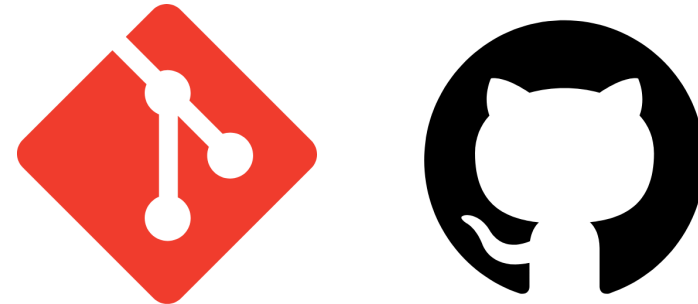


# Desenvolvimento Web I

## Git e GitHub

Primeiros Passos



# Roteiro

---

- Versionamento de Código
- Introdução ao Git
- Principais Comandos Git
- GitHub
- Conventional Commits
- Referências

# O que é versionamento de código?

---

Versionamento de código é uma técnica (*não um software*) que ajuda a **gerenciar o código-fonte** de uma aplicação

- Existe um registro de **todas as modificações** de código, podendo também **revertê-las**;
- Podemos criar versões de um software em **diferentes estágios**, podendo **alternar** facilmente entre elas;
- Cada membro da equipe pode trabalhar em uma versão diferente;
- Existem ferramentas de software específicas para trabalhar com controle de versão: `git` e `SVN` ;
- O Git é a **ferramenta mais utilizada** para esse propósito.

# Versionar código pra que?

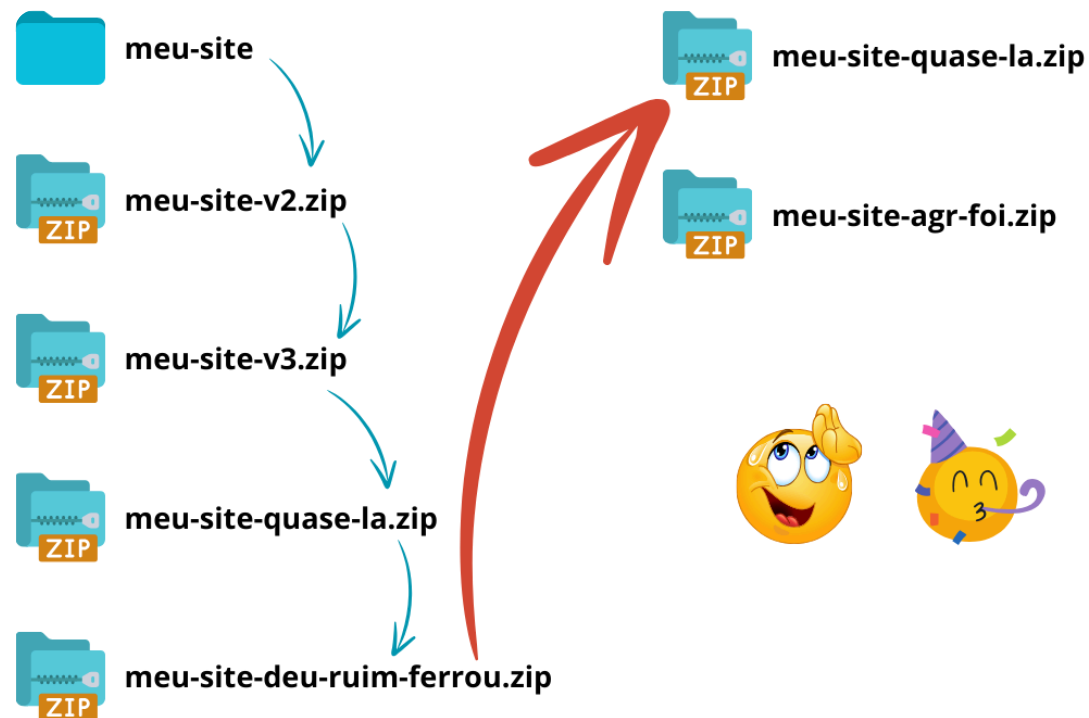
---

Em poucas palavras: para não perder o que foi feito

- Imagine que você está desenvolvendo um site para um cliente;
- Provavelmente você vai desenvolver esse site durante um determinado período de tempo, fazendo **atualizações e modificações frequentes**;
- Se por algum motivo uma modificação que você tenha feito gerar um problema e você queira **retornar a uma versão anterior do seu desenvolvimento**?

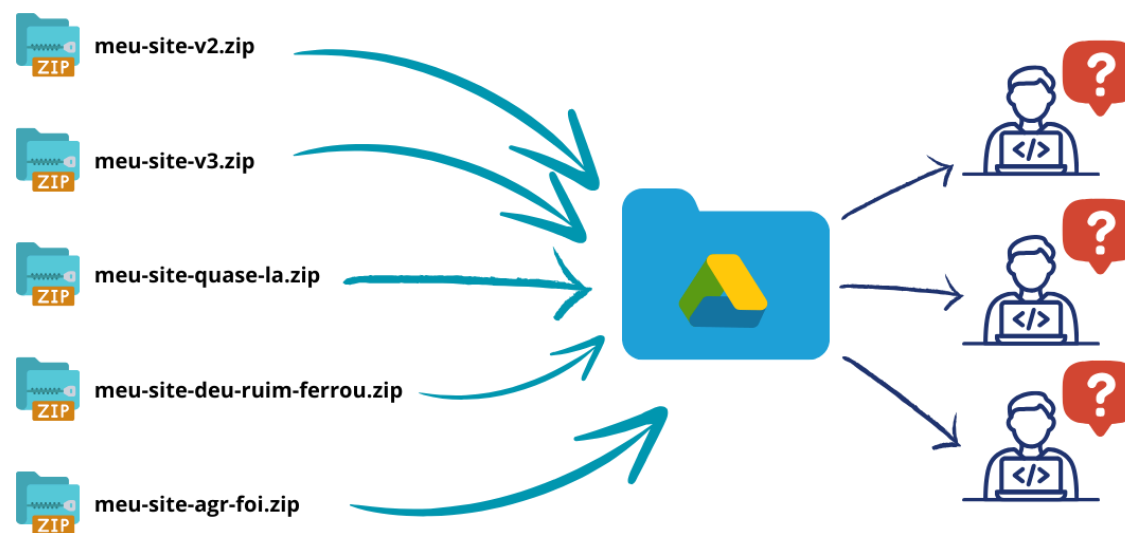
# Versionar código pra que?

Em poucas palavras: para não perder o que foi feito



# Versionar código pra que?

Em poucas palavras: organizar trabalho em equipe



# O que é Git?

---

O Git é o sistema de versionamento de código **mais utilizado** no mundo atualmente.

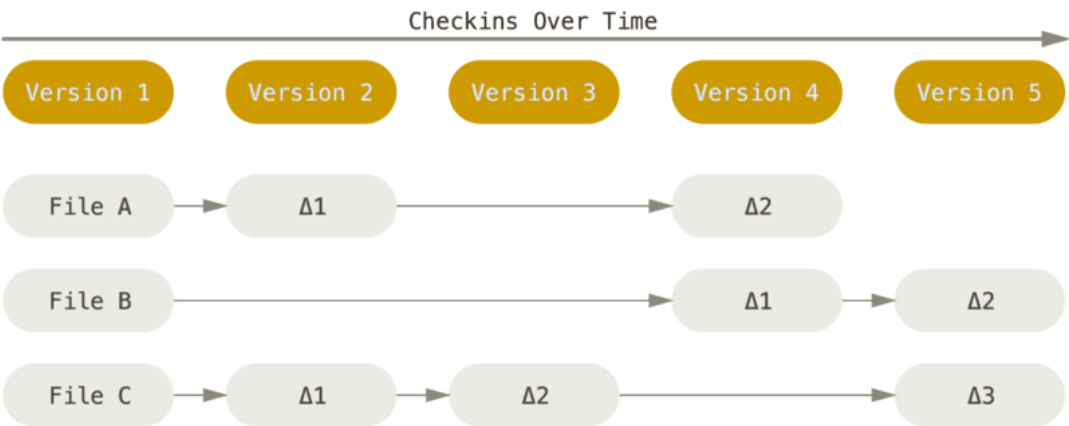
- Totalmente baseado em **repositórios**;
- Um repositório contém todas as **versões do código de um projeto**, bem como as **cópias** de cada membro da equipe;
- Geralmente ao iniciar um novo projeto, iniciamos também um novo repositório com o Git;
- Repositórios pode ser **locais ou remotos**;
- O Git trabalha com **criptografia** para proteger todos os objetos do repositório;
- Git é **open source**.

**! Git não é mesma coisa que GitHub**

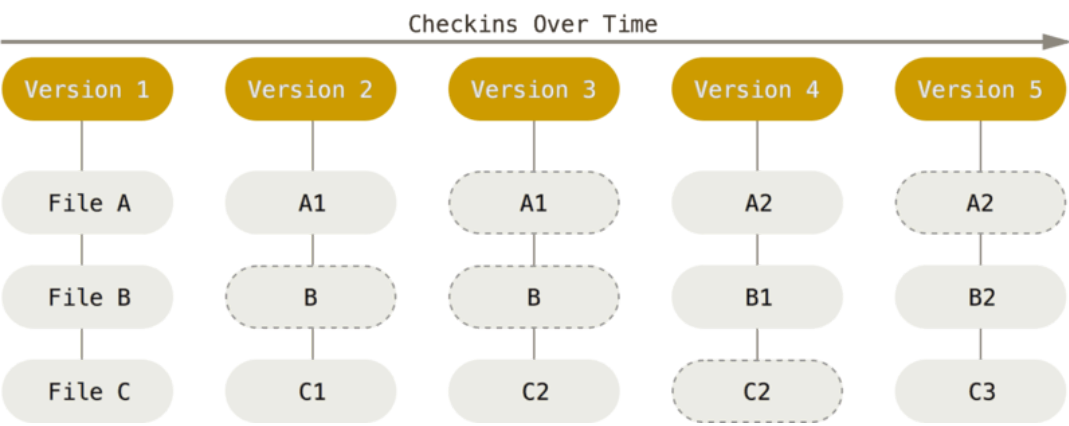
# Como o Git funciona?

O Git trata os dados como um conjunto de imagens de um sistema de arquivos (*snapshots*)

## - Outros CVS



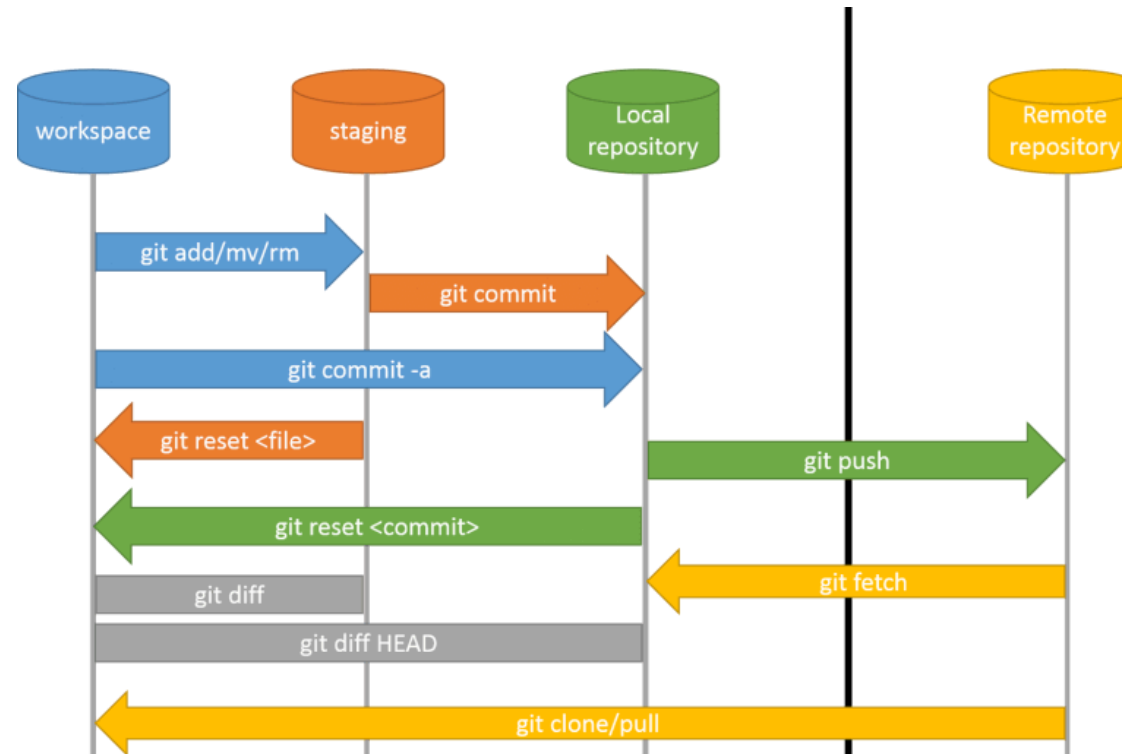
## - Git





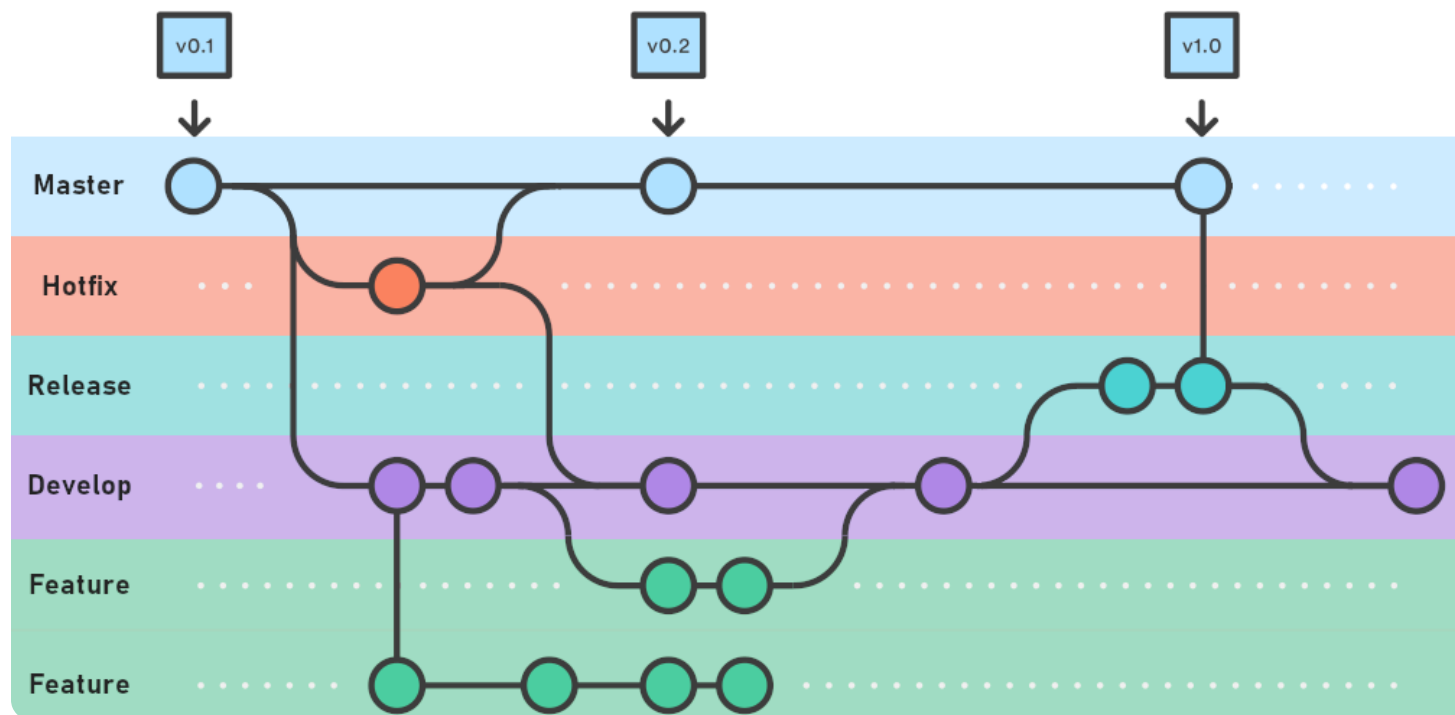
# Como o Git funciona?

O Git é um CVS distribuído



# Como o Git funciona?

O Git trabalha com ramificações de código (*branches*)



# O que é um repositório?

---

Local onde o código do projeto é armazenado.

- O repositório é criado, geralmente, ao iniciar um projeto;
- Podemos utilizar **servidores remotos** especializados em gerenciamento de repositórios;
- Com repos remotos, podemos trabalhar de forma **colaborativa** com outros membros da equipe sem se preocupar com questões de performance e gerenciamento dos repositórios;
- Cada membro pode baixar um repositório remoto e **criar versões diferentes** em sua máquina local.

# Criação de repositórios

---

Para criar um novo repositório utilizamos o comando `git init` na pasta do projeto.

- Ao inicializar um repositório, o git **cria uma pasta oculta chamada `.git`**, bem como os arquivos necessários ao repositório dentro dessa pasta;
- Após a criação do repositório o git **reconhece o diretório atual como um projeto**;
- Podemos então aplicar os demais comandos git para fazer o gerenciamento necessário do repositório.

# Criação de repositórios

Para criar um novo repositório utilizamos o comando `git init` na pasta do projeto.

- Antes de criar o repositório, não conseguimos executar nenhum comando git:

```
Lucas Mendes@Lucas-PC MINGW64 ~/Documents/Projetos - cursos/git-udemy
$ git status
fatal: not a git repository (or any of the parent directories): .git
```

- Após inicializar um repositório, temos um retorno diferente:

```
Lucas Mendes@Lucas-PC MINGW64 ~/Documents/Projetos - cursos/git-udemy
$ git init
Initialized empty Git repository in C:/Users/Lucas Mendes/Documents/Projetos - c
ursos/git-udemy/.git/

Lucas Mendes@Lucas-PC MINGW64 ~/Documents/Projetos - cursos/git-udemy (master)
$ git status
On branch master

No commits yet
```

# Comandos Git

## Principais comandos

- Inicializar um repositório:
  - `git init` — cria um novo repositório local em um diretório existente
  - `git clone [url]` — clona um repositório existente em um servidor remoto (repositório no GitHub, por exemplo)
    - Exemplo: `git clone https://github.com/EbookFoundation/free-programming-books.git`
- Comandos básicos:
  - `git status` — verifica o estado dos arquivos
  - `git add [arquivos]` — adiciona arquivos que serão monitorados pelo git para um próximo commit (estado de *staging*)
    - `git add .` — adiciona todos os arquivos modificados

# Comandos Git

## Principais comandos

- Comandos básicos:
  - `git commit` — armazena o estado atual do índice (arquivos monitorados) em um novo commit no repositório local
    - `git commit -m "mensagem"` — no commit, devemos especificar uma mensagem de registro e descrição
    - fazemos o commit geralmente depois de usar o comando `git add`
  - `git push` — transfere commits a partir do seu repositório local para um repositório remoto
  - `git pull` — faz o inverso do `push`, atualizando o repo local com as alterações do repo remoto

# Comandos Git

## Configurações básicas a serem feitas

- A primeira coisa a se fazer quando instalar o Git é definir o seu nome de usuário e endereço de e-mail
- Isso é importante porque todos os commits no Git utilizam essas informações
- Configurando a identidade de forma global:
  - `git config --global user.name "John Doe"`
  - `git config --global user.email johndoe@example.com`
- Para configurar a identidade para um repositório específico é só retirar a opção `--global`
- Para resetar essas configurações é necessário usar a opção `--unset` após o `config --global`
  - Exemplo: `git config --global --unset user.name`



# Comandos Git

## Configurações básicas a serem feitas

- Vincular repositório local a um repositório remoto:
  - `git remote add origin <url_do_repositorio_remoto>` — *origin* é um nome que damos ao repo remoto (pode ser outro nome)
  - `git push -u origin main` — *main* é a branch para onde vamos enviar as modificações
    - com a opção `-u` ( `--set-upstream` ), estamos definindo o repo remoto e a branch padrões
    - após a configuração do destino padrão de *upstream*, basta utiliza o comando `git push` nas próximas vezes

# Comandos Git

---

## Básico de gestão de branches

- O que são **branches**?
  - Um branch é uma linha de desenvolvimento independente dentro de um repositório Git (ramificação)
  - Permite que você trabalhe em funcionalidades ou correções sem afetar o código principal
  - O branch principal padrão é o main (ou master em projetos mais antigos)
  - **Vantagem:** Facilita o trabalho em equipe e a organização de novas features

# Comandos Git

---

## Básico de gestão de branches

- Operações básicas com **branches**
  - `git branch <nome_do_branch>` — cria uma branch
  - `git checkout <nome_do_branch>` — troca a branch ativa
  - `git checkout -b <nome_do_branch>` — atalho para criar e trocar a branch

# Comandos Git

---

## Básico de gestão de branches

- Fluxo básico de trabalho

### 1. Criar uma branch para uma tarefa:

- `git checkout -b feature/nova-funcionalidade`

### 2. Fazer alterações e commits na branch:

- `git add .`
- `git commit -m "Implementa nova funcionalidade"`

### 3. Voltar à branch principal:

- `git checkout main`

# Comandos Git

## Básico de gestão de branches

- Fluxo básico de trabalho - *integração de alterações*

### 1. Mesclar uma branch no principal:

- `git merge <nome_do_branch>`

### 2. Resolver conflitos (se necessário):

- Edite os arquivos conflitantes
- Finalize o **merge**:
  - `git add .`
  - `git commit`

### 3. Excluir a branch após o **merge**:

- `git branch -d <nome_do_branch>`

## E GitHub, o que é?

O GitHub gerencia repositórios de forma remota.

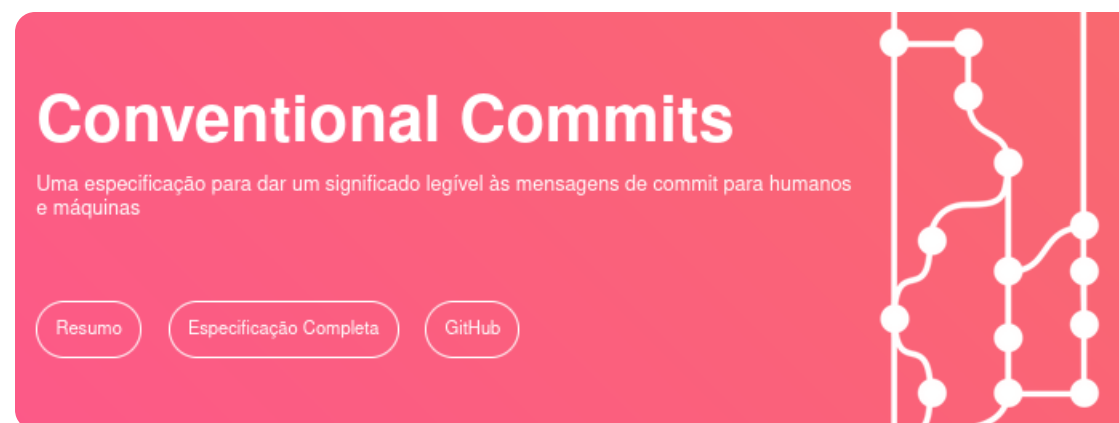
- Serviço **gratuito e amplamente utilizado**;
- Podemos disponibilizar nossos projetos para outros devs, mantendo o **repositório centralizado**;
- Pode ser integrado em serviços de hospedagem e servidores de produção;
- O GitHub possibilita a criação de **projetos públicos e privados** de forma gratuita;
- Link para o GitHub: <https://github.com/>

⚠ Até 2019 o GitHub tinha limitações em relação aos repositórios privados, o que fez com que outras ferramentas ganhassem destaque, como o GitLab e o Bitbucket.

# Padrão de Commits

O padrão *Conventional Commits*

- A especificação do [Conventional Commits](#) é uma convenção simples para **padronizar as mensagens de commit**.
- A ideia é **facilitar a compreensão** das alterações realizadas em um determinado commit.
- Além disso, seguindo essas regras, torna-se possível a criação de **ferramentas automatizadas**.



# Padrão de Commits

O padrão *Conventional Commits*

- A mensagem de *commit* deve ser estruturada da seguinte forma:

```
<tipo>[(escopo opcional)]: <descrição>
```

```
[corpo opcional]
```

```
[rodapé(s) opcional(is)]
```

- Exemplo:

```
feat(lang): adiciona tradução para português brasileiro
```

```
veja o ticket para detalhes sobre as bibliotecas e dependências utilizadas
```

```
Refs #133
```



# Padrão de Commits

---

## *Commits Semânticos no Conventional Commits*

- O commit semântico possui os elementos estruturais abaixo (tipos), que informam a intenção do seu commit ao utilizador(a) de seu código.
  - **feat:** - Commits do tipo **feat** indicam que seu trecho de código está incluindo um **novo recurso (feature)**;
  - **fix:** - Commits do tipo **fix** indicam que seu trecho de código commitado está **solucionando um problema**;
  - **docs:** - Commits do tipo **docs** indicam que houveram **mudanças na documentação**, como por exemplo no Readme do seu repositório;
  - **build:** - Commits do tipo **build** são utilizados quando são realizadas modificações em **arquivos de build e dependências**;
  - **chore:** - Commits do tipo **chore** indicam **atualizações de tarefas** de build, configurações de administrador, pacotes;

# Padrão de Commits

---




## *Commits Semânticos no Conventional Commits*

- O commit semântico possui os elementos estruturais abaixo (tipos), que informam a intenção do seu commit ao utilizador(a) de seu código.
  - **test:** - Commits do tipo **test** são utilizados quando são realizadas **alterações em testes**;
  - **style:** - Commits do tipo **style** indicam que houveram alterações referentes a **formatações de código**, semicolons, trailing spaces, etc.;
  - **refactor:** - Commits do tipo **refactor** referem-se a mudanças devido a **refatorações que não alterem sua funcionalidade**;
  - **cleanup:** - Commits do tipo **cleanup** são utilizados para remover código comentado ou qualquer outra forma de **limpeza do código-fonte**;
  - **remove:** - Commits do tipo **remove** indicam a **exclusão de arquivos**, diretórios ou funcionalidades obsoletas ou **não utilizadas**

# Padrão de Commits

## Padrões de emojis

- Usar um emoji no início da mensagem de commit pode ser útil para facilitar a identificação do tipo de alteração realizada. Veja os exemplos a seguir:

Tipo do commit	Emoji	Palavra-chave
Bugfix	 <code>:bug:</code>	<code>fix</code>
Documentação	 <code>:books:</code>	<code>docs</code>
Novo recurso	 <code>:sparkles:</code>	<code>feat</code>

- Veja mais sobre **Conventional Commits** em: <https://github.com/iuricode/padroes-de-commits>

# Materiais e Links Úteis

---

Se aprofunde em Git e GitHub com esses materiais e ferramentas úteis

- <https://www.gitfluence.com/> — ferramenta baseada em IA para auxiliar a encontrar o comando Git adequado para uma determinada situação (você pode fazer perguntas em linguagem natural)
- <https://comandosgit.github.io/> — site com a listagem e breve explicação dos comandos Git
- <https://git-scm.com/book/pt-br/v2> — versão digital do livro **Pro Git**
- <https://readme.so/pt> — auxilia na criação de arquivos de **README** profissionais e bem organizados
- <https://www.cursoemvideo.com/curso/curso-de-git-e-github/> — curso gratuito de Git e GitHub do Curso em Vídeo (Prof. Gustavo Guanabara)
- <https://youtu.be/2c7yWlpWDJM?feature=shared> — curso gratuito de Git e GitHub do Canal Tiago Matos

# Referências e Materiais Complementares

---

- Materiais sobre Git e GitHub no W3Schools: <https://www.w3schools.com/git/default.asp>
- Conventional Commits:
  - Documentação oficial do Conventional Commits: <https://www.conventionalcommits.org/pt-br/v1.0.0/>
  - Repositório didático com recomendações sobre Conventional Commits: <https://github.com/iuricode/padroes-de-commits>
- Cursos Gratuitos no YouTube:
  - Canal Tiago Matos: <https://youtu.be/2c7yWlpWDJM?feature=shared>
  - Canal Curso em Vídeo (Gustavo Guanabara): <https://www.youtube.com/live/xEko29OWILE?feature=shared>