – Copiar, sobrepor ou até mesmo apagar um trecho de código, ou até mesmo um ou vários arquivos e não conseguir recuperar.



O que é controle de versão?

A principal funcionalidade do Git, que o faz ser amplamente utilizado em projetos de desenvolvimento de *software*, é a **possibilidade de fazer o controle de versões de modo colaborativo**, ou seja, é possível que o mesmo arquivo seja modificado ao mesmo tempo por dois desenvolvedores diferentes, e que ambas as alterações sejam salvas sem que nenhum código seja sobrescrito.

Para permitir o modo colaborativo, o Git utiliza o conceito de ramificação ou *branch*, onde cada *branch* é uma linha do tempo que possui marcos ou *commits*, e nesse *branch* os arquivos podem ser alterados livremente sem impactar outras ramificações.



O que é controle de versão?

No mercado temos basicamente dois tipos de versionadores:

1 – Que verifica e salva os arquivos que apresentam diferenças em versões e não possibilita versões paralelas.

Ex: CVS, Subversion, etc.

2 – Git ele faz snapshots (como se fosse uma foto que registra o estado naquele momento).

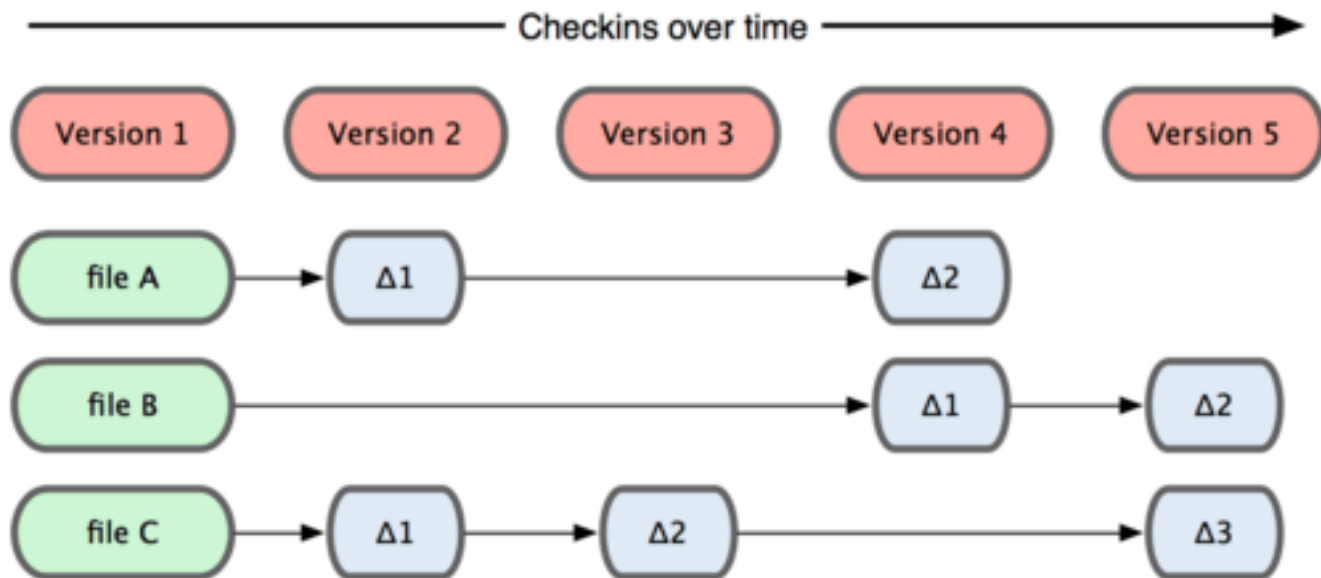
Ex: Git





O que é controle de versão?

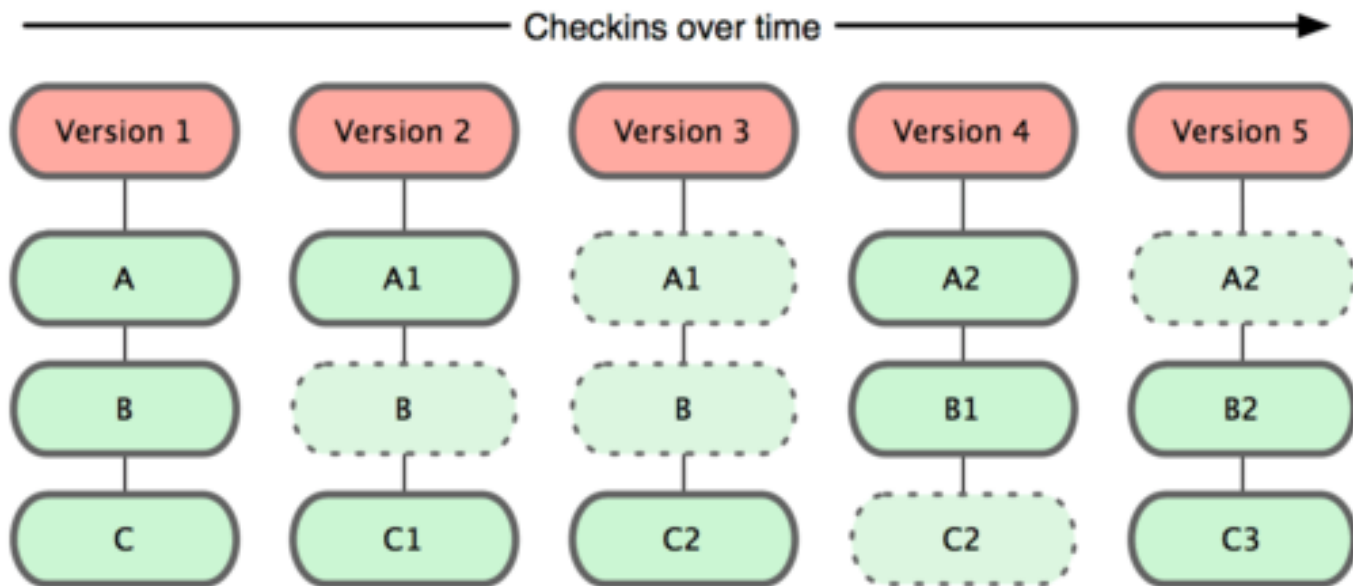
- Outros Sistemas:





O que é controle de versão?

- Sistema Git:





O que é controle de versão?

- Sistema Git:





Um pouco sobre o Git

O Git foi criado por Linus Torvalds, o mesmo criador do sistema operacional Linux, depois de uma quebra de contrato com a BitKeeper, empresa que fornecia o sistema de versionamento para eles na época.

Insatisfeito com valores e desempenho da ferramenta, ele quebrou o contrato e decidiu criar o seu próprio versionador, trazendo melhorias como:

- Velocidade e espaço de armazenamento;
- Design Simples, mais fácil de utilizar;
- Robustez, permite a criação de milhares versionamentos paralelos;
- Capaz de lidar com grandes projetos



Github e Gitlab



Github e Gitlab

Github e Gitlab são plataformas de hospedagem de código-fonte. Elas **permitem que os desenvolvedores contribuam em projetos privados ou abertos** (mais conhecidos como projetos *open source*).

Nessas plataformas, cada projeto contendo um código-fonte é considerado um repositório. Por exemplo, se você participa de projeto em que é desenvolvido o site de um e-commerce, e o código do frontend é desenvolvido separadamente do código [backend](#), cada um desses códigos-fontes serão hospedados como repositórios separados.

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.

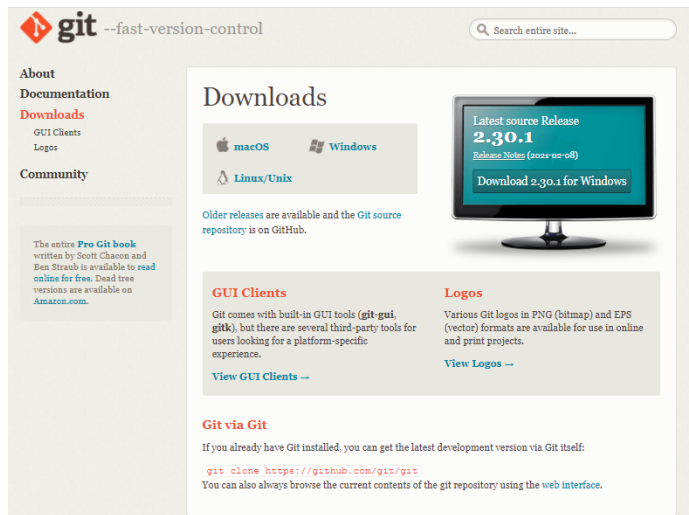
Sua função principal é armazenar projetos na web versionados pelo Git.



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>





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 "lsilva@fiap.com.br"
```



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`

`lsilva@fiap.com.br`

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.

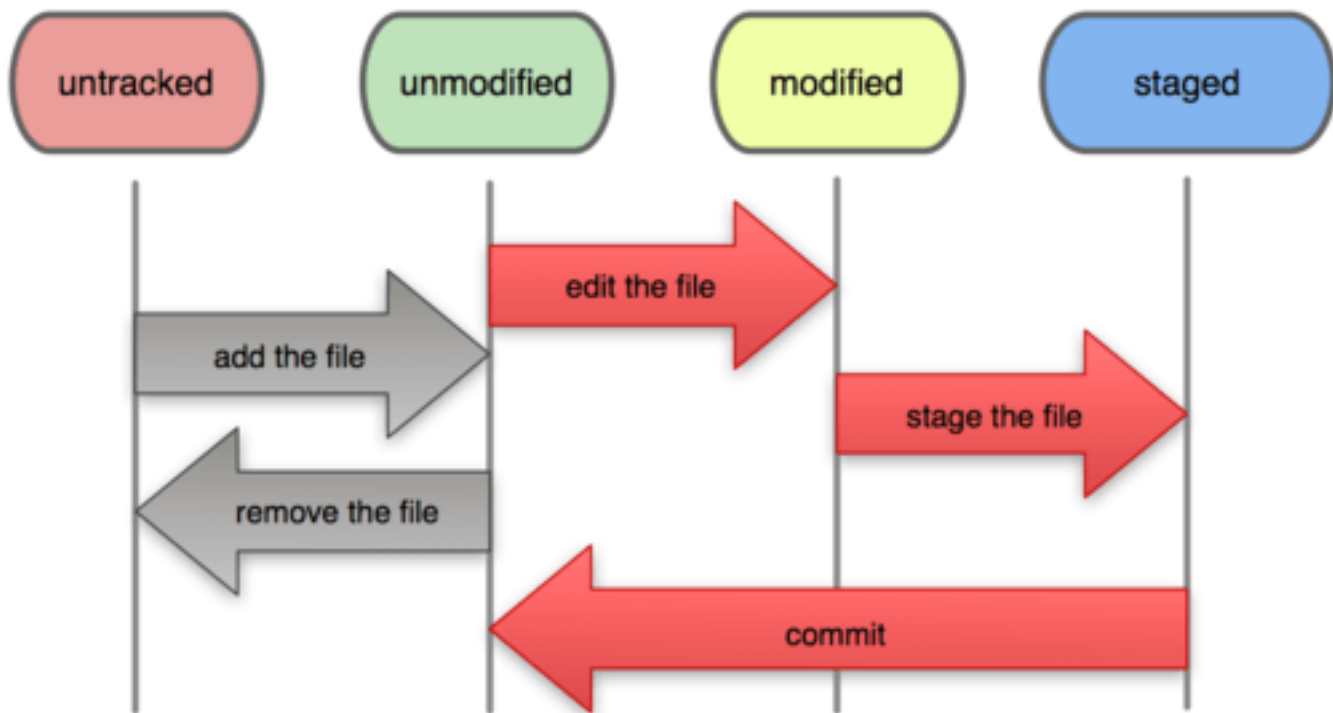
Agora no terminal digite ls, este comando lista do diretório.

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



Estados dos Arquivos

Para controlar o versionamento dos nossos arquivos o Git utiliza 4 Status diferentes em seu ciclo de vida:





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.



Commit

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
```



Commit

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 <lsilva@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
```



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
```

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



Commit

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
```

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
```



Commit

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 voltam para antes do staged;

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

Obs. A hash deve ser a do commit para que teremos retornar

Github – Repositório Remoto

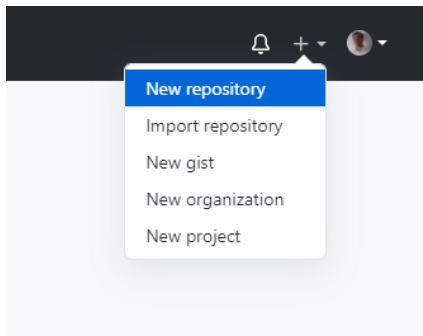




Criando o Primeiro Repositório Remoto

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

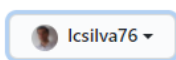
Agora temos que colocar as seguintes informações:

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 *

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!!!

The screenshot shows the GitHub interface for a repository named 'Exemplo-Git' by user 'Icsilva76'. At the top, the repository name is displayed with a red arrow pointing to it. Below this is a navigation bar with tabs for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The 'Code' tab is selected. Below the navigation bar, there are buttons for 'Go to file', 'Add file', and a green 'Code' button. The main content area shows a commit by 'Icsilva76' titled 'Segundo commit' with the commit hash '4ec40e7' and the message 'Segundo commit'. A file named 'primeiro.txt' is listed as part of this commit, with a red arrow pointing to it. At the bottom, there is a blue box with the text 'Help people interested in this repository understand your project by adding a README.' and a green 'Add a README' button.

Icsilva76 / Exemplo-Git

<> Code ! Issues 🔗 Pull requests 🔄 Actions 📁 Projects 📖 Wiki 🛡 Security 📊 Insights ⚙ Settings

🔗 master 1 branch 0 tags Go to file Add file Code

Icsilva76 Segundo commit 4ec40e7 3 hours ago 2 commits

📄 primeiro.txt Segundo commit 3 hours ago

Help people interested in this repository understand your project by adding a README. Add a README



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!

The screenshot shows a GitHub repository page for 'lcsilva76'. At the top, a commit message 'Add segundo.txt' is displayed with the commit hash 'f20a77a' and the time '6 minutes ago'. Below this, a table lists the commit history:

File	Commit Message	Time
primeiro.txt	Segundo commit	3 hours ago
segundo.txt	Add segundo.txt	6 minutes ago

A red arrow points to the 'segundo.txt' row. At the bottom of the page, there is a prompt: 'Help people interested in this repository understand your project by adding a README.' with a green button labeled 'Add a README'.

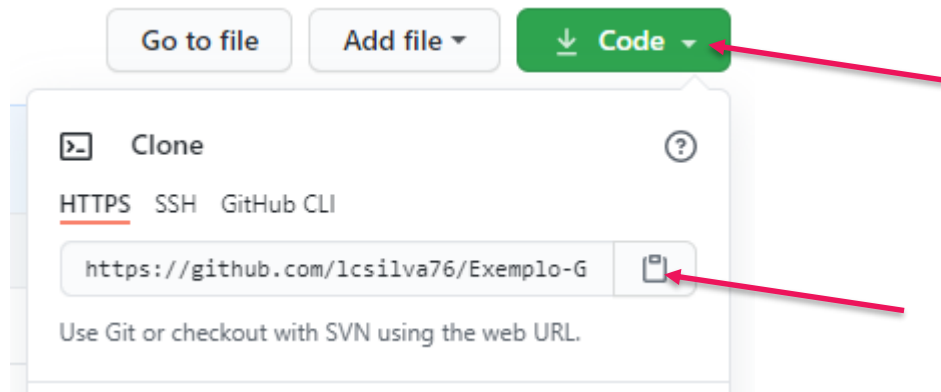


Clonando um Repositório Remoto

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

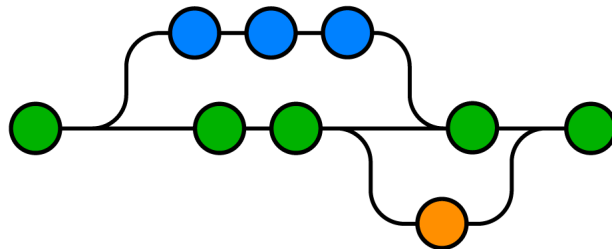
Luis@PC-LUIS MINGW64 ~/Desktop/Exemplo-Github-Clone (master)
$ ls
primeiro.txt  segundo.txt
```



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 -m**

```
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 <lsilva@fiap.com.br>
   | Date: Sun Feb 28 20:41:24 2021 -0300
   |
   | Merge branch 'branch2'
   |
   | * commit 3d3e7218bab972904e0caa8393a7c22890f21e1d (branch2)
   |   | Author: lcsilva76 <lsilva@fiap.com.br>
   |   | Date: Sun Feb 28 20:39:30 2021 -0300
   |   |
   |   | Atualizacao da branch2
   |   |
   | * commit a65475d01c90beee3ebf4498c21c139aff8852a0
   |   | Author: lcsilva76 <lsilva@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.



FIAP

