



escola  
britânica de  
artes criativas  
& tecnologia

# Desenvolvimento Front-End do Zero ao Pro

Versionamento de código usando o Git

# Conteúdo do módulo

- Stage
- Commits
- Logs
- Histórico
- Branches
- Merge
- Github

# Stage

# Conteúdo da aula

Stage

- Comandos básicos para usar no terminal
- Configuração inicial do Git
- Criando um repositório
- Adicionando e removendo arquivos

# Comandos básicos para usar no terminal

Descrição do comando	Powershell	Prompt de Comando
Listar o conteúdo da pasta atual	ls	dir
Limpar o conteúdo do terminal	CTRL + L ou cls	cls
Navegar entre pastas	cd ./pasta_destino	cd ./pasta_destino
Criar uma nova pasta	mkdir nome_pasta	mkdir nome_pasta
Criar um novo arquivo	echo "conteúdo"> destino.txt	echo "conteúdo" > destino.txt
Remover pasta	rm -r ./nome_pasta	rmdir ./nome_pasta
Remover arquivo	rm -r ./arquivo.txt	del arquivo.txt
Ver o conteúdo de um arquivo	cat ./arquivo.txt	more ./arquivo.txt

# Configuração inicial do Git

No terminal, executar os comandos

```
git config--global user.name "Seu nome"  
git config--global user.email "Seu e-mail"
```

## Criando um repositório Git

No terminal, dentro da pasta do projeto, executar o comando

```
git init .
```

# Adicionando e removendo arquivos

Ao se trabalhar com o Git, não basta incluir ou deletar um conteúdo na pasta do projeto, é necessário adicionar essa mudança ao stage.

Podemos ver os arquivos que estão sendo considerados pelo Git, que estão no stage, através do comando:

```
git status
```

Para adicionar um arquivo ao stage do Git, dizer ao Git que ele deve considerar o arquivo, execute o comando:

```
git add nome_do_arquivo
```

E para remover o arquivo do stage:

```
git rm—cached nome_do_arquivo
```

Para adicionar todos os arquivos da pasta no stage:

```
git add .
```

# Commits



# Conteúdo da aula

Commits

- Salvando alterações
- Padrões para nomear commits
- Changelog

# Salvando alterações

Para salvar as alterações que foram adicionadas ao stage, usamos o comando:

**git commit** –irá abrir um editor de textos, dentro do terminal

```
GNU nano 6.2 C:/git/.git/COMMIT_EDITMSG
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#   deleted:    arquivo.txt
#   deleted:    arquivo2.txt
#

```

[ Read 9 lines ]

<b>^G</b> Help	<b>^O</b> Write Out	<b>^W</b> Where Is	<b>^K</b> Cut	<b>^T</b> Execute	<b>^C</b> Location	<b>M-U</b> Undo	<b>M-A</b> Set Mark
<b>^X</b> Exit	<b>^R</b> Read File	<b>^I</b> Replace	<b>^U</b> Paste	<b>^J</b> Justify	<b>^/_</b> Go To Line	<b>M-E</b> Redo	<b>M-6</b> Copy

# Salvando alterações

Nesse editor vamos descrever a alteração que será salva, exemplo:

remoção de arquivos: `arquivo.txt` e `arquivo2.txt`

Para salvar o texto usamos: **Ctrl+ O**

Para sair do editor: **Ctrl+ X**

**Podemos salvar as alteração, criar um commit, sem ter que acessar esse editor, fazemos isso através do comando:**

`git commit-m "minha mensagem"`

Existe um caminho ainda mais curto, **onde podemos adicionar o arquivo ao stage e já salvar a alteração**, para isso executamos o comando:

`git commit-am "minha mensagem"`

# Padrões para nomear commits

Existe um padrão que podemos seguir, ao escrever as mensagens para os commits.

**Chore:** usado para pequenas tarefas

`gitcommit-m "chore: removendo arquivo.txt"`

**Fix:** correções

`gitcommit-m "fix: correção no cálculo de médias"`

**Feat:** inclusão de funcionalidade

`gitcommit-m "feat: inclusão de função para calcular mediana"`

**Docs:** atualização de documentação

`gitcommit-m "atualizando o changelog.md"`

# Changelog

**Changelog** é um arquivo onde registramos todas as mudanças que aconteceram em um projeto ao decorrer do tempo.

# Logs

# Conteúdo da aula

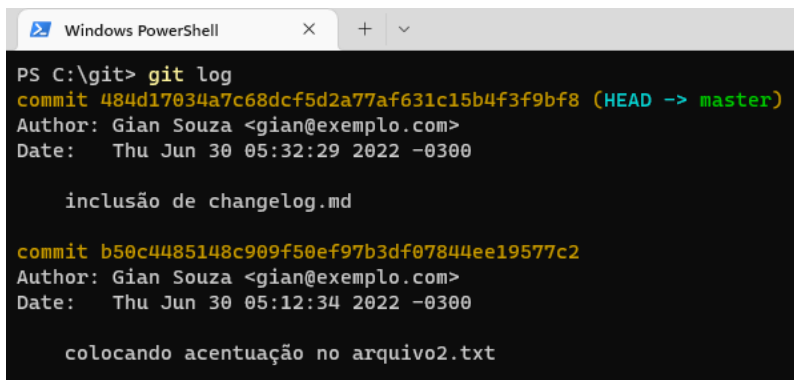
Logs

- Visualizando o registro de commits realizados
- Variações na visualização de commits

# Git log

Para visualizar o histórico de commits, usamos o comando:

`git log`



```
PS C:\git> git log
commit 484d17034a7c68dcf5d2a77af631c15b4f3f9bf8 (HEAD -> master)
Author: Gian Souza <gian@exemplo.com>
Date: Thu Jun 30 05:32:29 2022 -0300

    inclusão de changelog.md

commit b50c4485148c909f50ef97b3df07844ee19577c2
Author: Gian Souza <gian@exemplo.com>
Date: Thu Jun 30 05:12:34 2022 -0300

    colocando acentuação no arquivo2.txt
```



# Git log

O registro de um commits é composto por:

**Commit:** identificador único do commit

**Author:** autor do commit

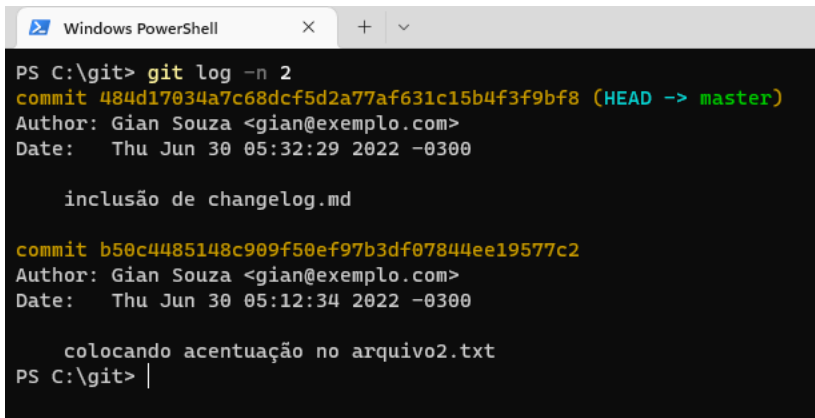
**Date:** data em que o commitfoi feito

Mensagem informado no git commit

# Git log - variações

Para visualizar os últimos dois commits:

`git log -n 2*` (ou outro número)



```
PS C:\git> git log -n 2
commit 484d17034a7c68dcf5d2a77af631c15b4f3f9bf8 (HEAD -> master)
Author: Gian Souza <gian@exemplo.com>
Date: Thu Jun 30 05:32:29 2022 -0300

    inclusão de changelog.md

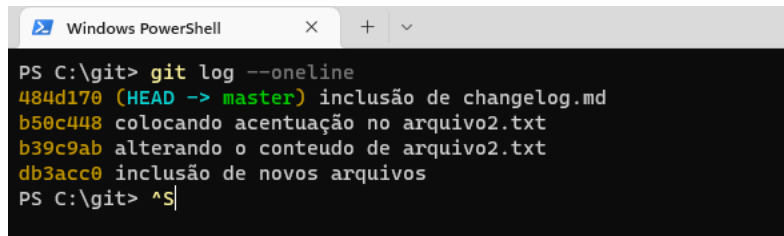
commit b50c4485148c909f50ef97b3df07844ee19577c2
Author: Gian Souza <gian@exemplo.com>
Date: Thu Jun 30 05:12:34 2022 -0300

    colocando acentuação no arquivo2.txt
PS C:\git> |
```

# Git log - variações

Para visualizar uma versão resumida do registro de commits:

`git log --oneline`



```
PS C:\git> git log --oneline
484d170 (HEAD -> master) inclusão de changelog.md
b50c448 colocando acentuação no arquivo2.txt
b39c9ab alterando o conteúdo de arquivo2.txt
db3acc0 inclusão de novos arquivos
PS C:\git> ^S
```

# Git log - variações

Podemos visualizar o registro de uma forma mais detalhada solicitando o conteúdo da alteração, através do comando:

`git log --stat`

```
Windows PowerShell
PS C:\git> git log --stat
commit 484d17834a7c68dcf5d2a77af631c15b4f3f9bf8 (HEAD -> master)
Author: Gian Souza <gian@exemplo.com>
Date: Thu Jun 30 05:32:29 2022 -0300

    inclusão de changelog.md

    changelog.md | 9 ++++++++
    1 file changed, 9 insertions(+)

commit b50c4485148c909f50ef97b3df07844ee19577c2
Author: Gian Souza <gian@exemplo.com>
Date: Thu Jun 30 05:12:34 2022 -0300

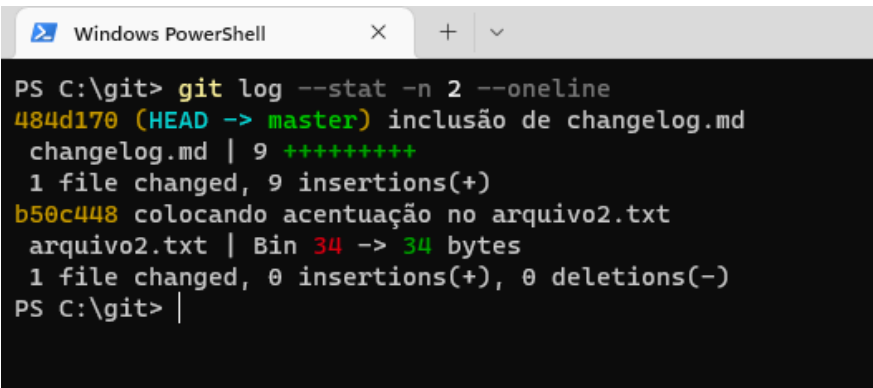
    colocando acentuação no arquivo2.txt

    arquivo2.txt | Bin 34 -> 34 bytes
    1 file changed, 0 insertions(+), 0 deletions(-)
```

# Git log - variações

É possível combinar as variações, solicitamos o conteúdo de dois registros com apenas o identificador do commit:

```
git log --stat -n 2 --oneline
```



```
PS C:\git> git log --stat -n 2 --oneline
484d170 (HEAD -> master) inclusão de changelog.md
 changelog.md | 9 ++++++++
 1 file changed, 9 insertions(+)
b50c448 colocando acentuação no arquivo2.txt
 arquivo2.txt | Bin 34 -> 34 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\git> |
```

# Histórico

# Conteúdo da aula

Histórico

- Desfazendo alterações
- Navegando entre commits
- Revertendo commits

# Desfazendo alterações

Podemos desfazer alterações em arquivos que ainda não foram adicionados ao **stage**, usando o comando:

```
git checkout meu_arquivo
```



## Navegando entre commits

Com o Git podemos viajar no tempo, para o passado, através do comando **git checkout**, podemos voltar o conteúdo de um repositório para o conteúdo de um determinado commit.

```
git checkout COMMIT_ID
```

## Revertendo Commits

O Git nos dá a possibilidade de reverter os commits. **Uma reversão significa que outro commit será criado desfazendo a alteração**, se incluimos um arquivo no commit que queremos reverter, outro commit será criado removendo este arquivo.

Fazemos isso através do comando:

```
git revert HEAD COMMIT_ID
```

# Branches

# Conteúdo da aula

- Visão geral
- Listando as branches
- Criando uma branch
- Convenções para nomear uma branch
- Navegando entre branches
- Deletando uma branch

# Branches – Visão Geral

Com o Git podemos criar versões paralelas do nosso projeto, fazemos isso no Git através da **criação de branches**, que pode ser traduzido como galhos, ramificação, então pensamos em nosso repositório como uma árvore que pode possuir diversos galhos.

Com essa funcionalidade podemos fazer experimentos e trabalhar em equipe de uma forma organizada.

## Listando as branches do repositório

Para visualizar todas as branches que fazem parte do repositório, execute o comando:

**gitbranch**

# Criando uma branch

Para criar uma branch, executamos o comando:

```
git branch nova_branch
```

Com esse comando será criada uma nova ramificação do projeto a partir do estado da branch atual, com uma variação do comando podemos indicar uma branch de referência para a sua criação:

```
git branch nova_branch branch_referencia
```

# Convenções para nomear branches

No dia a dia seguimos um padrão para a nomenclatura de branches, o que nos facilita a identificação de seu conteúdo.

**hotfix/nome\_do\_bug:** usado para correções rápidas, exemplo: o botão de adicionar ao carrinho não funciona, a correção desse bug será feita numa branch hotfix.

**bugfix/nome\_do\_bug:** correção de um bug que não é um impeditivo ao usuário, não compromete um fluxo do site.

**feature/nome\_da\_funcionalidade:** a adição de uma funcionalidade será contida nessa nova branch, até que ela esteja desenvolvida e validada, exemplo:

feature/add\_to\_favorites.

# Navegando entre branches

Para **trocar de branch** usamos o comando:

```
Git checkout nome_branch
```

Podemos **combinar a criação com a troca de branch** com o comando:

```
Git checkout -b nome_branch
```

A branch nova será criada e nosso repositório estará baseado nela.

# Deletando uma branch

Para **deletar uma branch**, executamos o comando:

```
git branch -D nome_branch
```



# Merge

# Conteúdo da aula

Merge

- Mesclando branches
- Conflitos entre branches

# Mesclando branches

Para **mesclar duas branches**, trazendo as atualizações feitas em uma para a outra, executamos o comando:

```
git merge nome_branch
```

# Conflitos entre branches

Apesar do Git ser uma ótima ferramenta para o trabalho em equipe, **ele não pode impedir que duas pessoas alterem o mesmo arquivo.**

Imagine que o arquivo teste.html foi alterado na branch X e na branch Y, no momento em que o merge for realizado o Git criará um conflito, pois ele não sabe qual versão considerar. Esse conflito deverá ser resolvido de forma manual.

# Github

# Conteúdo da aula

- Sobre o Github
- Criando um repositório –projeto já existente
- Criando um repositório –novo projeto
- Enviando atualizações
- Recebendo atualizações

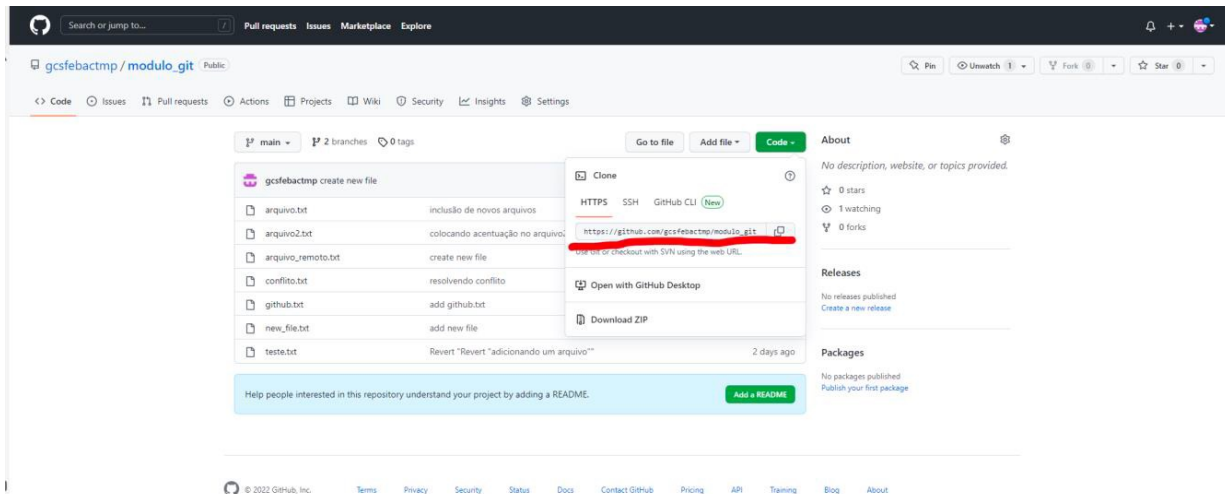
# Github

O **Github** é um site onde **podemos armazenar nossos repositórios Git**.

Além do armazenamento ele proporciona algumas funcionalidades de comunidade, como favoritar seus repositórios, seguir usuários e reportar bugs.

# Clonando um projeto

Para baixarmos um projeto do Github, o termo correto é **clonar**.  
Estando na página do repositório, clique em code e escolha a opção HTTPS, copiando o endereço:





# Clonando um projeto

Com o endereço do repositório copiado, execute o comando:

```
git clone LINK -l pasta_destino *
```

*\* O parâmetro -l é opcional, sem ele será criada uma pasta com o nome do repositório*

## Enviando atualizações

Após fazer o commit de nossas alterações, precisamos enviá-las ao repositório, que está armazenado no Github, fazemos isso através do comando:

```
git push
```

## Recebendo atualizações

Para manter nosso projeto atualizado com o conteúdo remoto, o conteúdo presente no Github, executamos o comando:

```
git pull
```

## Enviando uma determinada branch

Para enviarmos o conteúdo de outra branch, que não é a principal, primeiro mudamos para ela, usando o **git checkout** e depois executamos o **git push** com um parâmetro:

```
git push origin nome_da_branch
```