

# AULA 04 - CONTROLE DE VERSÃO (GIT)

Disciplina de Backend - Professor Ramon Venson - SATC 2024

# Git

O **Git** é um sistema de controle de versão, criado pelo finlandês Linus Torvalds (o mesmo criador do Linux). O sistema Git é utilizado principalmente nos projetos de desenvolvimento de software, sendo capaz de manter a organização das edições de código, além de permitir que diversas pessoas trabalhem simultaneamente em um mesmo projeto.



## O que é o controle de versão?

- Gerenciamento do código-fonte
- Contribuições de diferentes programadores
- Desfazer alterações problemáticas
- Resolução de conflitos de código

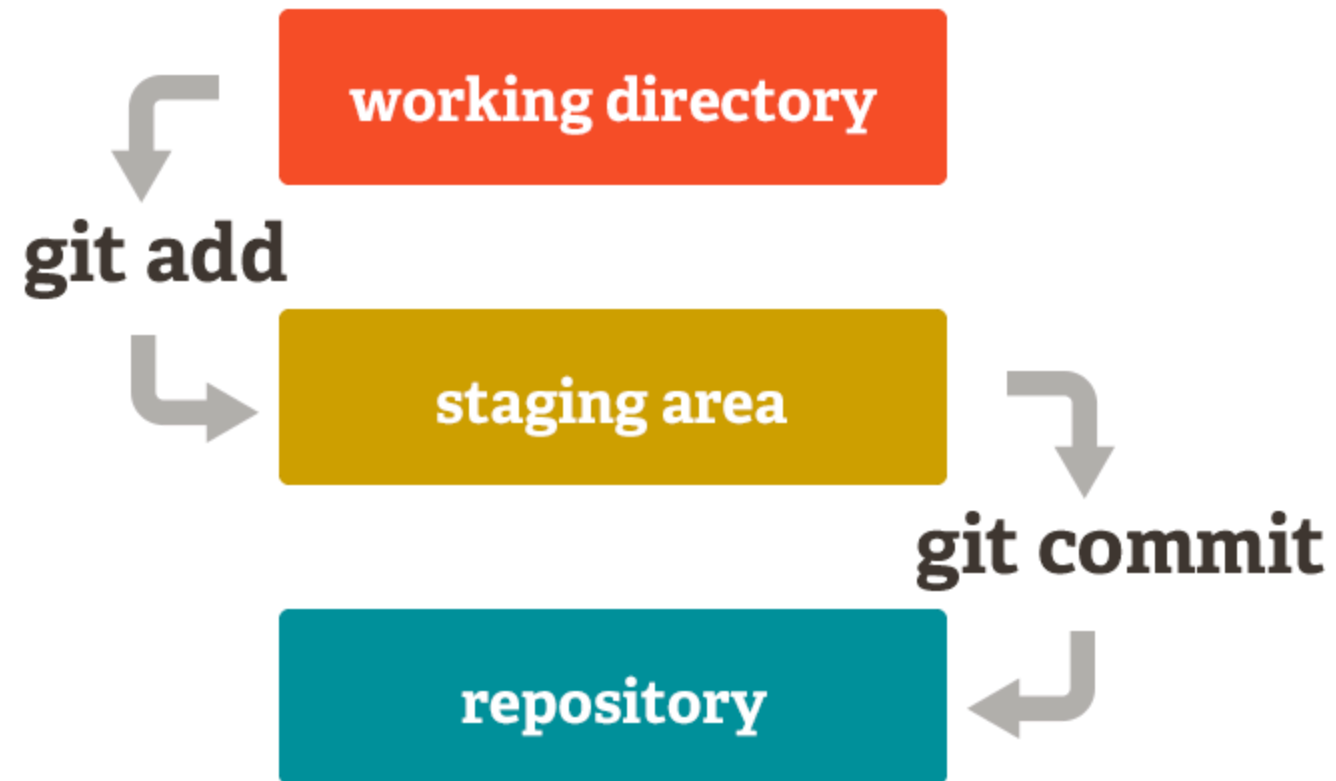
## Configuração do Git

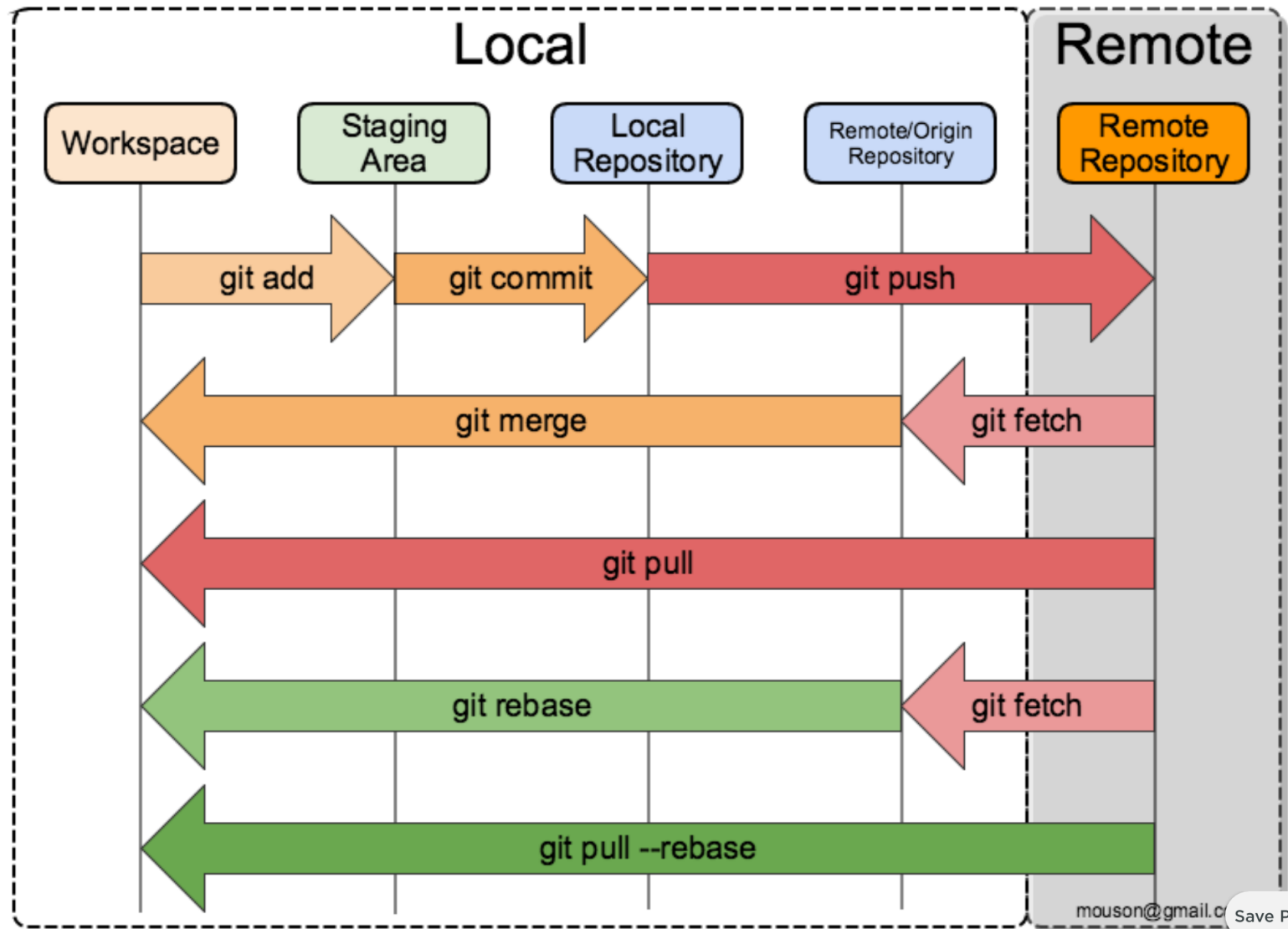
Após instalado, execute os seguintes comandos no terminal / git bash para configurar seu git com suas informações:

```
$ git config --global user.name "Alan Turing"  
$ git config --global user.email "alan@turing.com"
```

Isso não se trata de autenticação e sim de uma identificação do commiter da máquina. Utilize a configuração sem o --global dentro da pasta do repositório para configurações individuais.

# Áreas do Git

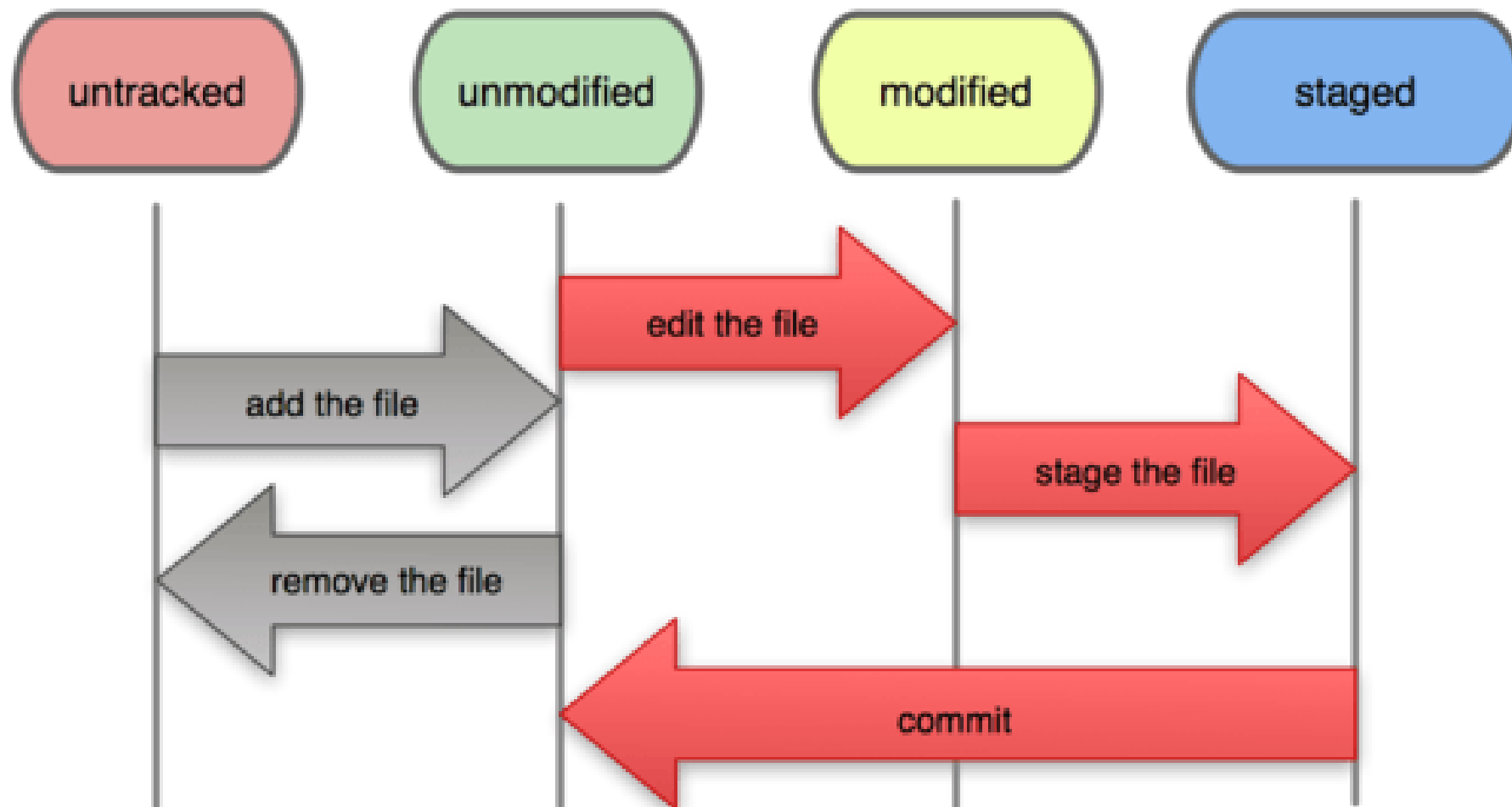




- **Stash:** Área alternativa de armazenamento de alterações.
- **Workspace** ou **Working Directory:** Área de trabalho que mantém o estado real/atual de todos os arquivos do projeto.
- **Stage Area (Index):** Área de trabalho intermediária para organização das modificações a serem armazenadas no repositório local através do commit.
- **Local Repository:** Repositório de todas as versões geradas pelo comando commit. Esta área representa o estado mais estável do desenvolvimento.
- **Remote Repository:** Repositório remoto que reúne as contribuições dos repositórios locais.

# Estágios de arquivo

## File Status Lifecycle

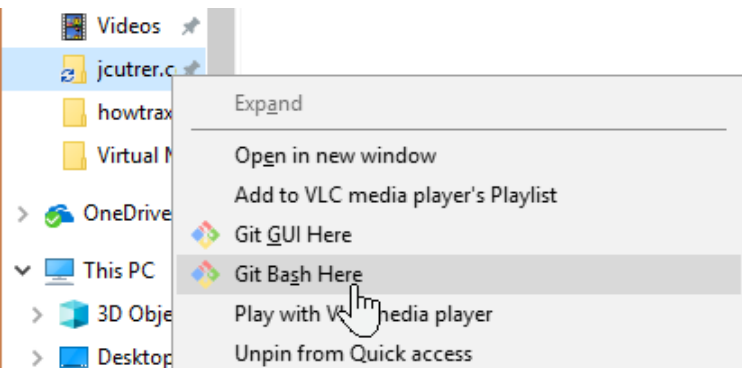


- **Untracked:** Arquivo que não está sendo versionado.
- **Unmodified:** Arquivo que não foi modificado quando comparado à ultima versão.
- **Modified:** Arquivo que foi modificado quando comparado à ultima versão.
- **Staged:** Arquivo alterado que foi adicionado ao INDEX e está pronto para o versionamento.

# Comandos

## Navegação Básica

Há duas maneiras distintas de iniciar o terminal de comandos do Git:



1. Através da interface gráfica, entre na pasta onde deseja criar/abrir o repositório do Git e selecione a opção ***Git Bash Here***.
2. Abra a aplicação **Git Bash** e navegue até a pasta desejada usando o comando `cd <caminho da pasta>` (Ex.: `cd Downloads/pascalzim6031/pascalzin` )

## Navegação no Bash

Para navegar no bash, utilize os seguintes comandos como referência:

| Comando                         | Descrição                           |
|---------------------------------|-------------------------------------|
| <code>cd &lt;caminho&gt;</code> | Acessa a pasta do caminho           |
| <code>ls</code>                 | Mostra arquivos no diretório atual  |
| <code>pwd</code>                | Mostra o caminho do diretório atual |

## Iniciando novo repositório

Para iniciar um novo repositório Git, basta executar os seguintes passos:

1. Certifique-se de que está na pasta correta no bash usando o comando *pwd* ou consultado a barra de título
2. Execute o comando: `git init`

NOTA: Repare que é possível iniciar um novo repositório em qualquer pasta que não possua um repositório ativo. Isso inclui pastas que já possuem conteúdo.

## Adicionando alterações

Após adicionar novos arquivos ou realizar alterações nos arquivos existentes, utilize o comando abaixo para adicionar estas alterações na *staging area* do repositório.

```
git add .
```

NOTA: O ponto no final refere-se ao diretório em questão. Este comando adiciona TODOS os arquivos alterados à *staging area*. Para adicionar pastas ou arquivos específicos, utilize o nome do arquivo/pasta no lugar do ponto.

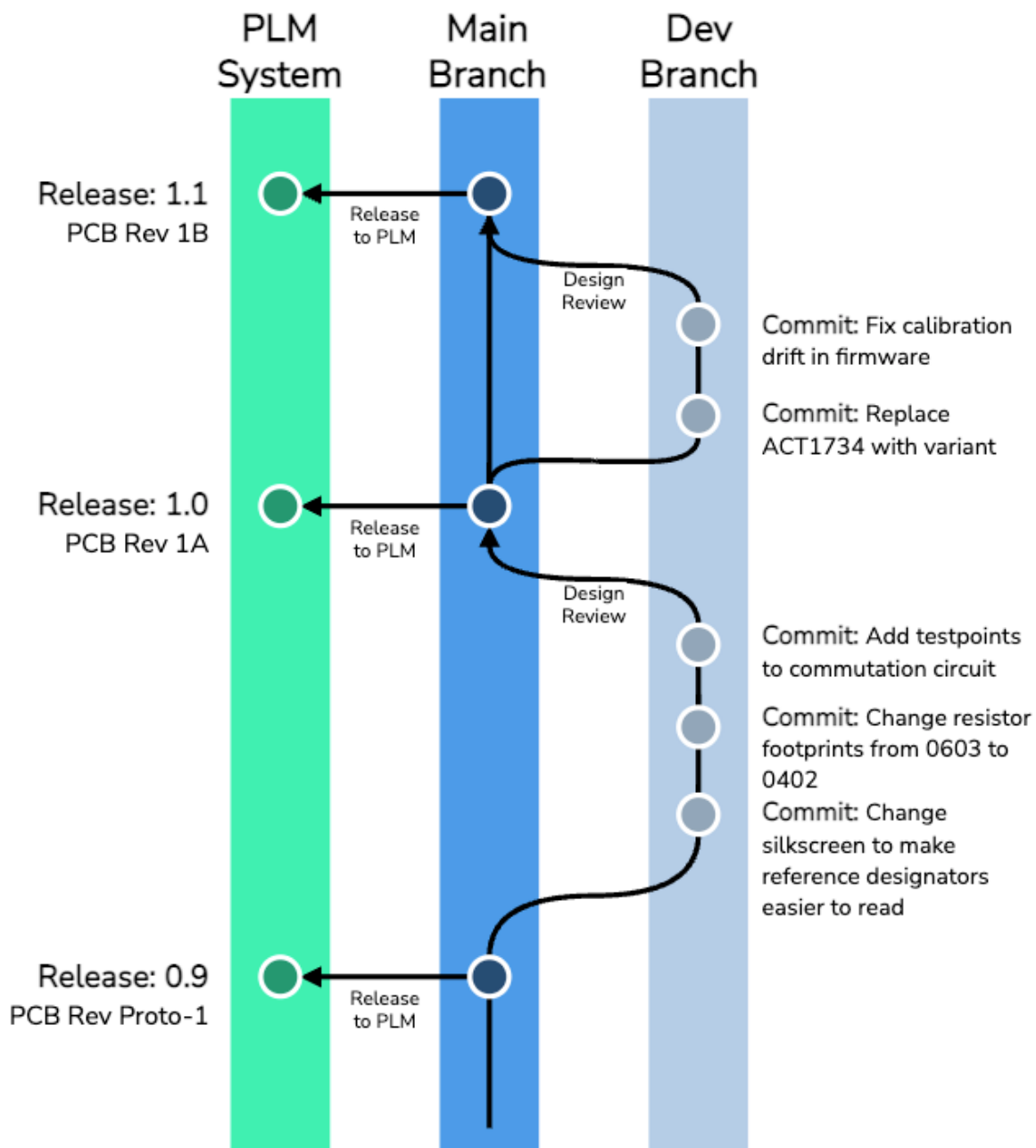
## Efetivando alterações (commit)

Com as alterações adicionadas à staging area, elas estão prontas para serem efetivas. O comando `git commit` torna as alterações persistentes no repositório.

```
git commit -m "Descrição da alteração"
```

A descrição da alteração é importante para organizar a versão e manter outros programadores cientes de que tipo de alterações foram realizadas naquele commit.

NOTA: Adicionando mais um comando `-m "descricao"` ao final do código acima é possível introduzir também uma descrição mais completa ao commit realizado



Two branch strategy Dev-&gt;Main

## Branches

Um branch é uma linha de desenvolvimento paralela geralmente utilizada para realizar modificações seguras sem interferir nas outras linhas.

É uma forma de reduzir os conflitos ao introduzir múltiplas features ao software.

```
// Cria um novo branch chamado developer
git branch developer
// Deleta um branch chamado developer
git branch -d developer
// Exibe os branches ativos
git branch -a
// Altera o branch atual para developer
git checkout developer
// Cria um branch vazio chamado developer
git checkout --orphan developer
```

## Merge

O comando `merge` é responsável por incorporar as modificações de outro branch.

```
// Incorpora as modificações do branch developer no master  
git checkout master  
git merge developer
```

## Git Remote

Uma das funções do git é a possibilidade de estender a utilização do repositório git para um endereço remoto, oferecendo um local centralizado para compartilhamento e gerencia do código do projeto.

## Clonando um repositório

Para clonar um repositório git já existente, utilizamos o comando: `git clone [url]`

```
git clone https://gitlab.com/rVenson/linguagemdeprogramacao.git
```

Também é possível clonar um repositório que esteja acessível no sistema de arquivos da máquina atual, como por exemplo:

```
git clone /c/Users/rvenson/Documents/LinguagemDeProgramacao
```

## Configurando repositório local

O comando `git remote add origin <servidor>` permite adicionar um atalho para o link do repositório remoto que queremos utilizar.

```
git remote add origin https://gitlab.com/rVenson/linguagemdeprogramacao.git
```

Neste exemplo, adicionaremos um atalho chamado **origin** ao endereço **https** especificado. Utilize o comando `git remote -v` para verificar os repositórios configurados.

## Atualizando o repositório

Para atualizar seu repositório git local com as alterações mais recentes do repositório remoto, podemos utilizar os seguintes comandos:

`git fetch` : este comando atualiza as referências do projeto local, no entanto, não executa nenhuma mudança

`git pull` : este comando atualiza as referências do projeto local e executa as mudanças (merge)

## Enviando alterações

Após incluir novos commits ao seu repositório local você pode unir estas alterações ao repositório remoto, para que outros contribuidores possam acessar. Basta utilizar o comando `git push`.

Exemplos:

```
git push ou git push origin ou git push origin master
```

## .gitignore

O arquivo `.gitignore`, geralmente presente na pasta raiz do projeto, é responsável por especificar os arquivos e pastas que não devem ser versionados pelo projeto. Este arquivo é de extrema importância para que arquivos residuais, temporários e builds não sejam integrados ao controle de versão.

Para utilizar o `gitignore`, basta criar um arquivo com este mesmo nome na pasta principal do projeto. Dentro do arquivo, deve-se especificar linha por linha quais arquivos, tipos de arquivos e pastas devem ser recusados pelo git.

```
# Ignora toda a pasta bin dentro do projeto
bin/

# Ignora todos os arquivos com final .import
*.import

# Ignora o arquivo project.config
project.config
```

Cada IDE geralmente possui conjunto de arquivos e pastas que podem ser ignoradas, para saber quais são estes arquivos, você pode consultar o site <https://www.gitignore.io/> e gerar o arquivo .gitignore padrão.

## Outros Materiais

- Fracz - Exercícios Interativos ([link](#))
- Git Kata ([link](#))
- Git Branching ([link](#))
- Git Handbook ([link](#))
- Guia Prático ([link](#))

## O que aprendemos hoje

- O que é um controle de versão
- Como versionar um código usando a ferramenta git
- Como manipular diferentes branches
- Como enviar código para outro repositório