
SOFTEX
PERNAMBUCO

Softex

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

# Aula 16 | Módulo: Git e GitHub (continuação)

- ➔ Criação e troca de branch: git branch, checkout, merge
- ➔ Resolução básica de conflitos e entendimento do fluxo de merge
- ➔ Boas práticas de commits e .gitignore
- ➔ Colaboração via GitHub: forks e pull requests
- ➔ Publicação de projetos estáticos com GitHub Pages

# Abrindo editor de código

## Vamos agora abrir o VSCode e criar os arquivos

- Com o programa aberto, clique em File > Open Folder... (Arquivo > Abrir Pasta...).
- Escolha um local para criar a sua pasta, crie uma nova pasta e dê o nome de **seunome_aula_16**. Depois dê dois clique nessa pasta criada e clique em **Selecionar pasta**. O VSCode reabrirá dentro dessa pasta que foi criada.
- Agora vamos criar os arquivos HTML, JS e CSS:
- Dê o nome de **index.html** , **script.js** e **style.css**

# Git e GitHub (continuação)

## O que já vimos?

- Instalação do Git
- Ciclo: add > commit > push
- GitHub (clonar, criar repositório, push/pull)

# Git e GitHub (continuação)

**Antes de seguir, vamos limpar possíveis credenciais no Windows**

- Isso que iremos fazer abaixo, apaga logins salvos por outros alunos.
- Abra o Gerenciador de Credenciais:
  - Clique na lupa do Windows e digite **Gerenciador de Credenciais**
- Vá em Credenciais do Windows
- Procure entradas com **git:** ou **github.com**
- Clique e escolha Remover

# Git e GitHub (continuação)

**Antes de seguir, vamos limpar possíveis credenciais no Windows**

- Verificar Configuração Local do Git:
  - No terminal, digite o comando: **git config --list**
  - Se aparecer nome/email de outro aluno, vamos apagar.
- Apagar configuração global:
  - **git config --global --unset user.name**
  - **git config --global --unset user.email**
- Depois verifique novamente se ainda existe configuração:
  - **git config --list**

# Git e GitHub (continuação)

## Instalação e Configuração

- Baixar e instale o Git: **https://git-scm.com**
- Depois que o Git estiver instalado, entre na pasta do seu projeto, abra o terminal (no VSCode vá no menu: **Terminal > New Terminal**) e digite os comandos abaixo:
  - Inicializar o Git no projeto:
    i. **git init**
  - Configurar Git globalmente:
    i. **git config --global user.name "Seu Nome"** *(seu usuário do GitHub)*
    ii. **git config --global user.email "seuemail@exemplo.com"** *(seu email do GitHub)*
  - Visualizar se foi cadastrado corretamente:
    i. **git config --list**

# Git e GitHub (continuação)

## Autenticação Segura

- O GitHub não aceita mais senha de conta para login no Git.
- É necessário usar:
  - Token Pessoal de Acesso (PAT) ou
  - Autenticação via navegador (mais fácil).
- Exemplo prático (primeiro push):
  - O Git abre o navegador, você entra com login e senha e pronto.
  - O token fica salvo apenas nessa sua máquina até limparmos no final.

## Limpeza Final (Encerrando a Aula)

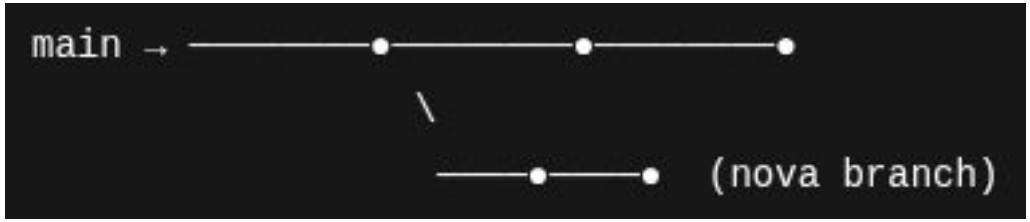
- Volte ao **Gerenciador de Credenciais** e remova qualquer dado relacionado a github.
- Rode os comandos abaixo no terminal:
  - **git config --global --unset user.name**
  - **git config --global --unset user.email**

Assim, nenhum próximo aluno conseguirá usar sua conta.

# Git e GitHub (continuação)

## O que é uma Branch?

- Branch = ramificação dentro do projeto.
- Permite criar uma “cópia” do código em que você pode trabalhar sem mexer no tronco principal (main).
- Usado para:
  - Criar novas funcionalidades.
  - Testar ideias sem arriscar o projeto.
  - Corrigir erros isoladamente.




Exemplo visual

# Git e GitHub (continuação)

## Por que usar Branches?

- Trabalho em equipe fica organizado.
- Evita que todos editem o mesmo arquivo da main ao mesmo tempo.
- Dá segurança: só unimos (merge) quando estiver pronto.
- Fluxo básico de como trabalhamos com branch:
  - Verifica em qual branch você está: **git branch** (para sair da visualização, digite **\q**)
  - Suponha que você esteja na branch **main** e precisa criar uma nova funcionalidade.
  - Então você criar uma nova branch:
    i. Comando: **git checkout -b nova-funcionalidade**
  - Faz então as mudanças necessárias e commits
  - Voltar para main:
    i. Comando: **git checkout main**
  - Juntar as alterações (estando na main): **git merge nova-funcionalidade**

# Git e GitHub (continuação)

## Comandos básicos de Branch

- **git branch** - lista as branches existentes
- **git branch nome** - apenas cria uma nova branch
- **git checkout -b nome** - cria e já muda para a nova branch
- **git checkout nome** - muda para a branch
- **git merge nome** - mescla branch “nome” na atual

# Git e GitHub (continuação)

## Exercício guiado

- Pressuponho que já tenha configurado as credenciais no computador.
- Dentro do VSCode conectado na pasta da aula de hoje, abra o terminal.
- Vamos iniciar o git nessa pasta: **git init**
- Agora vamos criar/renomear a branch principal: **git branch -M main**

# Git e GitHub (continuação)

## Exercício guiado

- Crie uma página básica no **index.html** e configure o **style.css** para ela

index.html > ...

```html
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <title>Aula 16</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Bem-vindo</h1>
    <p>Página inicial do projeto.</p>
</body>
</html>
```

style.css > ...

```css
body {
    font-family: Arial, sans-serif;
    margin: 40px;
}

h1 {
    font-size: 28px;
}
```

# Git e GitHub (continuação)

## Exercício guiado

- Agora vamos versionar o projeto rodando os comandos abaixo:
  - **git status**
  - **git add .**
  - **git commit -m "Cria estrutura inicial (index.html e style.css)"**
  - **git log --oneline** (para sair da visualização, digite **\q**)

## Exercício guiado

- Agora vamos criar uma branch a partir dessa nossa atual (main) e trabalhar nela:
  - **git checkout -b cabecalho** (isso vai criar uma nova branch com uma cópia dessa atual que estamos, que é a main). O comando -b já conecta na branch criada.
  - **git branch** (esse comando verifica qual branch estamos, deve mostrar * cabecalho). Para sair da visualização, digite **\q**.

# Git e GitHub (continuação)

## Exercício guiado

- Edite o **index.html** (adicione um cabeçalho).

```
8   <body>
9       <header>Cabeçalho</header>
10      <h1>Bem-vindo</h1>
11      <p>Página inicial do projeto.</p>
12  </body>
13  </html>
```

# Git e GitHub (continuação)

## Exercício guiado

- Agora versione a branch:
  - **git status**
  - **git add .**
  - **git commit -m "Adicionado cabeçalho"**

# Git e GitHub (continuação)

## Exercício guiado

- Troque para a branch main e observe os arquivos:
  - **git checkout main**
- Na branch main, o cabeçalho que criamos não aparece nem os estilos aplicados no css (os arquivos voltam ao estado do commit inicial).
- Volte para a branch que criamos e veja as mudanças que fizemos:
  - **git checkout cabecalho**

# Git e GitHub (continuação)

## Exercício guiado

- Agora vamos fazer o merge da branch cabecalho na main.
- Volte para a main e faça o merge
  - **git checkout main**
  - **git merge cabecalho**
- A mensagem de Fast-forward (sem conflitos) deve aparecer no terminal.
- Abra o **index.html** e confirme que o cabeçalho agora está na main.

## Exercício guiado

- Opcionalmente você pode remover a branch que foi criada para fazer o trabalho separado.
- Para apagar a branch já mesclada, rode o comando abaixo:
  - **git branch -d cabecalho**
  - **git branch** # deve restar só main (para sair da visualização, digite **\q**)

# Git e GitHub (continuação)

## O que é um conflito no Git?

- Conflito aparece quando o Git não consegue mesclar automaticamente mudanças de duas branches.
- O caso mais comum: duas pessoas alteraram as mesmas linhas do mesmo arquivo em branches diferentes.
- Outros casos:
  - add/add: o mesmo arquivo novo criado nas duas branches com conteúdo diferente.
  - modify/delete: uma branch altera e a outra apaga o arquivo.
  - rename/rename: o mesmo arquivo foi renomeado de formas diferentes.
  - binários (imagens, .psd, .pdf): o Git não “entende” como mesclar; exige escolha manual.

# Git e GitHub (continuação)

## Como o Git sinaliza o conflito no arquivo

- O Git insere marcadores no arquivo para você decidir:

<<<<<<< HEAD
*(conteúdo da branch atual, onde você está. ex.: main)*
=======
*(conteúdo vindo da outra branch, a que está sendo mesclada. ex.: cabecalho)*
>>>>>>> cabecalho

- HEAD = sua branch atual.
- Incoming (ou nome da outra branch) = mudanças vindas do merge.
- Nunca commitar com marcadores <<<<<<<, =======, >>>>>>> no arquivo.
- Use git status para ver os arquivos com conflito “both modified”.

# Git e GitHub (continuação)

## Simulando um conflito

- Vamos criar uma nova branch para editar nosso **index.html**

index.html > ...

You, 9 minutes ago | 1 author (You)

```
1   <!DOCTYPE html>
2   <html lang="pt-BR">
3   <head>
4       <meta charset="UTF-8">
5       <title>Aula 16</title>
6       <link rel="stylesheet" href="style.css">
7   </head>
8   <body>
9       <header>Cabeçalho</header>
10      <h1>Bem-vindo</h1>
11      <p>Página inicial do projeto.</p>
12  </body>
13  </html>
```

# Git e GitHub (continuação)

## Simulando um conflito

- Rode o comando: **git checkout -b editando-header**
- Certifique que está na nova branch criada: **git branch**
- Agora vamos alterar o header atual para **<header>Cabeçalho do Site</header>**
- Versione a branch **editando-header**:
  - **git add index.html**
  - **git commit -m "Alterando header para Cabeçalho do Site na editando-header"**

```
8   <body>
9       <header>Cabeçalho do Site</header>
10      <h1>Bem-vindo</h1>
11      <p>Página inicial do projeto.</p>
12  </body>
```

# Git e GitHub (continuação)

## Simulando um conflito

- Voltar para a main:
  - **git checkout main**
- Edite o index.html na main, mas mude a MESMA linha do <header> para um texto diferente:
  - <header>Topo do Site</header>
- Versione a branch main:
  - **git add index.html**
  - **git commit -m "Altera header para Topo do Site na main"**
- Agora temos duas histórias divergentes que alteram a mesma linha em index.html. Isso causará um conflito.

# Git e GitHub (continuação)

## Simulando um conflito

- Estando na main, rode:
  - **git merge editando-header**
- Você deverá ver uma mensagem indicando CONFLICT no index.html.
- Abra o **index.html**. Você verá alguns marcadores do Git:
  - <<<<<<< HEAD (Current Change)
  - <header>Topo do Site</header>
  - =======
  - <header>Cabeçalho do Site</header>
  - >>>>>>> editando-header (Incoming Change)
- HEAD = sua branch atual (main) “Topo do Site”
- editando-header = mudanças que estão entrando “Cabeçalho do Site”

# Git e GitHub (continuação)

## Resolver o conflito (Opção A)

- Escolher a versão da main (HEAD):
- Linha de comando (pega o arquivo inteiro do lado “ours/HEAD”):
  - **git checkout --ours index.html**
  - **git add index.html**
  - **git commit -m "Resolve conflito mantendo header da main"**

# Git e GitHub (continuação)

## Resolver o conflito (Opção B)

- Escolher a versão da editando-header (incoming)
  - **git checkout --theirs index.html**
  - **git add index.html**
  - **git commit -m "Resolve conflito mantendo header do editando-header”**

# Git e GitHub (continuação)

## Resolver o conflito (Opção C)

- Combinar as duas (editar manualmente)
- Edite o arquivo removendo os marcadores e montando um texto final - por exemplo:
  - **<header>Topo do Site - Cabeçalho do Site</header>**
- Salve e finalize:
  - **git add index.html**
  - **git commit -m "Resolve conflito combinando títulos do header"**

# Git e GitHub (continuação)

## Conferir o resultado do merge

- Rode o comando: **git log --oneline --graph --decorate --all**
- Agora a main deve conter o commit de resolução e as mudanças do editando-header (além das suas).
- Abra o **index.html** e confirme que o <header> ficou como você decidiu na etapa de resolução.
- Agora podemos apagar a branch já mesclada:
  - **git branch -d editando-header**

# Git e GitHub (continuação)

**E se travar no meio do merge? (desfazer o merge em andamento)**

- Se quiser desistir da mesclagem e voltar ao estado anterior (disponível enquanto o merge ainda não foi finalizado/commitado):
  - **git merge --abort**

# Git e GitHub (continuação)

## Boas práticas no commit

- Cada commit é como uma foto do projeto em um momento.
- Um histórico limpo ajuda a:
  - Entender a evolução do código.
  - Encontrar bugs em versões passadas.
  - Trabalhar em equipe sem se perder.
- Mensagens ruins = confusão no futuro.

Imagine que alguém herde seu projeto daqui 6 meses.
O que ajuda mais?

- commit **‘ajustes’** ou
- commit **‘Corrige erro de validação no login’**

# Git e GitHub (continuação)

**Errei a mensagem, e agora?**

- Para alterar a mensagem do último commit:
  - **git commit --amend -m "Mensagem corrigida"**

**Atenção:** só use antes de enviar (push). Depois do push, o ideal é criar um novo commit corrigindo.

# Git e GitHub (continuação)

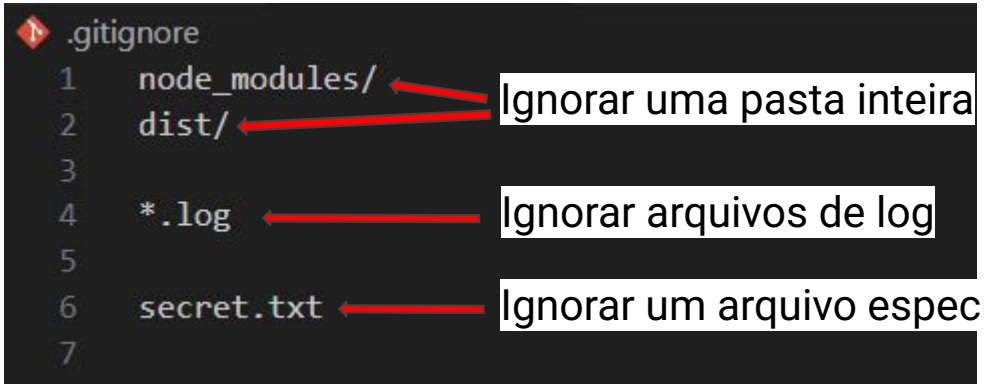
## O que é o .gitignore?

- É um arquivo de texto chamado .gitignore
- Nele listamos arquivos/pastas que o Git deve ignorar.
- Serve para:
  - Evitar subir arquivos desnecessários (logs, temporários).
  - Proteger informações sensíveis (senhas, configs pessoais).
  - Deixar o repositório mais limpo e organizado.
- Exemplo:
  - Você não quer enviar a pasta node_modules/ (pode ter milhares de arquivos).
  - Então adiciona node_modules/ no .gitignore.

# Git e GitHub (continuação)

## Como funciona o .gitignore

- Cada linha do .gitignore = um padrão de exclusão.
- Exemplos:




Dica: Arquivos já adicionados ao repositório não são ignorados retroativamente. Se já subiu, precisa removê-los do Git antes.

# Git e GitHub (continuação)

## Onde criar o .gitignore

- O .gitignore deve ficar na raiz do repositório.
- É versionado: ou seja, faz parte do projeto.
- Cada pessoa que clonar o projeto terá as mesmas regras.
- Depois de criar o arquivo, rode o comando:
  - **git add .gitignore**
  - **git commit -m "Adicionado regras de ignore"**

# Git e GitHub (continuação)

## GitHub ajuda com modelos

- O GitHub possui modelos para várias linguagens e frameworks.
- Acesse: **https://github.com/github/gitignore**
- Quando cria um repositório novo no GitHub, dá para escolher um .gitignore pronto (ex: Node, Python, Unity etc).

# Git e GitHub (continuação)

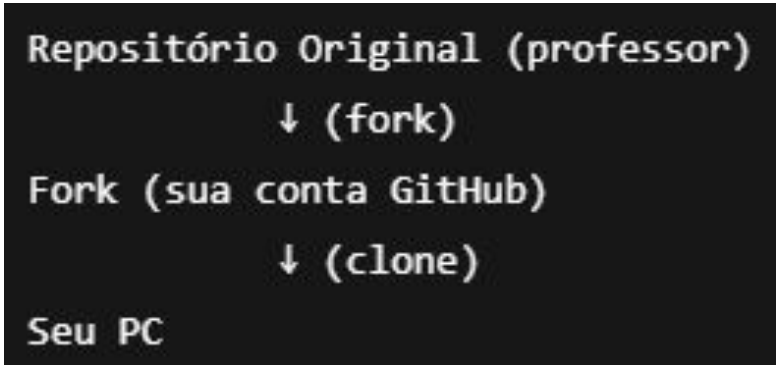
**Como várias pessoas trabalham no mesmo projeto?**

- O GitHub é uma plataforma que permite:
  - Hospedar repositórios.
  - Compartilhar projetos com o time.
  - Contribuir em projetos de outras pessoas.
- O fluxo de colaboração:
  - Fork ou Clone do repositório.
  - Criar branch para suas alterações.
  - Commit + Push.
  - Enviar Pull Request (PR) para revisão e integração.

# Git e GitHub (continuação)

## Qual a diferença entre Fork e Clone?

- Clone = cópia local de um repositório (normalmente do seu projeto).
- Fork = cópia do repositório para a sua conta no GitHub (geralmente quando o repositório é de outra pessoa/organização).
- Após o fork, você pode clonar o fork para o seu computador.



# Git e GitHub (continuação)

**Criando e trabalhando em uma branch no seu fork**

- Faça o Fork de um repositório existente para o seu Github.
- Sugiro cada um de vocês criar um repositório público para que outros colegas façam o fork do repositório e colaborem com o trabalho.
- No repositório que deseja colaborar, você encontrará um botão assim:


- Basta clicar e criar um Fork no seu repositório. Isso fará um Clono do repositório no seu Github.
- Depois que fizer o Fork, faça o clone do novo repositório criado no seu Github:
  - **git clone https://github.com/seu_usuario/nome_do_repositorio.git**

# Git e GitHub (continuação)

**Criando e trabalhando em uma branch no seu fork**

- Abra o VSCode no projeto clonado.
- Crie uma branch:
  - **git checkout -b minha-alteracao**
- Faça mudanças, adicione e commit:
  - **git add .**
  - **git commit -m "Adiciona meu nome na lista de alunos"**
- Envie para o seu fork:
  - **git push origin minha-alteracao**

# Git e GitHub (continuação)

## O que é um Pull Request (PR)?

- Um Pull Request é um pedido para que suas mudanças sejam adicionadas ao repositório original.
- Passos:
  - Acesse seu fork no GitHub (o mesmo que você enviou a atualização).
  - Clique em Compare & Pull Request (esse botão aparece no repositório).
  - Escreva título e descrição claros.
  - Envie: O dono do projeto revisa e decide aceitar (merge) ou pedir ajustes.

# Git e GitHub (continuação)

## O que é o GitHub Pages?

- Serviço do GitHub para publicar sites estáticos (HTML, CSS, JS).
- Gratuito e integrado ao repositório.
- URL gerada automaticamente:
  - https://seuusuario.github.io/nome-do-repositorio/
- Útil para:
  - Portfólios
  - Documentações
  - Projetos simples em HTML/CSS/JS

# Git e GitHub (continuação)

## Preparando o repositório

- Coloque os arquivos do site na raiz do repositório (ex.: index.html, style.css).
- O GitHub Pages sempre busca um index.html como página inicial.
- Confirme que o projeto já está no GitHub (push feito).

# Git e GitHub (continuação)

## Ativando o GitHub Pages

- Vá no repositório: Settings.
- Role até a seção Pages.
- Em Branch, escolha:
  - main
  - /root (pasta raiz)
- Clique em Save.
- O GitHub gera um link: pode levar alguns minutos para ficar disponível.

# ATÉ A PRÓXIMA AULA!

*Front-end - Design. Integração. Experiência.*

**Professor:** Hygor Rasec
https://www.linkedin.com/in/hygorrasec
https://github.com/hygorrasec