



# Aula 06

## Publicar Aplicações WEB



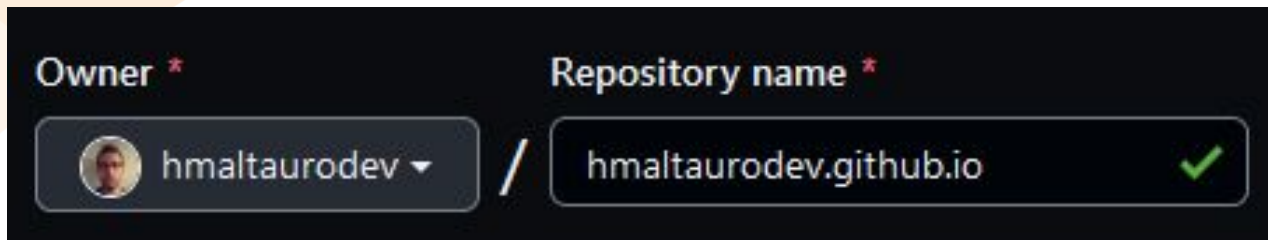
**Criem uma conta no site**

**<https://github.com>**


**De preferência com um nome  
de usuário decente**

**Criar um repositório novo  
no github, com o nome**

**nomeDeUsuario.github.io**



A screenshot of the GitHub repository creation interface. It features two main input fields on a dark background. The first field, labeled 'Owner' with a red asterisk, contains a user profile picture and the text 'hmaltaurodev' followed by a dropdown arrow. The second field, labeled 'Repository name' with a red asterisk, contains the text 'hmaltaurodev.github.io' and a green checkmark icon, indicating the name is valid. A slash '/' is positioned between the two fields.

Owner *	Repository name *
 hmaltaurodev ▼	hmaltaurodev.github.io ✓



**GIT**

**Versionamento de  
Código**

# GIT

- É um sistema de controle de versões, usado principalmente no desenvolvimento de software
- Inicialmente projetado e desenvolvido por **Linus Torvalds** para o desenvolvimento do kernel **Linux**, mas hoje é utilizado por quase toda a indústria de desenvolvimento de software



# GIT

- Trabalha com repositórios
- Imagine que cada projeto de software, é uma pasta diferente, um repositório diferente
- Dentro desse repositório, existe todo o histórico de alterações que foram realizadas naqueles arquivos

# GIT

- Existem diversos outros sistemas de versionamento, como **CVS** e **SVN**
- Porém nenhum deles se compara ao git, nos quesitos de desempenho, segurança e flexibilidade
- Isso que torna o git, o sistema de versionamento mais utilizado no mundo



# GIT

- Git e GitHub são coisas diferentes
  - Git é uma sistema de versionamento
  - GitHub é uma plataforma para hospedagem de repositórios git



# GIT

- Existem diversas plataformas de hospedagem de repositórios git
  - GitHub
  - GitLab
  - GitBucket
  - BitBucket
  - Azure DevOps

# GIT

- O git funciona através de comandos, que são executados em um terminal de comandos
- O git possui o próprio terminal de comandos, chamado **Git Bash**

MINGW64:/c/Henrique/Senac/git

Maltauro@DESKTOP-37E7RSK MINGW64 /c/Henrique/Senac/git (main)

\$ |

# GIT

- git config
  - Toda vez que você estiver usando uma instalação nova do git, você deve configurar seu nome de usuário e seu email
  - Neste caso, vamos fazer uso do nome de usuário e email que vocês utilizaram na conta do GitHub



*// Configura o nome do usuário*

```
git config --global user.name "nomeDoUsuario"
```

*// Configura o email do usuário*

```
git config --global user.email "email@email.com"
```



# GIT

- `git init`
  - Inicializa aquela pasta do computador, como um repositório git



*// Inicializa o repositório git*  
`git init`



# GIT

- git status
  - Mostra a status dos arquivos daquele repositório git
  - Se tem arquivos novos
  - Se tem arquivos que estão sendo removidos
  - Se tem arquivos que estão sendo modificados





*// Mostra o status dos arquivos*  
`git status`



# GIT

- `git add`
  - Prepara os arquivos para serem salvos no repositório



*// Prepara TODOS os arquivos para serem salvos*  
`git add .`

*// Prepara um arquivo para ser salvo*  
`git add nomeDoArquivo`

# GIT

- git commit
  - Confirmar o salvamento dos arquivos naquele repositório
  - Só confirma o salvamento de arquivos que estejam preparados para serem salvos, ou seja, que tenham passado pelo git add
  - É obrigatório informar uma mensagem de commit



*// Confirmar o salvamento dos arquivos*  
`git commit -m "Mensagem de Commit"`

# GIT

- git diff
  - Mostra as diferenças dos arquivos, entre o último commit e as alterações atuais do arquivo



*// Mostra as diferenças de TODOS os arquivos*

`git diff`

*// Mostra as diferenças de um arquivo específico*

`git diff nomeDoArquivo`



# GIT

- Branch
  - Ramificações
  - Quase todo sistema de versionamento trabalha com ramificações, e o git é uma delas
  - Todo repositório git começa com uma branch principal, normalmente chamada de **main** ou **master**





# GIT

- Branch

- Imagine uma branch, como se fosse uma “cópia” de outra branch
- Nessa “cópia” você pode fazer o que quiser, remover arquivos, alterar arquivos, adicionar arquivos novos
- A branch “original” se mantém sem alterações, somente a “cópia” possui essas alterações que foram realizadas



# GIT

- git log
  - Lista todos os commits realizados naquela branch



*// Lista todos os commits realizados naquela branch*  
`git log`



# GIT

- git branch
  - Permite listar, criar ou remover uma branch do repositório git



*// Lista todas branches*

`git branch`

*// Cria uma nova branch*

`git branch nomeDaBranch`

*// Remove uma branch*

`git branch -d nomeDaBranch`

# GIT

- git checkout
  - Permite a troca de branch ativa
  - Só permite a troca de branch, se a branch atual não possuir nenhum arquivo pendente de confirmação, ou seja, que não esteja pendente de passar pelo git commit



*// Troca de branch ativa*  
`git checkout nomeDaBranch`

*// Cria uma branch nova, e troca de branch ativa*  
`git checkout -b nomeDaBranch`



# GIT

- git merge
  - Combina os commits de duas branches
  - Ou seja, vai fazer com que uma das branches receba todas as alterações que foram realizadas na outra branch
  - **IMPORTANTE:** você precisa estar na branch que vai receber as alterações





*// Combina os commits de duas branches*  
`git merge nomeDaBranch`