

FIAP GRADUAÇÃO



TBD

Database Application Development
GIT & GITHUB

PROFA. PATRICIA ANGELINI profpatricia.angelini@fiap.com.br

- ✓ Tipos de versionamento

- ✓ Git

- ✓ Instalando Git

- ✓ Comandos Git

- ✓ Config

- ✓ Init

- ✓ Add

- ✓ Commit

- ✓ Status

- ✓ Restore

- ✓ Reset

- ✓ Branch

- ✓ Checkout

- ✓ Ls-tree

- ✓ Show

- ✓ Head

- ✓ Diff

- ✓ Merge

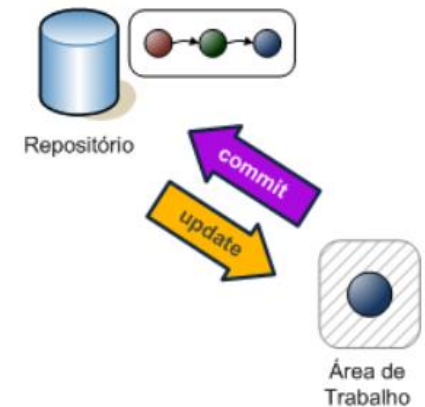
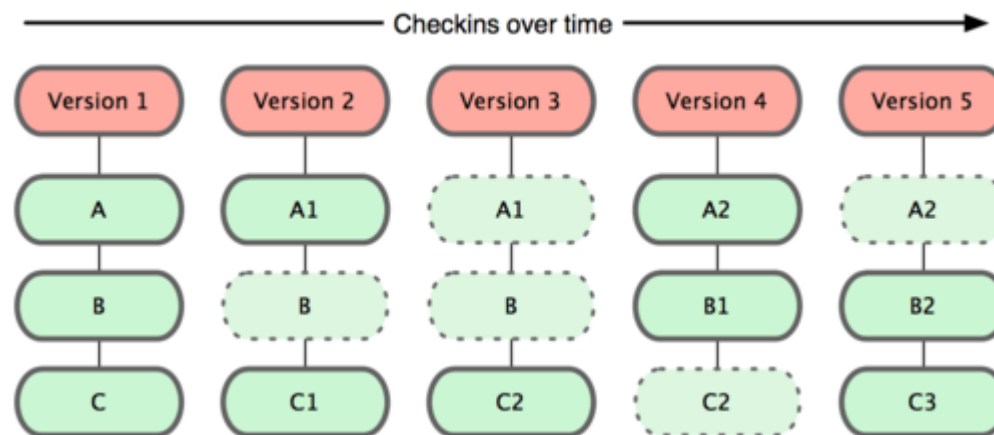
- ✓ Github
- ✓ Vantagens do Github
- ✓ Criando uma conta no Github
- ✓ Funcionalidade Github e Comandos Git
 - ✓ Create Repository
 - ✓ Pull Request
 - ✓ Git clone
 - ✓ Git add
 - ✓ Git commit
 - ✓ Git push
 - ✓ Git pull

SISTEMA DE CONTROLE DE VERSÃO

SISTEMA DE CONTROLE DE VERSÃO

Sistema de controle de versão (VCS)

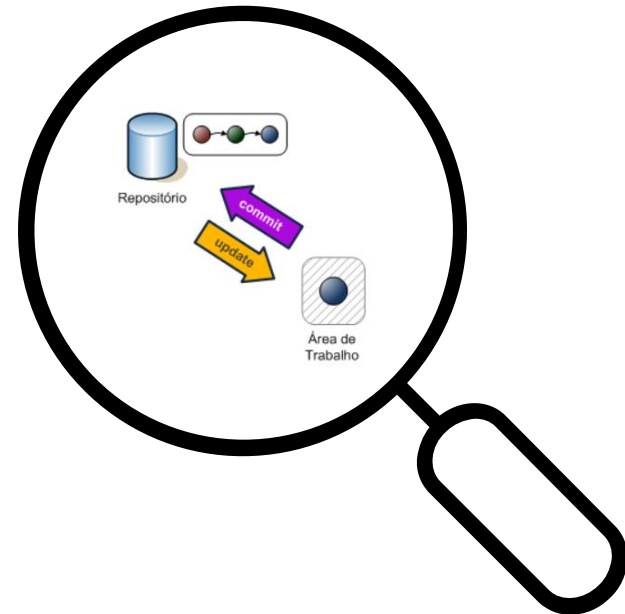
Trabalhando por dias num algoritmo você resolveu fazer algumas alterações e ele deixou de funcionar. Você queria tanto voltar ao algoritmo no ponto onde ele ainda funcionava! Você se arrependeu...



- Os sistemas de controle de versão são softwares que ajudam as pessoas a controlar as alterações no algoritmo (ou qualquer outro documento) ao longo do tempo.
- Alterações = versões
- Em 1972 o Source Code Control System (SCCS) foi criado para controlar a simultaneidade de alterações nos arquivos

SISTEMA DE CONTROLE DE VERSÃO

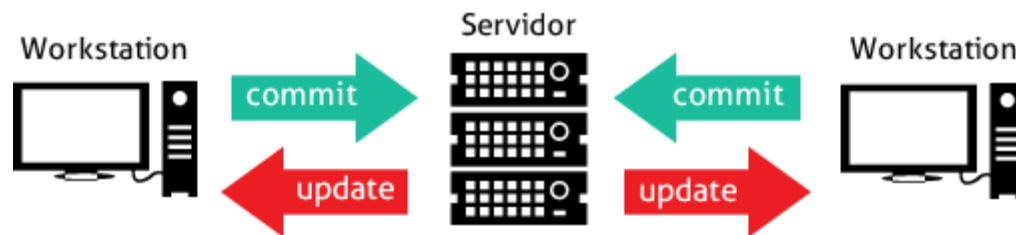
Sistema de Controle de Versão Local



- primeiros a serem criados
- os arquivos de um projeto são copiados para o seu computador, e dentro do próprio computador haviam versões diferentes dos arquivos que estavam sendo trabalhados
- trabalhando sozinho funciona, pois mantém histórico das alterações
- problema: se computador “pifar” perde tudo

SISTEMA DE CONTROLE DE VERSÃO

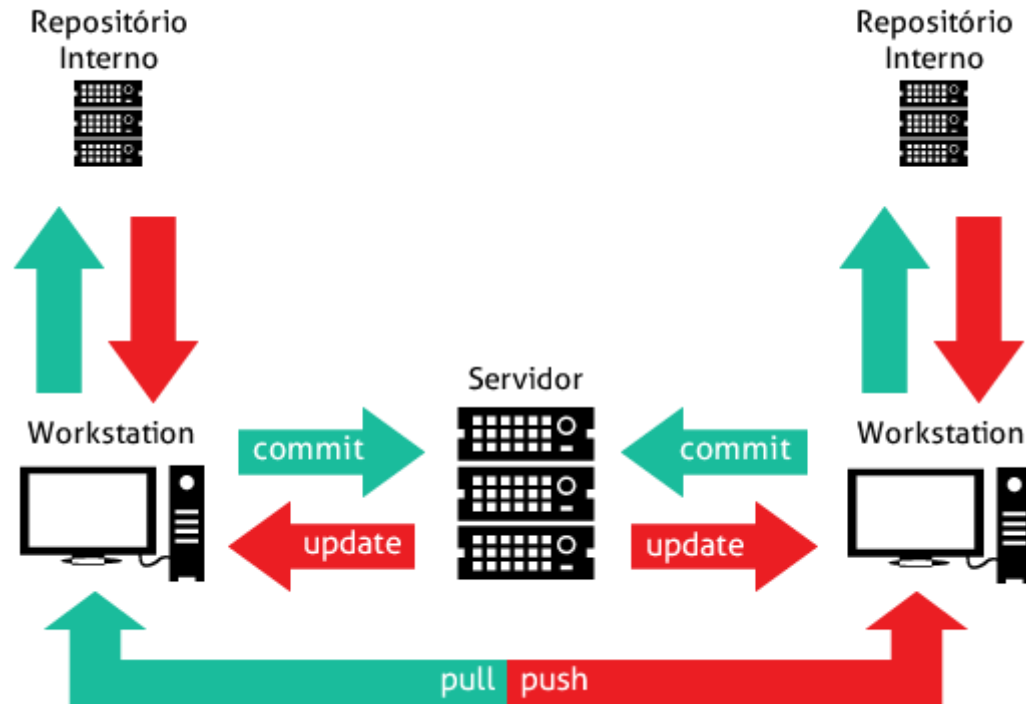
Sistema de Controle de Versão Centralizado



- o controle de versão centralizado foi a evolução do controle local
- trabalhar em equipe
- servidor que armazena tudo que o time todo esta fazendo
- escolhere o arquivo e faz as alterações necessárias diretamente no servidor
- mantem o histórico das versões
- problema: se o servidor estivesse lento ou indisponível? tudo depende do servidor

SISTEMA DE CONTROLE DE VERSÃO

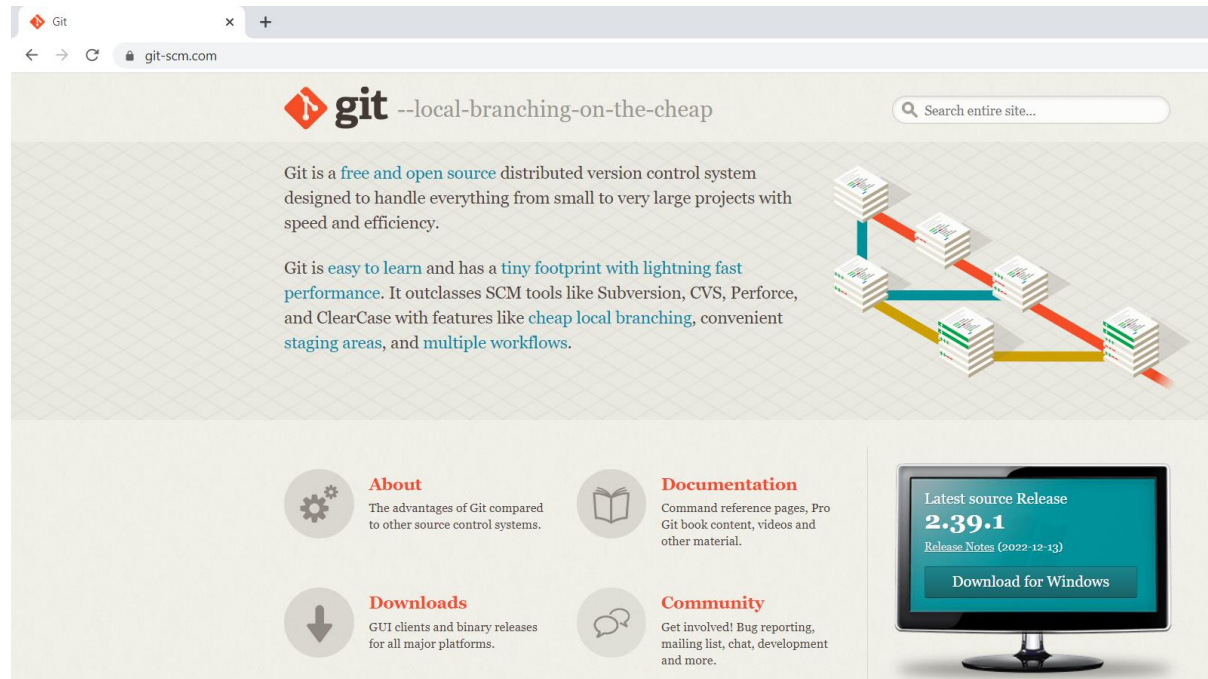
Sistema de Controle de Versão Distribuido



- o melhor dos dois mundos
- time trabalha localmente nas suas versões em cada computador
- repositório central com todos os arquivos fontes
- quando você termina suas alterações localmente daí grava as alterações no servidor

GIT

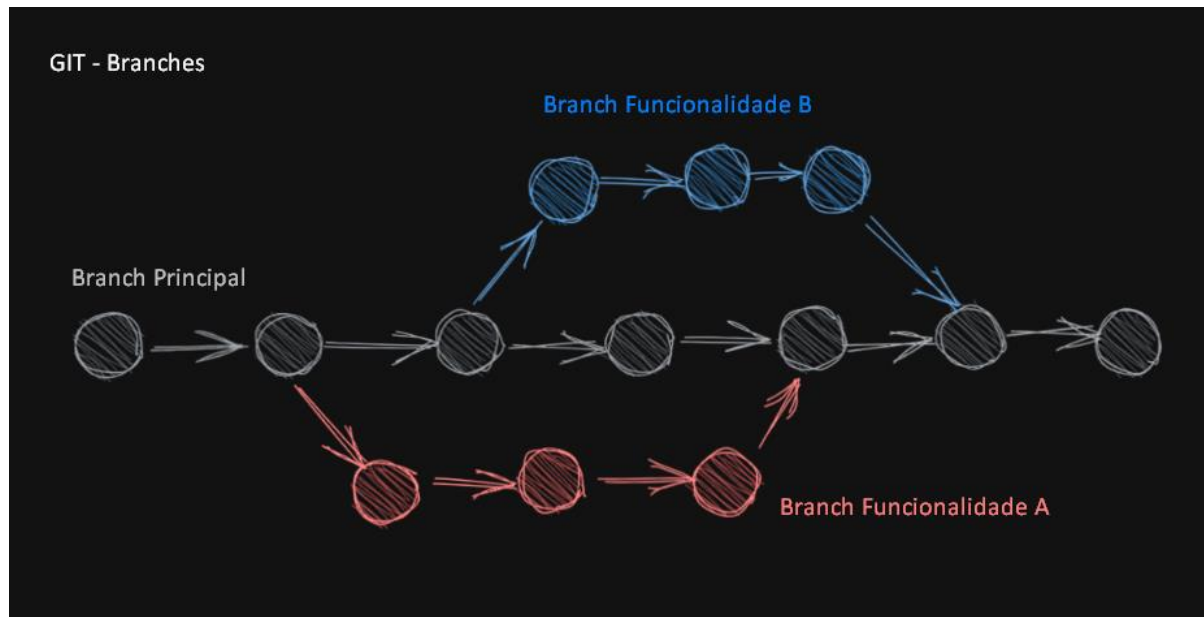
GIT



- O Git é o sistema de controle de versão livre muito usado pelos desenvolvedores
- Linus Torvald criou o Git quando estava desenvolvendo o Linux
- Foi lançado em 2005

GIT

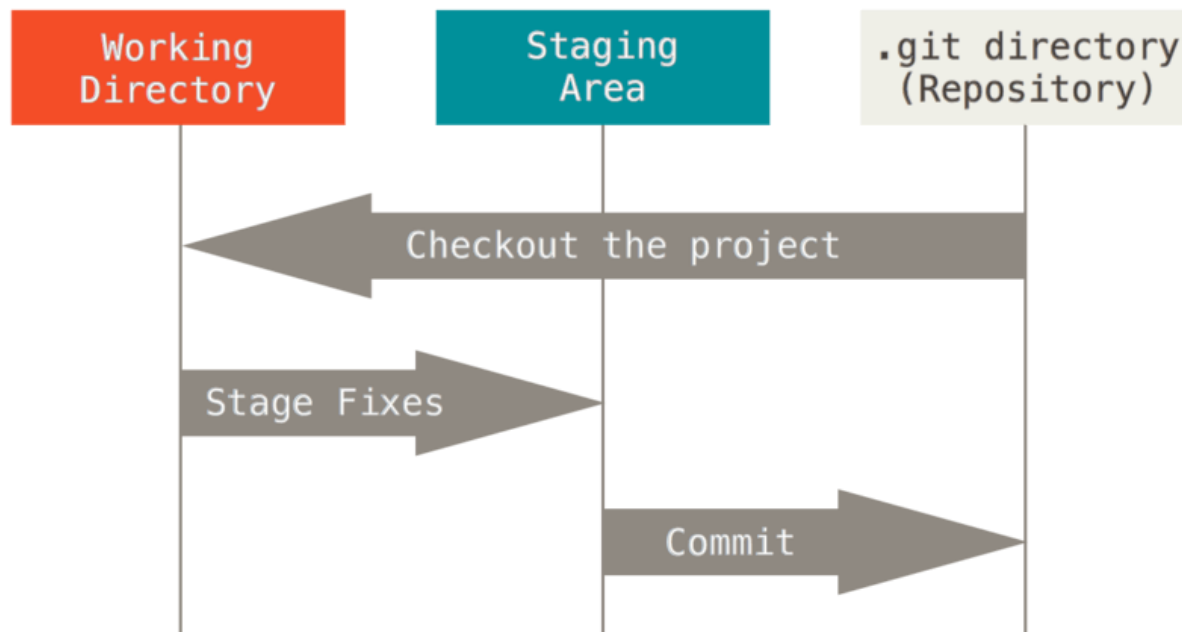
Como funciona?



- Usa um repositório, onde ele armazenará todas as informações dos arquivos: **repo**
- O repo conterá o controle de todas as **branches** (galhos)
- Os arquivos do seu projeto são o tronco da árvore e eles estão na **branch principal**
- Os galhos que vão sendo criados à medida que precisamos alterar algum arquivo, são as branches

GIT

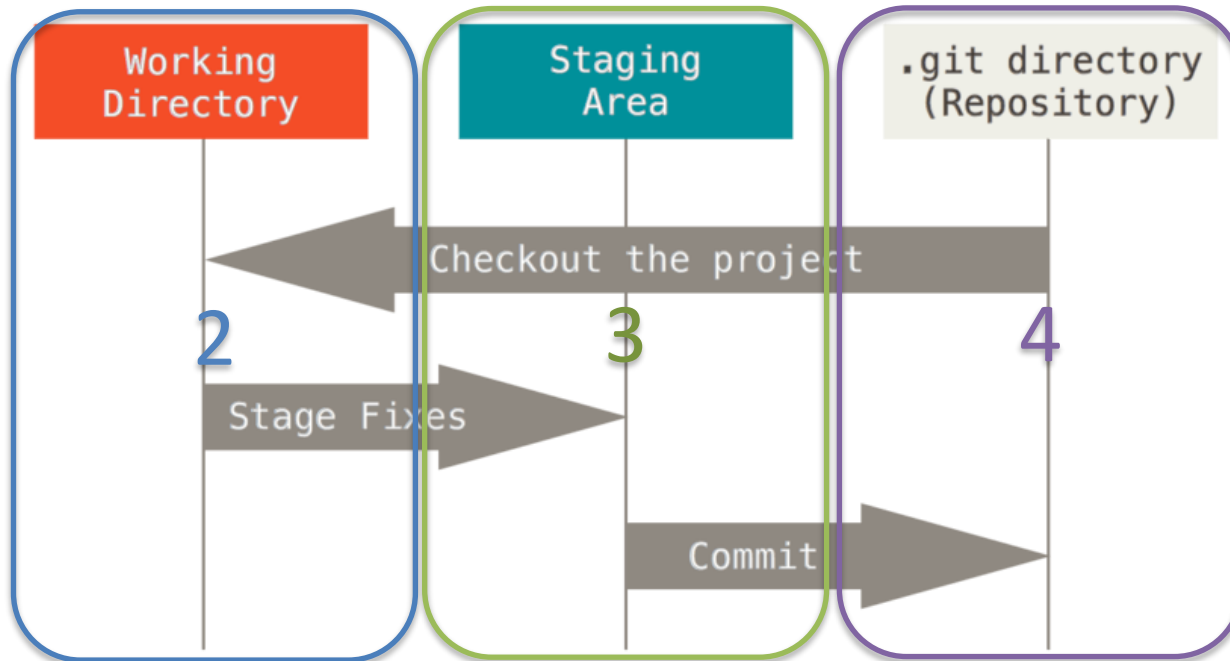
Como fazer o controle de versão?



- Temos 3 áreas para o controle de versão
- A área de trabalho (**Working Área**): usada para fazermos as alterações necessárias nos arquivo – o comando associado a essa área é o **checkout**
- A área de preparo (**Staging Área**): usada para informar que temos alterações ou arquivos novos: comando **add**
- A área de repositório (**Repo**): usada quando confirmamos as alterações: comando **commit**

GIT

Como fazer o controle de versão?



Este é um fluxo de trabalho típico (simplificado).

1. Você faz uma verificação do repositório local para ter uma nova cópia do instantâneo mais recente no local de trabalho/diretório;
2. Você faz algumas alterações em um arquivo em seu diretório de trabalho
3. Você prepara o arquivo, adicionando um instantâneo dele à sua área de teste
4. Você confirma definitivamente o arquivo preparado em seu repositório local.

GIT: Instalando

GIT

Instalando o Git

The image shows two screenshots from the Git website and a Windows File Explorer window.

Top Screenshot (git-scm.com): The 'Downloads' section shows the latest source release as **2.42.0**. Below this, there are links for 'GUI Clients' and 'Logos'.

Bottom Screenshot (git-scm.com): The 'Download for Windows' section provides instructions for downloading the latest (2.42.0) 64-bit version of Git for Windows. It lists three options: 'Standalone Installer', '32-bit Git for Windows Setup', and '64-bit Git for Windows Setup'. The 'Portable ("thumbdrive edition")' section is also visible, with links for '32-bit Git for Windows Portable' and '64-bit Git for Windows Portable'.

File Explorer Window: The window shows the contents of the 'PortableGit' folder. The files listed are:

| Name | Date modified | Type | Size |
|-----------------|-------------------|---------------|--------|
| bin | 03-Sep-23 9:35 PM | File folder | |
| cmd | 03-Sep-23 9:35 PM | File folder | |
| dev | 03-Sep-23 9:35 PM | File folder | |
| etc | 03-Sep-23 9:36 PM | File folder | |
| mingw64 | 03-Sep-23 9:35 PM | File folder | |
| tmp | 30-Aug-23 7:56 AM | File folder | |
| usr | 03-Sep-23 9:35 PM | File folder | |
| git-bash.exe | 30-Aug-23 6:49 AM | Application | 135 KB |
| git-cmd.exe | 30-Aug-23 6:49 AM | Application | 134 KB |
| LICENSE.txt | 30-Aug-23 7:56 AM | Text Document | 19 KB |
| README.portable | 30-Aug-23 7:56 AM | PORTABLE File | 4 KB |

- Em git-scm.com, escolha o tipo de download mais apropriado
- git-bash ou git-cmd são os iniciadores do Git na linha de comando

GIT

Instalando o Git

The image shows two screenshots. The top screenshot is the Git website's 'Downloads' page, which lists download links for macOS, Windows, and Linux/Unix. The bottom screenshot is a Windows file explorer window showing the contents of the 'PortableGit' folder. The files listed are:

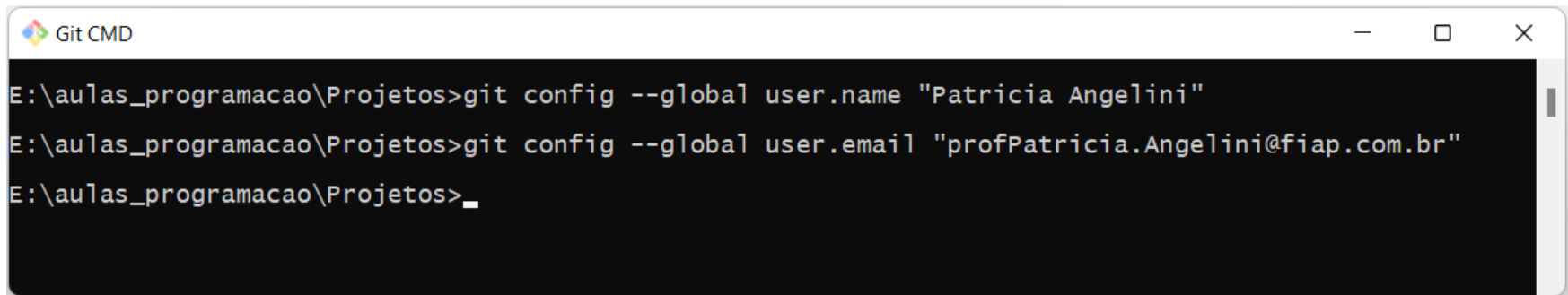
| Name | Date modified | Type | Size |
|-----------------|-------------------|---------------|--------|
| bin | 03-Sep-23 9:35 PM | File folder | |
| cmd | 03-Sep-23 9:35 PM | File folder | |
| dev | 03-Sep-23 9:35 PM | File folder | |
| etc | 03-Sep-23 9:36 PM | File folder | |
| mingw64 | 03-Sep-23 9:35 PM | File folder | |
| tmp | 30-Aug-23 7:56 AM | File folder | |
| usr | 03-Sep-23 9:35 PM | File folder | |
| git-bash.exe | 30-Aug-23 6:49 AM | Application | 135 KB |
| git-cmd.exe | 30-Aug-23 6:49 AM | Application | 134 KB |
| LICENSE.txt | 30-Aug-23 7:56 AM | Text Document | 19 KB |
| README.portable | 30-Aug-23 7:56 AM | PORTABLE File | 4 KB |

- Em git-scm.com, escolha o tipo de download mais apropriado
- git-bash ou git-cmd são os iniciadores do Git na linha de comando

GIT: Primeiros Passos

GIT

Inicializando o GIT



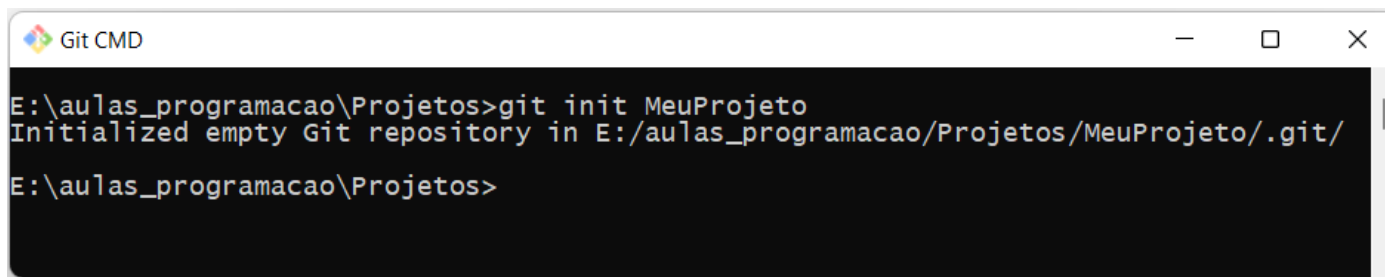
```
Git CMD
E:\aulas_programacao\Projetos>git config --global user.name "Patricia Angelini"
E:\aulas_programacao\Projetos>git config --global user.email "profPatricia.Angelini@fiap.com.br"
E:\aulas_programacao\Projetos>_
```

- O GIT precisa saber que você é pois quando fazemos o registro do nosso projeto no repositório e também quando confirmamos as alterações no repositório, o GIT irá adicionar um metadado para controle das versões de cada arquivo.
- As informações necessárias são nome e email.
- **git config --global user.name "<seu nome>"**
- **git config --global user.email "<seu email>"**

GIT

Criando o Repositorio

.git directory
(Repository)



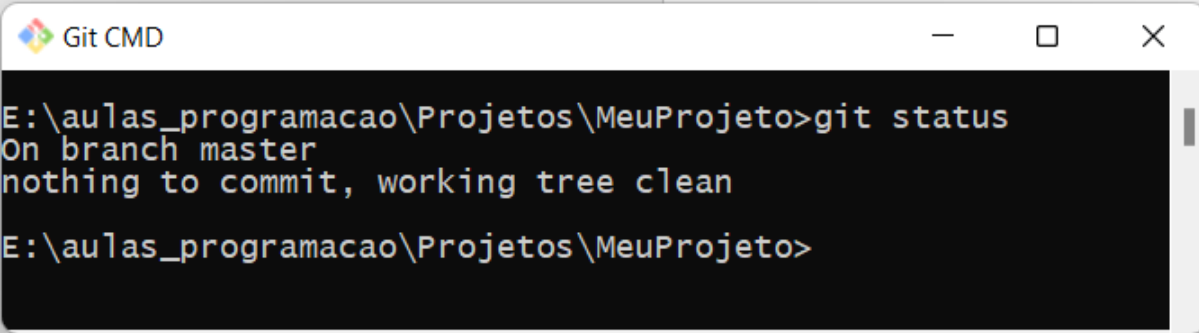
```
Git CMD
E:\aulas_programacao\Projetos>git init MeuProjeto
Initialized empty Git repository in E:/aulas_programacao/Projetos/MeuProjeto/.git/
E:\aulas_programacao\Projetos>
```



- Você já deve ter uma pasta com os arquivos de projeto que você gostaria de criar uma versão. É nessa pasta que vamos criar o repositório do GIT para ele começar a fazer o controle de versão
- Quando você inicia um novo repositório Git em um diretório de projeto, o Git cria uma subpasta oculta chamada ".git" dentro desse diretório. É nessa subpasta que o Git armazena todas as informações sobre o histórico de versões, as configurações do repositório e os objetos do Git (como os commits, árvores e blobs)
- **git init <nome-do-repositorio>**
- Ao executar o comando **git init**, uma branch chamada "**master**" é criada automaticamente como a **branch principal** do repositório

GIT

Verificando o status do repositório

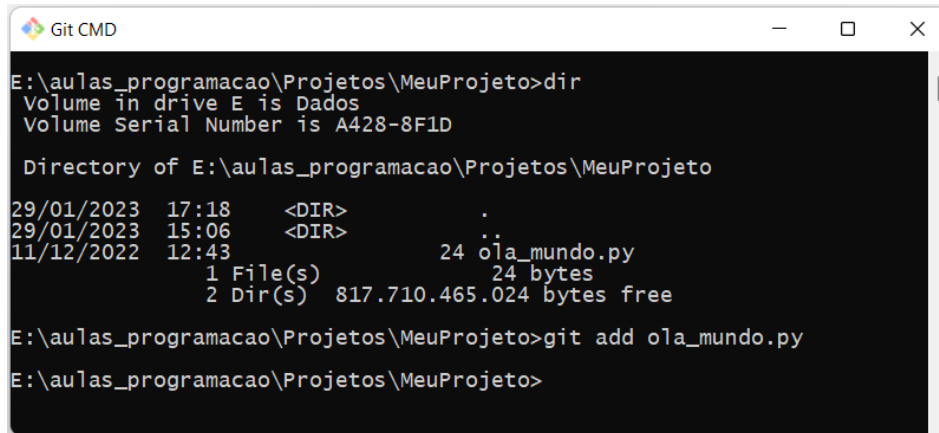


```
Git CMD
E:\aulas_programacao\Projetos\MeuProjeto>git status
On branch master
nothing to commit, working tree clean
E:\aulas_programacao\Projetos\MeuProjeto>
```

- O comando **git status** permite verificar o estado atual de um repositório Git,
- Com o git status, você obtém uma visão instantânea do progresso de seu trabalho
- **git status**

GIT

Adicionando os arquivos do projeto ao Repo



```
Git CMD
E:\aulas_programacao\Projetos\MeuProjeto>dir
Volume in drive E is Dados
Volume Serial Number is A428-8F1D

Directory of E:\aulas_programacao\Projetos\MeuProjeto

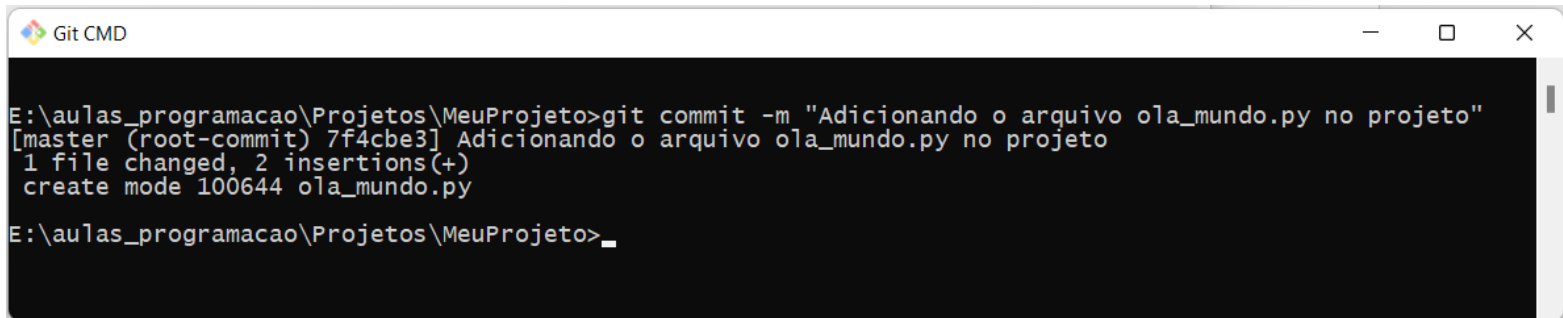
29/01/2023  17:18    <DIR>          .
29/01/2023  15:06    <DIR>          ..
11/12/2022  12:43                24 ola_mundo.py
               1 File(s)                24 bytes
               2 Dir(s)  817.710.465.024 bytes free

E:\aulas_programacao\Projetos\MeuProjeto>git add ola_mundo.py
E:\aulas_programacao\Projetos\MeuProjeto>
```

- Depois do repositório criado é necessário adicionar os arquivos que serão controlados pelo GIT.
- O **git add** é usado para adicionar arquivos e alterações específicas ao índice do Git (também conhecido como "staging area"), que é um espaço intermediário onde você seleciona as alterações que serão incluídas no próximo commit
- Aqui estão algumas maneiras de usar:
 - **git add <arquivo1.ext> <arquivo2.ext>** → adiciona arquivos
 - **git add <diretorio/>** → adiciona um diretório
 - **git add .** → adiciona todas as alterações não rastreadas (novos arquivos e modificações) nos arquivos do diretório atual e seus subdiretórios
 - **git add -u** → adicionar todos os arquivos e pastas que foram exclusivamente modificados e removidos no repositório local

GIT

Efetivando suas alterações no Repo



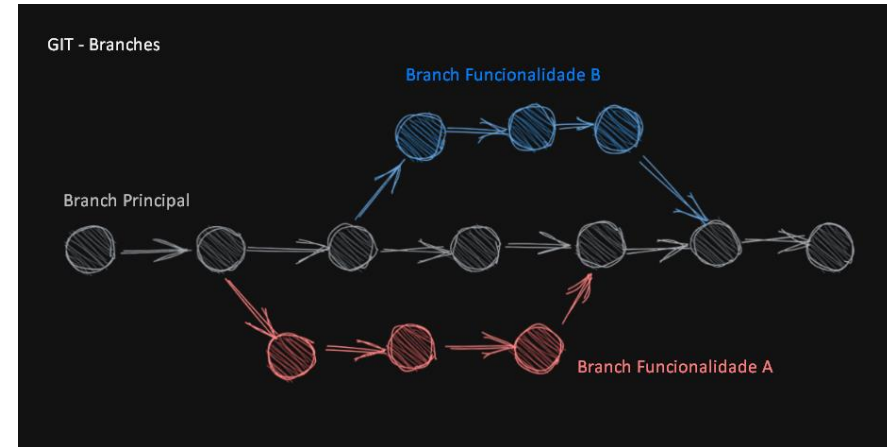
```
Git CMD
E:\aulas_programacao\Projetos\MeuProjeto>git commit -m "Adicionando o arquivo ola_mundo.py no projeto"
[master (root-commit) 7f4cbe3] Adicionando o arquivo ola_mundo.py no projeto
1 file changed, 2 insertions(+)
create mode 100644 ola_mundo.py
E:\aulas_programacao\Projetos\MeuProjeto>
```

- **git commit** é usado para salvar as alterações que você fez em seu repositório local em um ponto específico na história do repositório
- O commit é a forma de guardar o estado do código em um momento específico. Ele é importante para que possamos voltar a esse estado, reverter as alterações, ou compartilhar com outras pessoas
- Aqui estão algumas maneiras de usar:
- **git commit -m "<mensagem-de-commit>"** → -m é uma opção que especifica a mensagem de commit, que é uma descrição curta e significativa das alterações feitas
- **git commit -am "<mensagem-de-commit>"** → -a para automatizar o processo de adicionar todas as alterações atuais no repositório antes do commit
- **git commit -v -m "<mensagem-de-commit>"** → -v que mostra as diferenças de arquivos antes de salvar o commit

GIT: Outras Branchs

GIT

Trabalhando com várias branches

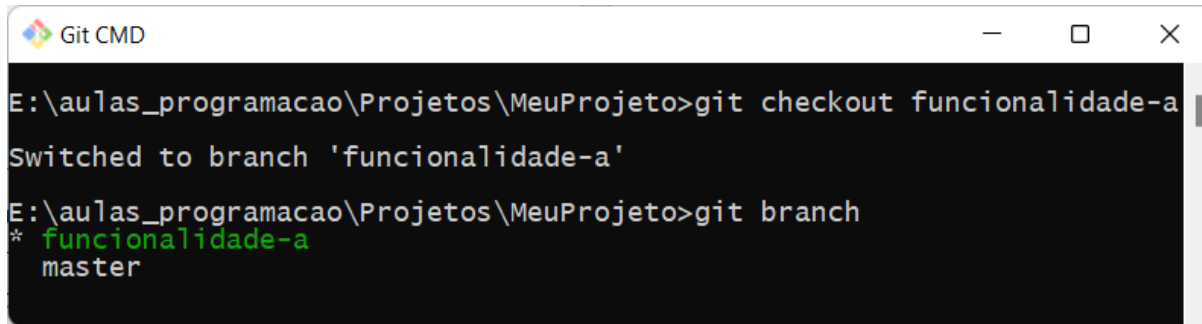


```
Git CMD
E:\aulas_programacao\Projetos\MeuProjeto>git branch funcionalidade-a
E:\aulas_programacao\Projetos\MeuProjeto>git branch
* master
```

- Com o que aprendemos até agora, conseguimos trabalhar num fluxo de versionamento que controla o nosso trabalho individualmente assumindo que estamos sempre trabalhando no ramo principal (que internamente o GIT nomeia de master)
- Podemos trabalhar num ramo do projeto fazendo alguma alteração específica para por exemplo uma funcionalidade A
- **git branch <nome da branch>**

GIT

Alternando entre branches

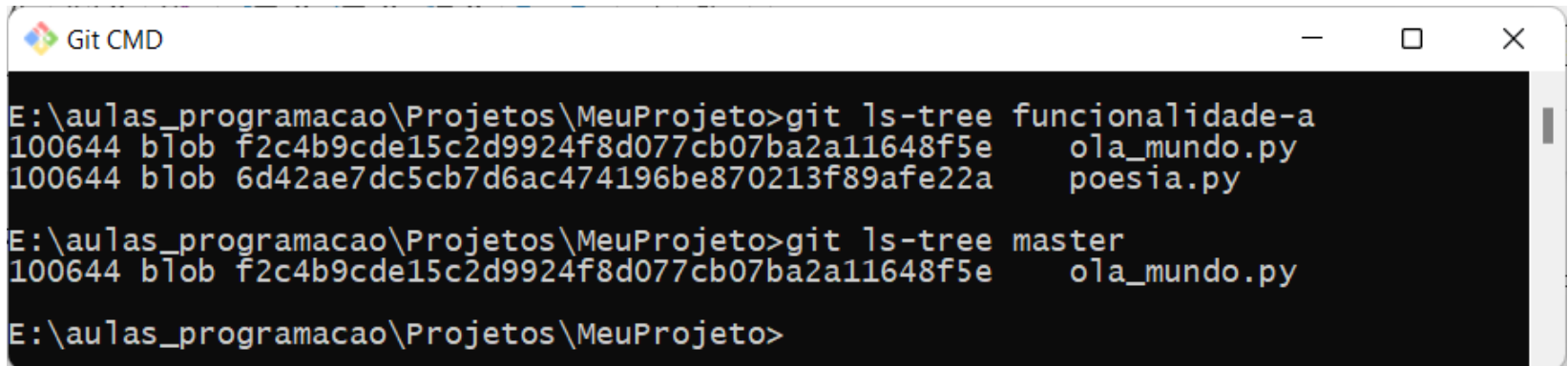
A screenshot of a Windows command prompt window titled "Git CMD". The window shows the execution of two Git commands. The first command is "git checkout funcionalidade-a", which results in the message "Switched to branch 'funcionalidade-a'". The second command is "git branch", which lists the current branches: "funcionalidade-a" (highlighted with an asterisk and green text) and "master".

```
Git CMD
E:\aulas_programacao\Projetos\MeuProjeto>git checkout funcionalidade-a
Switched to branch 'funcionalidade-a'
E:\aulas_programacao\Projetos\MeuProjeto>git branch
* funcionalidade-a
  master
```

- O comando **git checkout** é usado para mudar entre ramos (branches) no Git, ou para descartar alterações locais em arquivos e voltar a uma versão anterior
- Quando você usa git checkout para mudar para outro ramo, ele faz com que sua cópia local do repositório seja atualizada para refletir o conteúdo do ramo selecionado
- **git checkout <nome da branch>**
- **Dica:** você pode fazer a criação e a mudança de ramos fazendo
- **git checkout -b <nome da branch>**

GIT

Verificando as diferenças entre as branches



```
Git CMD
E:\aulas_programacao\Projetos\MeuProjeto>git ls-tree funcionalidade-a
100644 blob f2c4b9cde15c2d9924f8d077cb07ba2a11648f5e    ola_mundo.py
100644 blob 6d42ae7dc5cb7d6ac474196be870213f89afe22a    poesia.py

E:\aulas_programacao\Projetos\MeuProjeto>git ls-tree master
100644 blob f2c4b9cde15c2d9924f8d077cb07ba2a11648f5e    ola_mundo.py

E:\aulas_programacao\Projetos\MeuProjeto>
```

- Uma das maneiras de ver as diferenças entre os ramos é usar o comando **git ls-tree**
- Mostra os arquivos que estão presentes em uma branch específica
- **git ls-tree <nome-do-ramo>**

GIT

Verificando as diferenças entre as branches



```
Git CMD - "C:\Program Files\Git\cmd\git.exe" show funcionalidade-a

E:\aulas_programacao\Projetos\MeuProjeto>git show funcionalidade-a
commit 645f1d1e2245a7c692da14b6647e9bac313b77d4 (HEAD -> funcionalidade-a)
Author: Patricia Angelini <profPatricia.Angelini@fiap.com.br>
Date: Sun Jan 29 22:14:51 2023 -0300

    novo programa poesia.py

diff --git a/poesia.py b/poesia.py
new file mode 100644
index 0000000..6d42ae7
--- /dev/null
+++ b/poesia.py
@@ -0,0 +1,4 @@
+print("De tudo, ao meu amor serei atento")
+print("Antes, e com tal zelo, e sempre, e tanto")
+print("Que mesmo em face do maior encanto")
+print("Dele se encante mais meu pensamento.")
(END)
```

- podemos ver especificidades de um determinado ramo usando o comando **git show**
- mostra o último commit da branch, incluindo o hash, a mensagem de commit e a lista de arquivos modificados
- **git show <nome da branch>**

GIT

Descobrimos os commits de uma branch

```
D:\PortableGit\git-cmd.exe

D:\Projetos\MeuPrimeiroProjeto>git log
commit 29e71bec46acddf94b5fc3b5f0648584a56b9111 (HEAD -> main)
Author: Patricia Angelini <pf1076@fiap.com.br>
Date:   Sun Sep 3 22:46:57 2023 -0300

    poema Mikhael

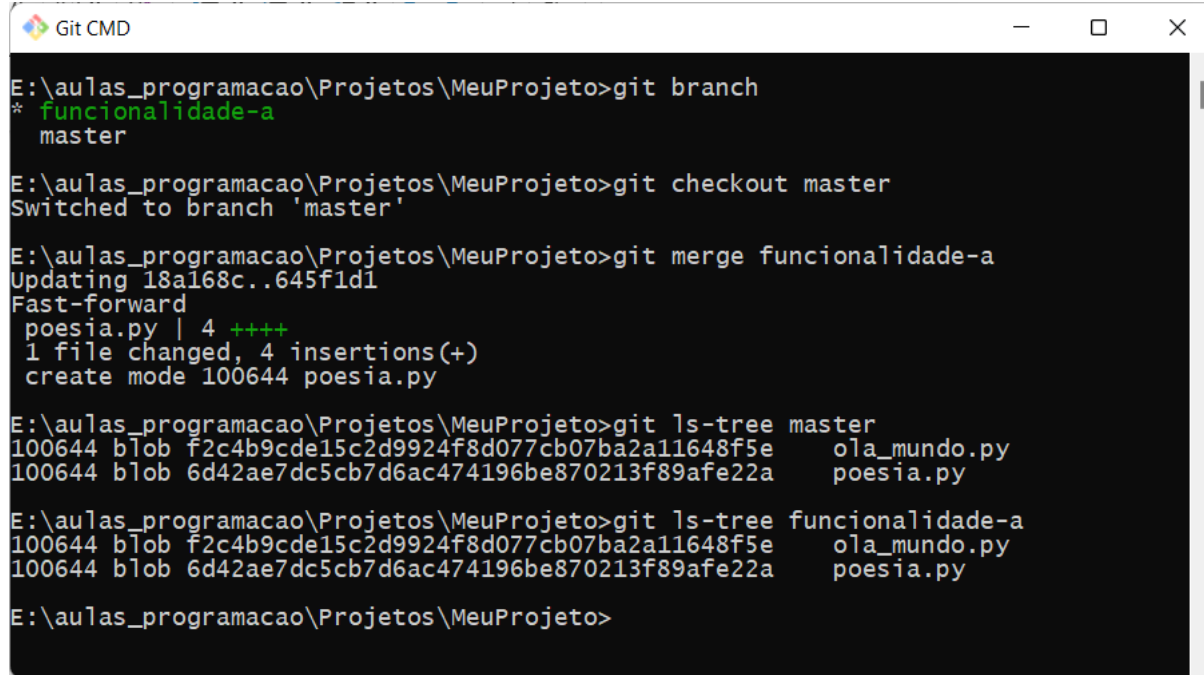
commit 867f091667a7870f86f393a5d9f2e9fcfbef3957
Author: PatriciaAngelini <62622184+PatriciaAngelini@users.noreply.github.com>
Date:   Sun Sep 3 14:45:07 2023 -0300

    Initial commit
```

- Para recuperar os hashes dos commits em uma branch Git específica, você pode usar o comando **git log**. O **git log** exibe o histórico de commits para a branch especificada, incluindo os hashes dos commits
- **git log**
- **git log -n <quantidade>** → indica uma quantidade específica de commits
- **git log --since="2023-01-01" --until="2023-12-31"** → indica um intervalo de commits

GIT

Juntando as branches



```
Git CMD
E:\aulas_programacao\Projetos\MeuProjeto>git branch
* funcionalidade-a
  master

E:\aulas_programacao\Projetos\MeuProjeto>git checkout master
Switched to branch 'master'

E:\aulas_programacao\Projetos\MeuProjeto>git merge funcionalidade-a
Updating 18a168c..645f1d1
Fast-forward
 poesia.py | 4 ++++
 1 file changed, 4 insertions(+)
 create mode 100644 poesia.py

E:\aulas_programacao\Projetos\MeuProjeto>git ls-tree master
100644 blob f2c4b9cde15c2d9924f8d077cb07ba2a11648f5e    ola_mundo.py
100644 blob 6d42ae7dc5cb7d6ac474196be870213f89afe22a    poesia.py

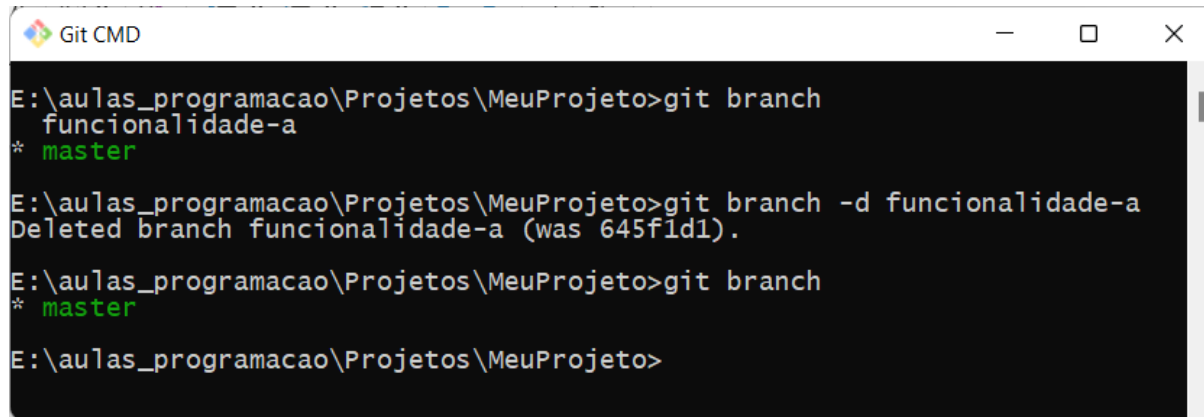
E:\aulas_programacao\Projetos\MeuProjeto>git ls-tree funcionalidade-a
100644 blob f2c4b9cde15c2d9924f8d077cb07ba2a11648f5e    ola_mundo.py
100644 blob 6d42ae7dc5cb7d6ac474196be870213f89afe22a    poesia.py

E:\aulas_programacao\Projetos\MeuProjeto>
```

- Depois que você terminar a funcionalidade e quiser incorporá-la na branch master, você pode fazer um **merge** da branch "funcionalidade-a" na branch "master"
- Para isso você precisa voltar para a branch "master" com o comando **git checkout master**
- Depois usar o comando **git merge funcionalidade-a**
- **git merge <nome-do-ramo>**

GIT

Removendo o ramo já incorporado no ramo principal



```
Git CMD
E:\aulas_programacao\Projetos\MeuProjeto>git branch
funcionalidade-a
* master

E:\aulas_programacao\Projetos\MeuProjeto>git branch -d funcionalidade-a
Deleted branch funcionalidade-a (was 645f1d1).

E:\aulas_programacao\Projetos\MeuProjeto>git branch
* master

E:\aulas_programacao\Projetos\MeuProjeto>
```

- O passo final é garantir que somente o ramo principal tem as últimas alterações
- Para isso vamos excluir o ramo da funcionalidade a
- **git branch -d <nome-do-ramo>**

GIT: Conhecendo um pouco mais

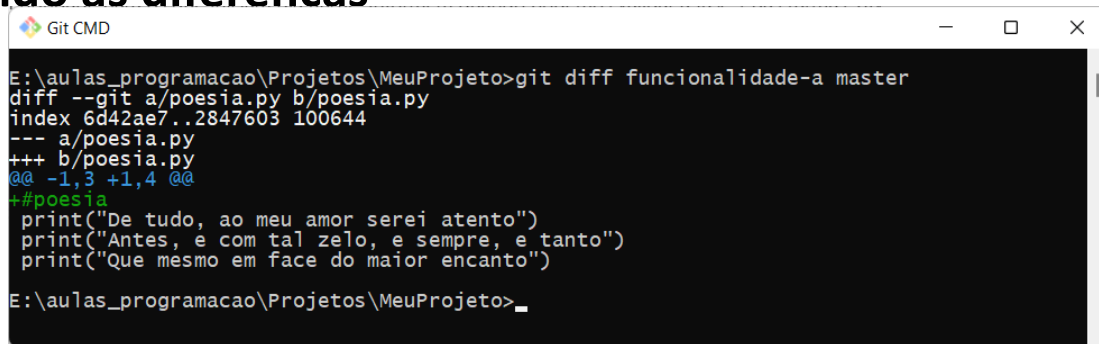
GIT

HEAD

- **HEAD** é um ponteiro no Git que aponta para o último commit realizado na branch atual. É usado frequentemente para se referir ao estado mais recente do repositório
- se você estiver na branch "master", HEAD apontará para o último commit na branch "master". Se você mudar para a branch "FuncionalidadeA", HEAD mudará para apontar para o último commit na branch " FuncionalidadeA

GIT

Examinando as diferenças

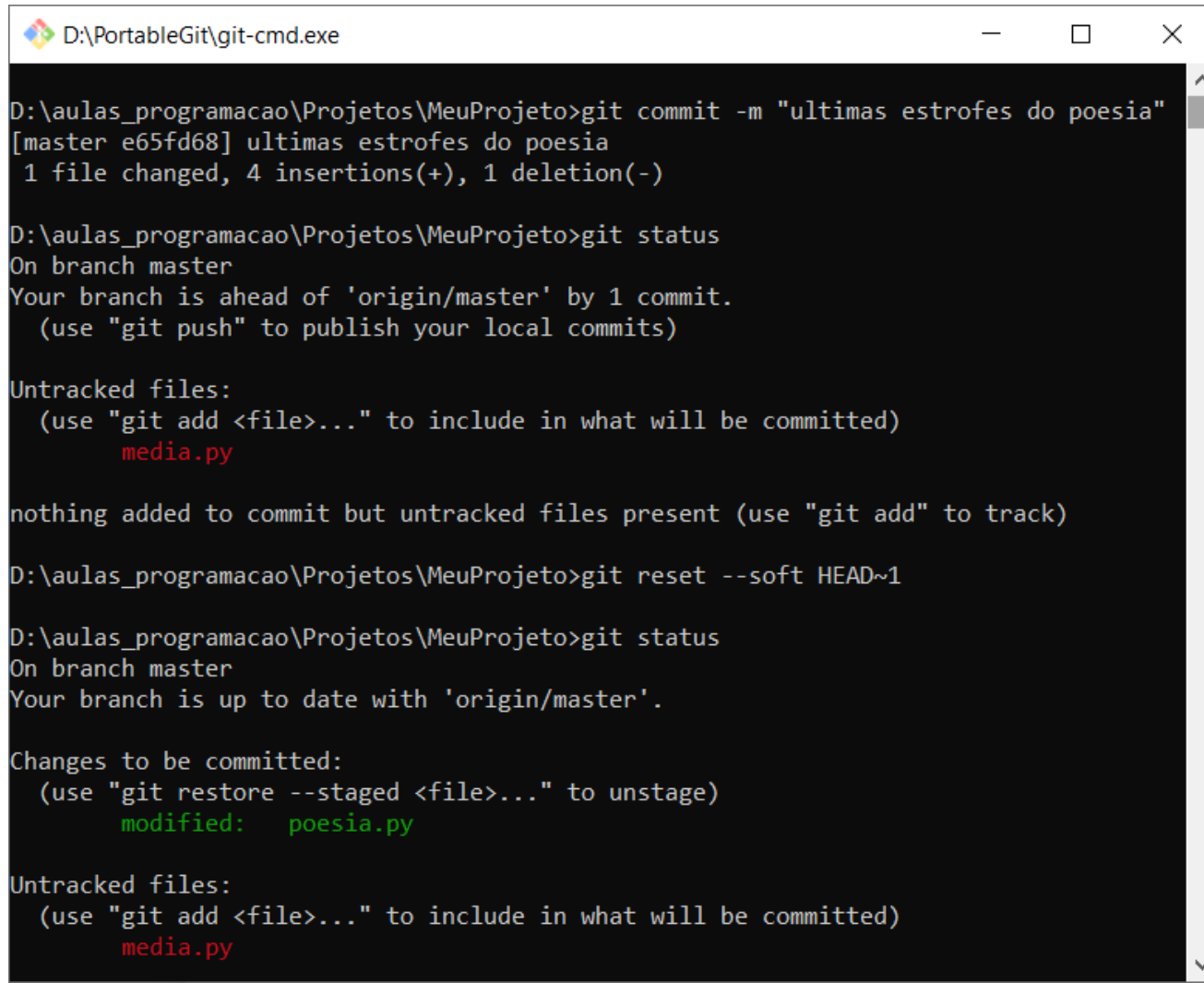


```
E:\aulas_programacao\Projetos\MeuProjeto>git diff funcionalidade-a master
diff --git a/poesia.py b/poesia.py
index 6d42ae7..2847603 100644
--- a/poesia.py
+++ b/poesia.py
@@ -1,3 +1,4 @@
+#poesia
print("De tudo, ao meu amor serei atento")
print("Antes, e com tal zelo, e sempre, e tanto")
print("Que mesmo em face do maior encanto")
E:\aulas_programacao\Projetos\MeuProjeto>
```

- Podemos examinar as diferenças entre versões de arquivos ou até mesmo de branches inteiras
- **git diff [OPTIONS] [<commit>] [--] [<path>...]**
- Alguns usos mais comuns
- **git diff <nome_do_arquivo>** → Compara a versão atual de um arquivo com a última versão “comitada” (efetivada)
- **git diff <branch_A>..<branch_B>** → mostra as diferenças entre as duas branches
- **git diff HEAD** → compara a versão atual com o último commit
- Parametros mais usados
- **--cached**:mostra as diferenças entre a versão atual dos arquivos e a versão indexada (staged). Ex: **git diff --cached <nome_do_arquivo>**

GIT

Voltando um passo



```
D:\PortableGit\git-cmd.exe

D:\aulas_programacao\Projetos\MeuProjeto>git commit -m "ultimas estrofes do poesia"
[master e65fd68] ultimas estrofes do poesia
1 file changed, 4 insertions(+), 1 deletion(-)

D:\aulas_programacao\Projetos\MeuProjeto>git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    media.py

nothing added to commit but untracked files present (use "git add" to track)

D:\aulas_programacao\Projetos\MeuProjeto>git reset --soft HEAD~1

D:\aulas_programacao\Projetos\MeuProjeto>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   poesia.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    media.py
```

GIT

Voltando um passo

- Quando ainda não fizemos a confirmação (commit) das nossas alterações no Repo podemos querer voltar essas alterações
- O comando **git restore** é usado para descartar alterações em arquivos, restaurar arquivos do índice (staging area) para o diretório de trabalho e também pode ser usado para interagir com commits específicos.
- O **git restore** substituirá o conteúdo do arquivo pelo estado do último commit, efetivamente desfazendo as alterações não commitadas no arquivo
- **git restore <nome_do_arquivo>**
- O **git restore** também pode ser usado para restaurar arquivos da área de staging (índice) para o diretório de trabalho
- **git restore --staged <nome_do_arquivo>**

GIT

Voltando um passo

- O comando **git reset** permite reverter alterações em um repositório Git, especialmente no que diz respeito à movimentação do ponteiro HEAD, que aponta para a commit atual, e ao índice (staging area)
- O **git reset -- soft** move o ponteiro **HEAD** e a ramificação atual para um commit anterior, mas mantém as mudanças dos commits revertidos no índice (staging area). Isso permite que você faça alterações adicionais antes de criar um novo commit com as mudanças desfeitas
- **git reset --soft <HEAD~1>**
- O **git reset -- mixed** (é o padrão) move o ponteiro **HEAD** e a ramificação atual para um commit anterior e também desfaz as mudanças do índice (staging area), mas mantém as alterações nos arquivos de trabalho. Isso permite que você reavalie e escolha quais alterações incluir em seu próximo commit.
- **git reset <HEAD~1>**
- O **git reset -- hard** move o ponteiro HEAD e a ramificação atual para um commit anterior, desfaz as mudanças do índice e também descarta todas as alterações nos arquivos de trabalho. MUITO CUIDADO AO USAR
- **git reset --hard <HEAD~1>**

GIT

Git Reset X Git RM X Git Revert

O git reset, o git rm e o git revert são comandos Git usados para diferentes finalidades.

git reset:

Finalidade: O comando git reset é usado para mover a branch e o índice para um commit específico.

Exemplo:

Digamos que você tenha o seguinte histórico de commits:

A - B - C - D - E (branch master)

Se você executar **git reset --hard B**, o histórico se tornará:

A - B (branch master)

O commit C, D e E são descartados e todos os arquivos no diretório de trabalho são revertidos para o estado de commit B

GIT

Git Reset X Git RM X Git Revert

O git reset, o git rm e o git revert são comandos Git usados para diferentes finalidades.

git rm:

Finalidade: O comando git rm é usado para remover arquivos do repositório Git.

Exemplo:

Suponha que você deseja remover um arquivo chamado file.txt. Você pode fazer o seguinte:

git rm file.txt

Após confirmar e fazer um novo commit, o arquivo file.txt será removido do repositório.

GIT

Git Reset X Git RM X Git Revert

O git reset, o git rm e o git revert são comandos Git usados para diferentes finalidades.

git revert:

Finalidade: O comando git revert é usado para criar um novo commit que desfaz as alterações introduzidas por um commit específico.

Exemplo:

Suponha que você tenha o seguinte histórico de commits:

A - B - C - D (branch master)

E você deseja desfazer as alterações introduzidas pelo commit C. Você pode fazer o seguinte:

git revert C

Isso criará um novo commit que desfaz as alterações introduzidas pelo commit C, mantendo o histórico em andamento:

A - B - C - D - E (revert de C) (branch master)

GIT

Git Reset X Git RM X Git Revert

Em resumo:

git reset: move a branch e o índice para um commit específico, geralmente usado para reverter o histórico.

git rm: remove arquivos do repositório.

git revert: cria um novo commit que desfaz as alterações introduzidas por um commit anterior, mantendo um histórico linear.

Git e Github

Git e Github

- O Git e o GitHub são duas ferramentas relacionadas, mas desempenham funções diferentes no desenvolvimento de software.
- O Git é a tecnologia subjacente que permite o controle de versão, enquanto o GitHub é uma plataforma que utiliza o Git para facilitar a colaboração, o gerenciamento de projetos e o armazenamento na nuvem.
- São frequentemente usados juntos para desenvolvimento de software colaborativo e controle de versão

Git e Github

| Característica | Git | GitHub |
|---------------------------|---|--|
| Natureza | Sistema de controle de versão distribuído | Plataforma de hospedagem de código |
| Armazenamento | Local no sistema de arquivos do computador | Remoto em servidores na nuvem |
| Colaboração | Não inclui recursos de colaboração online | Fornece recursos de colaboração online |
| Visibilidade | Repositórios locais podem ser privados ou públicos, definidos manualmente | Repositórios podem ser públicos ou privados |
| Backup e Redundância | Responsabilidade do usuário fazer backup | GitHub cuida do backup e redundância |
| Gerenciamento de Projetos | Não inclui recursos de gerenciamento de projetos | Fornece ferramentas de gerenciamento de projetos |
| CI/CD | Não inclui recursos de CI/CD (Integração Contínua/Implantação Contínua) | Oferece GitHub Actions para automação CI/CD |

Github



Github

Visão Geral

GitHub é uma plataforma de gerenciamento baseada na nuvem que oferece recursos de controle de versão do Git.

- **Repositórios:** Espaços para armazenar e gerenciar seus arquivos e códigos-fonte. Cada projeto tem seu próprio repositório podendo ser públicos ou privados.
- **Git:** Sistema de controle de versão subjacente, usado internamente pelo Github, que rastreia alterações e facilita a colaboração.
- **Clonagem:** Para começar a trabalhar em um projeto do GitHub, você normalmente clona (faz uma cópia local) do repositório em seu computador. Isso permite que você faça alterações no código e teste as modificações em seu ambiente de desenvolvimento local.
- **Ramificações (Branches):** Você pode criar ramificações do repositório principal para desenvolver novos recursos ou corrigir bugs sem afetar o código principal.



GitHub

Visão Geral

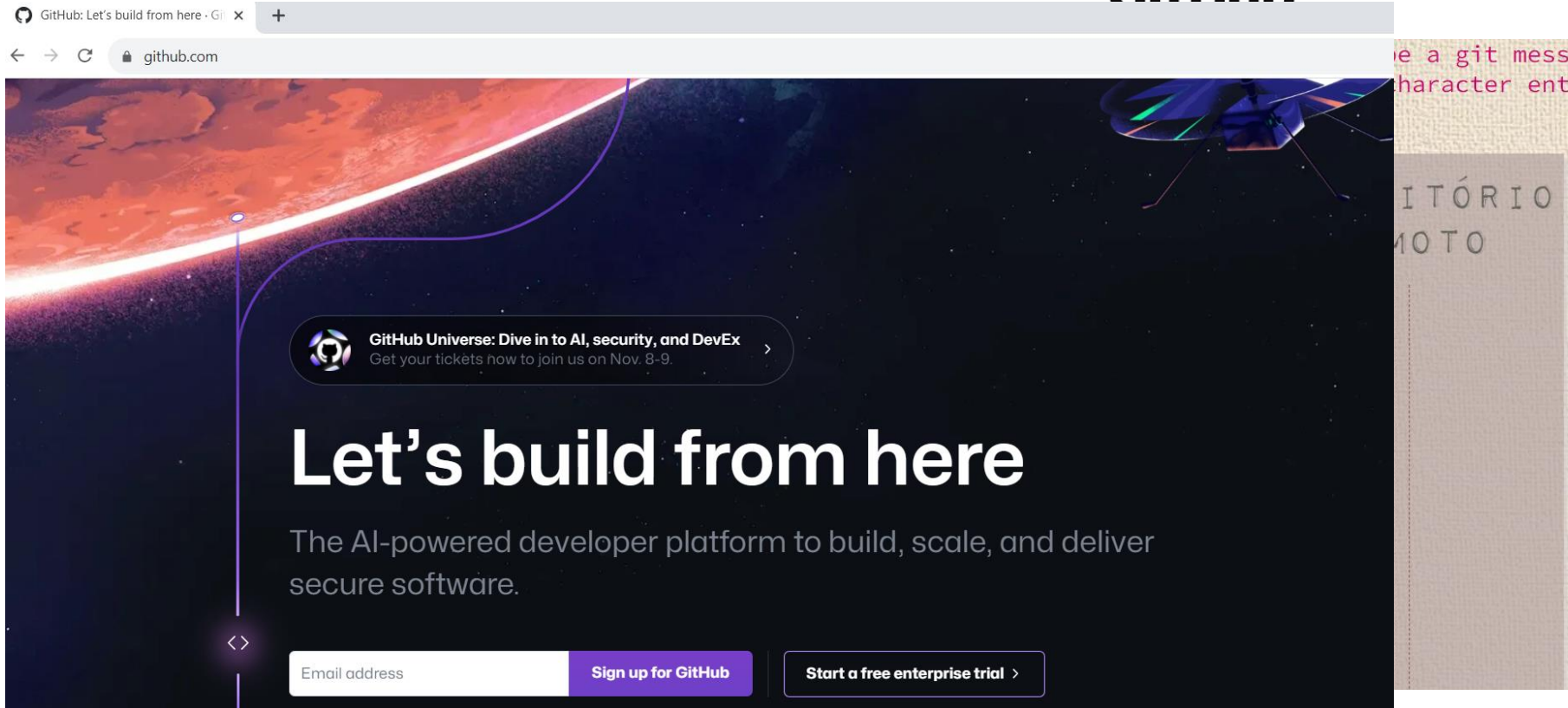
GitHub é uma plataforma de gerenciamento baseada na nuvem que oferece recursos de controle de versão do Git.

- **Pull Requests:** Quando o código que você estiver trabalhando estiver pronto, você cria uma solicitação de pull, ou seja solicitações para incorporar alterações ao código principal, permitindo revisões e feedback.
- **Colaboração:** Vários desenvolvedores podem colaborar em um projeto, trabalhando em suas próprias ramificações e enviando solicitações de pull para revisão.
- **Integração Contínua (CI) e Implantação Contínua (CD):** Muitos projetos no GitHub usam ferramentas de CI/CD para automatizar a construção, teste e implantação de código.
- **Problemas e Gerenciamento de Projetos:** Acompanhamento de problemas e gerenciamento de tarefas.
- **Wiki e Documentação:** Hospedagem de wikis e documentação relacionada ao projeto.
- **Comunidade:** Plataforma social para seguir projetos, contribuir para código aberto e interagir com outros desenvolvedores.

Github: Criando uma conta



Github – criando uma conta



- Crie uma Conta no GitHub:

Se você ainda não tiver uma conta no GitHub, vá para o site [GitHub.com](https://github.com) e clique em "Sign up" para criar uma conta gratuita. Siga as instruções para configurar seu perfil.



Github – criando uma conta

Welcome to GitHub!
Let's begin the adventure

Enter your email*

✓ pf@fiap.com.br

Create a password*

✓

Enter a username*

→ profFiap

Continue

- Preencha seus dados

Github: Primeiros Passos



Github – Repo no Github

The screenshot shows the GitHub web interface. In the top left, the 'New' button (a green square with a white plus icon) is highlighted with a red box. In the top right, the user profile dropdown menu is open, and the 'New repository' option is highlighted with a red box. The main content area shows a welcome message and a trending repository card for 'OpenBMB / ChatDev'.

Crie um Repositório no GitHub:

- No canto superior direito, clique no sinal de "+" e selecione "New repository".
- Ou você pode escolher o botão verde "New"




Github – Repo no Github

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * / Repository name *



 PatriciaAngelini / MeuPrimeiroProjeto

✓ MeuPrimeiroProjeto is available.

Great repository names are short and memorable. Need inspiration? How about [probable-enigma](#) ?

Description (optional)

Meu primeiro projeto

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

- ☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

- Preencha o nome do repositório, descrição e escolha se deseja torná-lo público ou privado.
- Você pode opcionalmente marcar “Add a README file” para acrescentar um arquivo inicial ao projeto
- Depois, clique em "Create repository".



Github – Repo no Github






PatriciaAngelini/MeuPrimeiroProj x +








github.com/PatriciaAngelini/MeuPrimeiroProjeto


PatriciaAngelini / MeuPrimeiroProjeto


Q Type [7] to search

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

 **MeuPrimeiroProjeto** Public  Pin  Unwatch 1  Fork 0  Star 0

 main  1 branch  0 tags  Go to file  Add file  <> Code  About

 PatriciaAngelini Initial commit 867f091 2 minutes ago 1 commit

 README.md Initial commit 2 minutes ago






README.md

MeuPrimeiroProjeto

Meu primeiro projeto

About

Meu primeiro projeto


-  Readme
-  Activity
-  0 stars
-  1 watching
-  0 forks

Releases

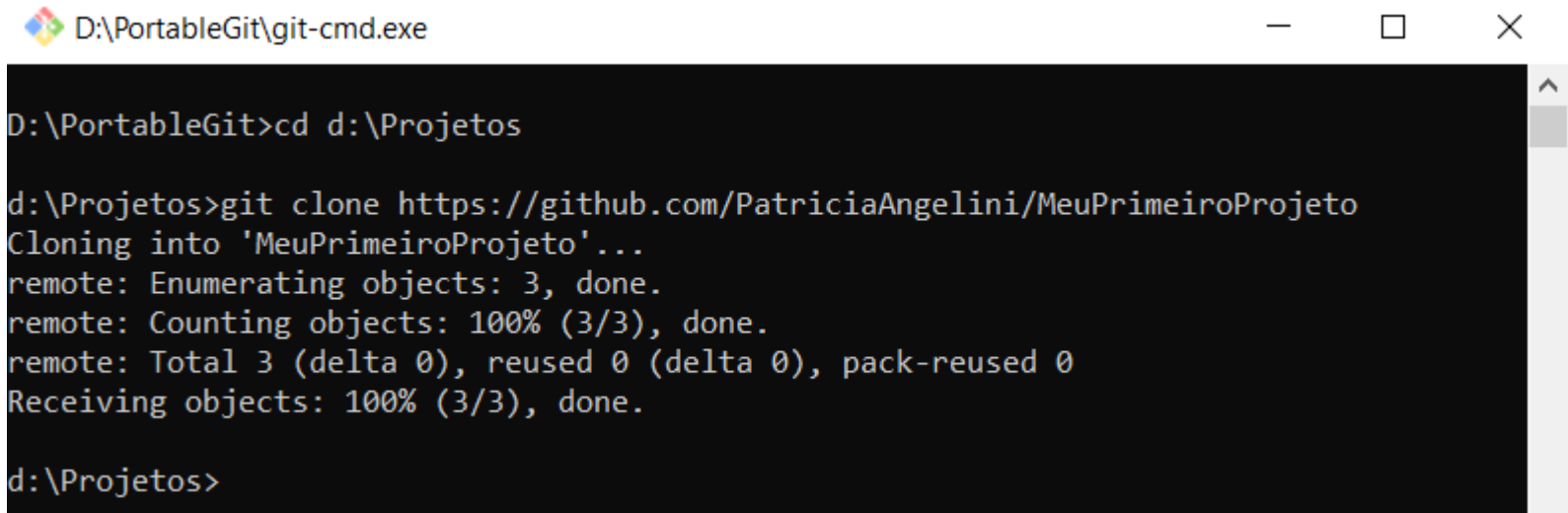
No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

 © 2023 GitHub, Inc. [Terms](#) [Privacy](#) [Security](#) [Status](#) [Docs](#) [Contact GitHub](#) [Pricing](#) [API](#) [Training](#) [Blog](#) [About](#)

Github – Clone Local




```
D:\PortableGit>cd d:\Projetos

d:\Projetos>git clone https://github.com/PatriciaAngelini/MeuPrimeiroProjeto
Cloning into 'MeuPrimeiroProjeto'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

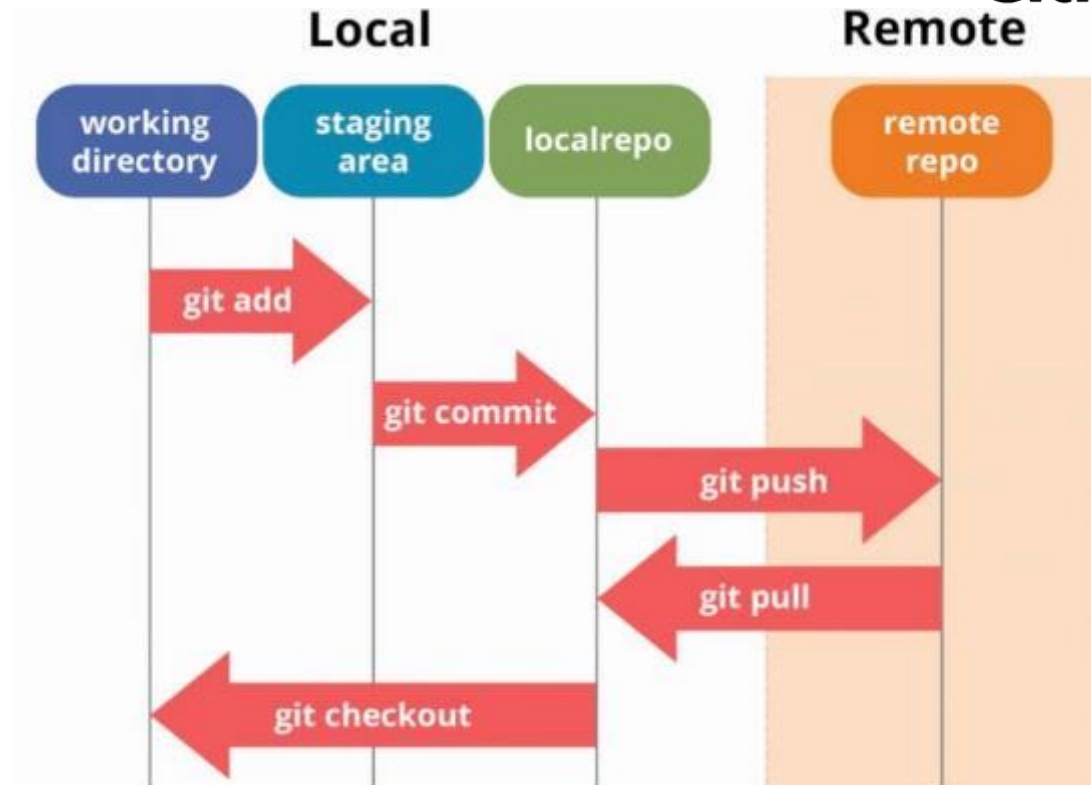
d:\Projetos>
```

Clone o Repositório para o seu Computador:

- No repositório recém-criado, clique no botão verde "Code" e copie a URL do repositório.
- Abra o terminal ou prompt de comando no seu computador (**git-cmd.exe**). 
- Navegue até a pasta onde deseja clonar o repositório.
- Use o comando **git clone** [[https://github.com/\[seu-nome-de-usuario\]/\[nome-do-projeto\].git](https://github.com/[seu-nome-de-usuario]/[nome-do-projeto].git)] para criar uma cópia local do repositório no seu computador
- Agora que você tem uma cópia do repositório no seu computador, você pode adicionar, modificar ou excluir arquivos de acordo com as necessidades do seu projeto.

Github: Trabalhando com o Clone Local e com o Repositorio do Github

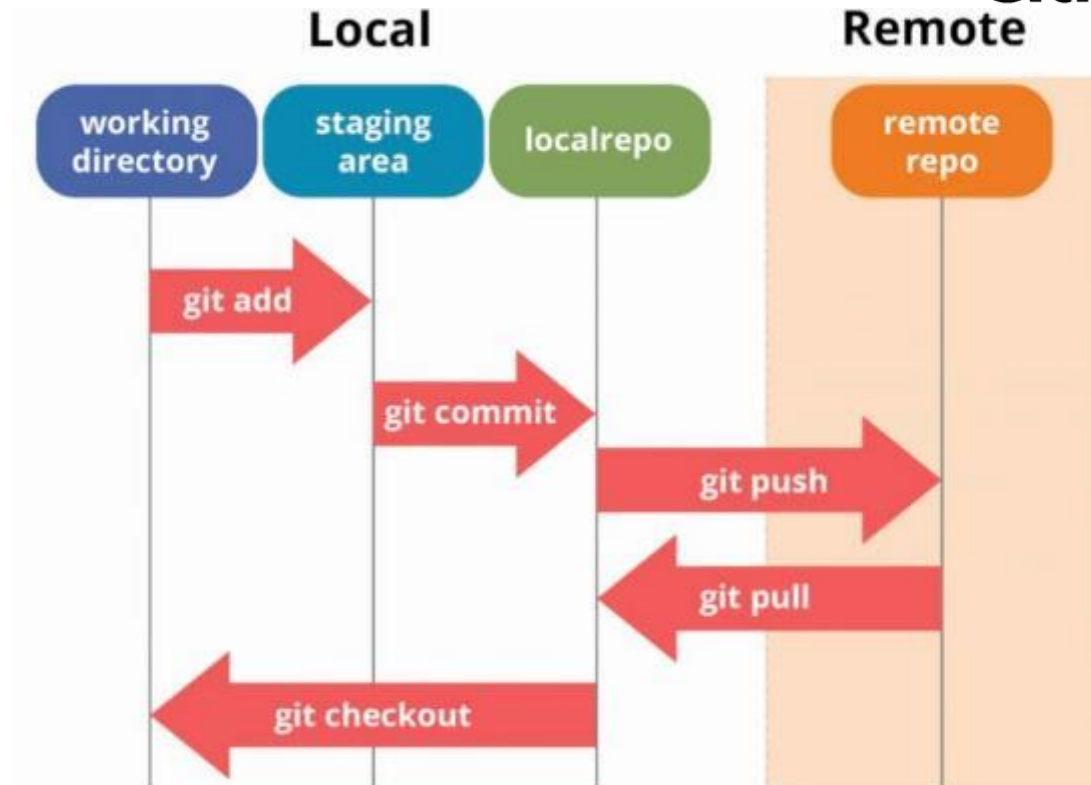
Github – Trabalhando com o Clone



Adds e Commits:

- Após fazer alterações nos arquivos, use o comando **git add [arquivos]** para preparar as mudanças para o commit.
- Em seguida, use o comando **git commit -m "[mensagem de commit]"** para criar um commit com uma mensagem descritiva das alterações realizadas.

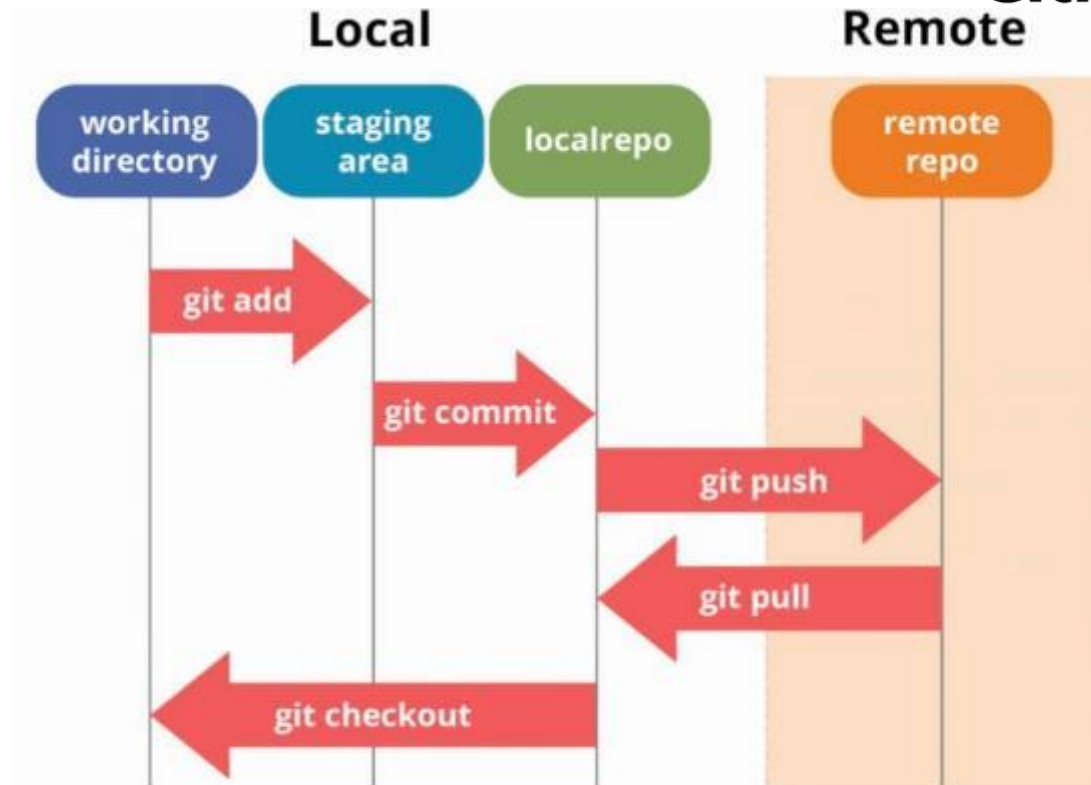
Github – Trabalhando com o Clone



Push:

- Use o comando **git push origin [nome-do-ramo]** para enviar seus commits para o repositório remoto no GitHub. O padrão é geralmente enviar para o ramo principal, que é chamado de "**main**" ou "**master**".

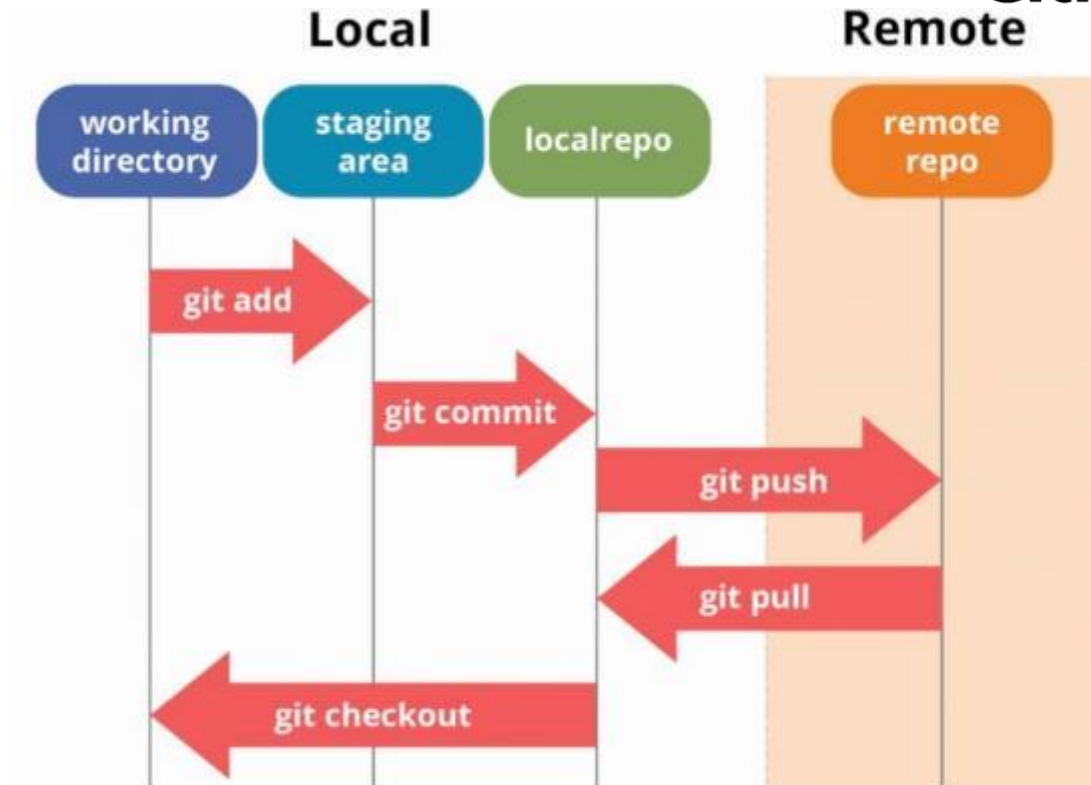
Github – Trabalhando com o Clone



Pull:

- Abra um terminal ou prompt de comando.
- Navegue até o diretório do seu repositório Git local usando o comando **cd**.
- Certifique-se de que seu repositório local está atualizado com o repositório remoto usando o comando **git pull origin [nome-do-ramo]**

Github – Trabalhando com o Clone



Checkout:

- Se você deseja trabalhar em uma nova funcionalidade ou correção, é uma boa prática criar uma nova ramificação usando o comando **git checkout -b [nome-da-ramificacao]**. Isso mantém seu trabalho isolado do ramo principal.

Github: Pull Request

Github – Pull Request



MeuPrimeiroProjeto Public

Pin Unwatch 1 Fork 0 Star 0

novapoesia had recent pushes 29 minutes ago [Compare & pull request](#)

main 2 branches 0 tags [Go to file](#) [Add file](#) [Code](#)

Your main branch isn't protected
Protect this branch from force pushing or deletion, or require status checks before merging. [Learn more](#) [Protect this branch](#)

| | | |
|------------------|----------------|--------------------------------|
| PatriciaAngelini | poema Mikhael | 29e71be 11 hours ago 2 commits |
| PoemaMikhael.txt | poema Mikhael | 11 hours ago |
| README.md | Initial commit | 19 hours ago |

README.md

MeuPrimeiroProjeto

Meu primeiro projeto

About
Meu primeiro projeto
Readme
Activity
0 stars
1 watching
0 forks

Releases
No releases published
[Create a new release](#)

Packages
No packages published
[Publish your first package](#)

Pull Request :

- Um Pull Request é uma solicitação formal feita por um colaborador para integrar as alterações que ele fez em seu próprio fork (cópia) de um repositório para o repositório principal (ou original).

Github – Pull Request



MeuPrimeiroProjeto Public

Pin Unwatch 1

novapoesia had recent pushes 29 minutes ago

Compare & pull request

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main ← compare: novapoesia ✓ Able to merge. These branches can be automatically merged.

poesia Mikhael traz querer

Write Preview

H B I ≡ <> 🔗 ≡ ≡ ≡ @ ↗ ↶

Acrescimo do arquivo
-PoemaMikaelTrazQuerer.txt

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Pull Request :

- Ao subir um novo ramo no Github, ficará pendente a solicitação do pull request. Clique em **Compare & pull request**
- O Github já verifica se não há conflitos.
- Descreva as alterações feitas no ramo criado e escolha **Create pull request**

Github – Pull Request

poesia Mikhael traz querer #1

 Open PatriciaAngelini wants to merge 1 commit into `main` from `novapoesia` 

 Conversation **0**  Commits **1**  Checks **0**  Files changed **1**



PatriciaAngelini commented 2 minutes ago

Owner ...

Acrescimo do arquivo
-PoemaMikaelTrazQuerer.txt




  poesia Mikhael traz querer


6edcfdc


Add more commits by pushing to the `novapoesia` branch on `PatriciaAngelini/MeuPrimeiroProjeto`.



 Require approval from specific reviewers before merging
[Branch protection rules](#) ensure specific people approve pull requests before they're merged.

Add rule ×

 Continuous integration has not been set up
[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.

 This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).



Write

Preview

H B I       @   

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

143



Pull Request aprovando e incorporando na Main:


- Se você for o responsável para rever as alterações, você poderá checar e aprovar o pull request, clicando em Merge pull request

Github – Pull Request


poesia Mikhael traz querer #1



 Open PatriciaAngelini wants to merge 1 commit into `main` from `novapoesia` 

 Conversation **0**  Commits **1**  Checks **0**  Files changed **1**


 PatriciaAngelini commented 7 minutes ago Owner ...


Acrescimo do arquivo
-PoemaMikaelTrazQuerer.txt



  poesia Mikhael traz querer 6edcfdc












Add more commits by pushing to the `novapoesia` branch on PatriciaAngelini/MeuPrimeiroProjeto.



poesia Mikhael traz querer 


This commit will be authored by 62622184+PatriciaAngelini@users.noreply.github.com

Confirm merge Cancel

  H B I         

Leave a comment


Attach files by dragging & dropping, selecting or pasting them. 0/4

 Close pull request **Comment**

Pull Request Efetivando:









- O último passo, e o mais importante para não haver problemas de deploy, é efetivar as alterações no ramo main

Github – Pull Request






Write

Preview

H B I  <>     @   

Não aprovada pois o texto está fora dos padrões

Attach files by dragging & dropping, selecting or pasting them. 

 Close with comment 

Pull Request Reprovando:

- Atenção essa área é quando estamos reprovando o pull request: é marcado como "fechado" e não pode mais ser mesclado.

Github – Pull Request



Pull request successfully merged and closed

You're all set—the `novapoesia` branch can be safely deleted.

Delete branch

```
D:\PortableGit\git-cmd.exe

d:\Projetos\MeuPrimeiroProjeto>git branch
main
* novapoesia


d:\Projetos\MeuPrimeiroProjeto>git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

d:\Projetos\MeuPrimeiroProjeto> git branch -D novapoesia
Deleted branch novapoesia (was 6edcfdc).


d:\Projetos\MeuPrimeiroProjeto>
```

Apague o ramo:


- No Github: Não se esqueça de apagar o ramo criado para não haver confusões: **delete branch**
- No Git: Mude para o ramo principal fazendo **git checkout main** e depois apague o ramo fazendo **git branch -D [nome-do-ramo]**




 main  1 branch  0 tags


Go to file

Add file 

 Code 

 PatriciaAngelini Merge pull request #1 from PatriciaAngelini/novapoesia ... 663aafd 3 minutes ago  4 commits

| | | | |
|---|----------------------------|----------------------------|--------------|
|  | PoemaMikaelTrazQuerier.txt | poesia Mikhael traz querer | 1 hour ago |
|  | PoemaMikhael.txt | poema Mikhael | 11 hours ago |
|  | README.md | Initial commit | 19 hours ago |

README.md 

MeuPrimeiroProjeto

Meu primeiro projeto

Main:

- Agora você pode ver suas alterações na **main**

**Credenciais: Pode acontecer com
você**

Git – Commit precisa de autenticação

```
D:\PortableGit\git-cmd.exe
d:\Projetos\MeuPrimeiroProjeto>git commit -m "poema Mikhael"
Author identity unknown

*** Please tell me who you are.

Run

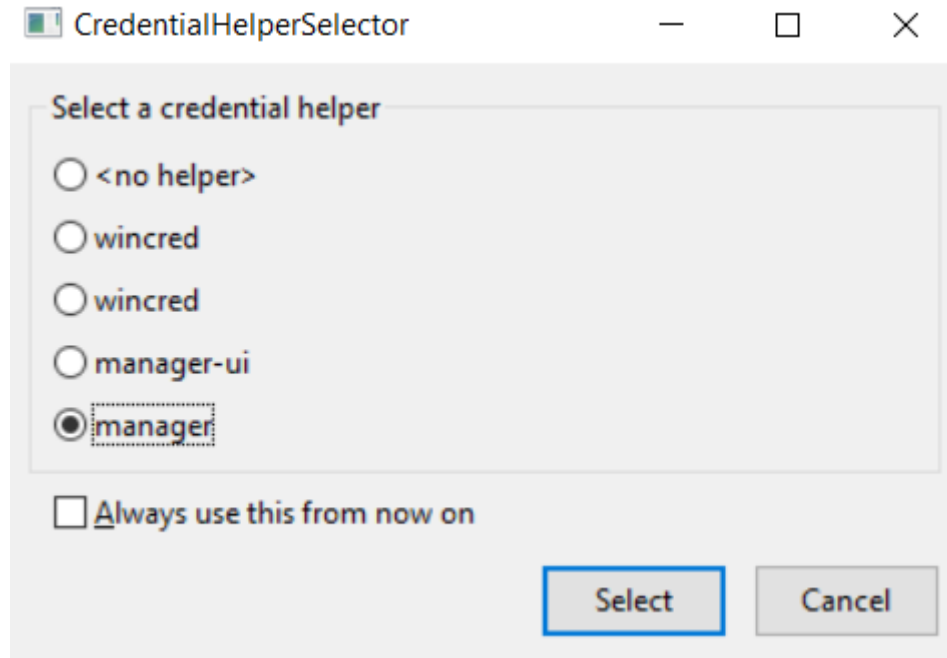
  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.
```

Quem é você?:

- Se você faz o fluxo de clonar um repo do Github e não configurou sua cópia local, ao tentar fazer o Commit, você terá erros de autenticação pois o Git não sabe quem você é. Use os comandos **git config --global user.email ["seu email"]** e **git config --global user.name ["seu nome"]**

Github – Push precisa de autenticação



Credenciais a cada Git Push:

- Quando você vai consolidar suas alterações no repositório do GitHub, ele deve ter as suas credenciais para fazer esse trabalho. Para isso ele pode abrir uma janela pedindo o método de autenticação. Você pode escolher o mais apropriado (wincred ou manager são boas opções)

EXERCÍCIOS

EXERCÍCIOS

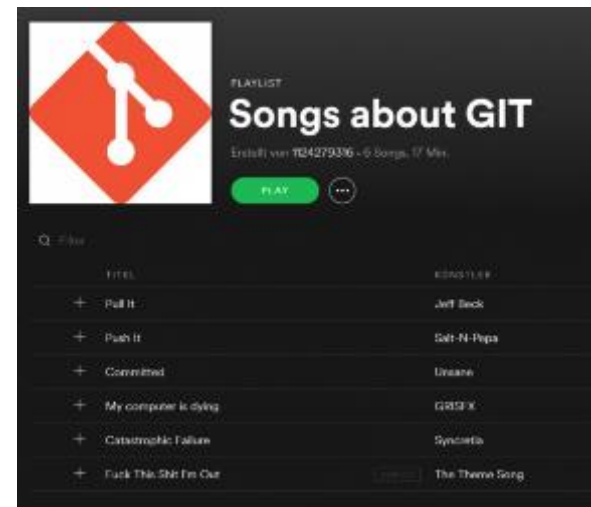
Trabalhando com o Git

- Comece seu trabalho com o GIT inicializando a linha de commando
- No seu diretorio de projeto (com 6 arquivos) crie um Repo novo.
 - Não se esqueça de se identificar.
 - Verifique se o diretório “escondido” foi criado
- Coloque 2 arquivos na área de STAGE
- Verique o status do seu Repo. Qual o nome do repositório que você está atualmente?
- Efetive o acrescimo dos dois arquivos ao projeto
- Verique o status do seu Repo
- Altere um arquivo
- Verique o status do seu Repo
- Adicione a área de stage tudo aquilo que foi modificado
- Verifique o status do seu projeto
- Efetive as alterações no Repo
- Verifique o status do seu Repo

EXERCÍCIOS

Trabalhando com o Git

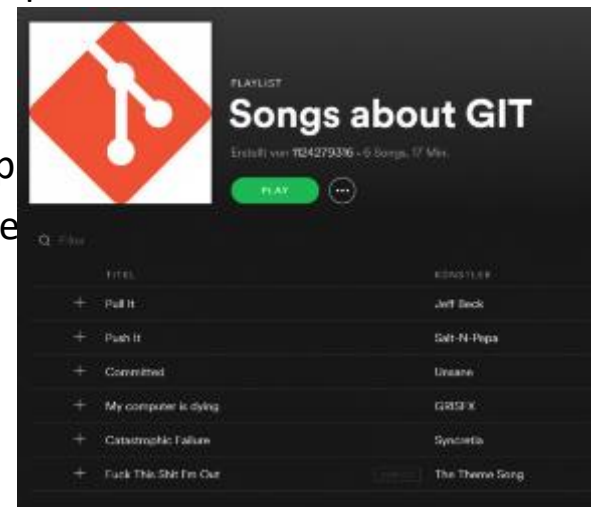
- Crie um novo ramo chamado PrimeiraFuncionalidade
- Verifique o status
- Garanta q vc está nesse ramo
- Adicione 2 arquivos na area de stage do seu novo ramo
- Verifique o status
- Efetive as alterações no seu novo ramo
- Volte um commit
- Efetive novamene as alterações nesse ramo
- Verifique as diferenças entre os dois ramos
- Junte o ramo novo ao ramo principal
- Apague o ramo PrimeiraFuncionalidade



EXERCÍCIOS

Trabalhando com o Git

- Crie um novo ramo chamado Segunda Funcionalidade
- Verifique o status
- Garanta q vc está nesse ramo
- Adicione 1 arquivos na area de stage do seu novo ramo
- Verifique o status
- Liste os arquivos desse ramo
- Efetive as alterações no seu novo ramo
- Volte ao ramo principal
- Faça uma alteração no arquivo do ramo principal
- Efetive as alteração direto no ramo principal
- Compare os ramos
- Pegue as alterações da feitas no ramo princip
- E incorpore no ramo Segunda Funcionalidade



REFERÊNCIAS



- OLIVEIRA, Jayr Figueiredo de; MANZANO, José Augusto N. G. **Algoritmos: Lógica para Desenvolvimento de Programação de Computadores**. 23ª Edição. São Paulo: Érica, 2010.
- MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C**. 2ª Edição. São Paulo: Pearson, 2008.

Copyright © 2023 Profa. Patrícia Angelini

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).