

[CRIANDO REPOSITÓRIO LOCAL](#)

[SALVAR ALTERÇÕES DO PROJETO](#)

[DESFAZER ALTERAÇÕES DO PROJETO](#)

[TRANSITANDO ENTRE VERSÕES / *COMMIT*S](#)

CRIANDO REPOSITÓRIO LOCAL

O Git cria um repositório local no diretório do projeto, que pode ser clonado posteriormente para um repositório remoto, como o GitHub. Também é possível fazer o contrário, como mostrado no PDF “4_Repositorio_Remoto”.

1. Dentro do diretório do projeto, clique com o botão direito do mouse e selecione “***Git Bash Here***”
2. No terminal que aparecer, digite o comando a seguir para criar um repositório local nessa pasta

```
git init
```

3. A pasta .git (repositório local) será criada no diretório de forma oculta, para exibi-la é necessário mudar as configurações de exibição no diretório

SALVAR ALTERAÇÕES DO PROJETO

Antes de armazenar o conteúdo e formar um grafo de *commits*, o Git precisa gerenciar as mudanças do projeto. Para isso, existe o que chamamos de container, onde se deve adicionar todos as pastas e arquivos criados ou modificados.

- | | | |
|-------------------------|---|--------------------------------------------------------|
| <code>git add</code> | - | Adiciona os arquivos no container |
| <code>git status</code> | - | Verifica o que está armazenado no container |
| <code>git commit</code> | - | Identifica e armazena o container no repositório local |
| <code>git log</code> | - | Mostra as mudanças recentes |



Para verificar se algum arquivo foi criado ou modificado no diretório do projeto e se há alguma ação pendente (*changes to be committed*) para armazenar as mudanças no repositório, utiliza-se

`git status`

Arquivos recém-criados aparecerão em vermelho no status e será solicitado que estes sejam adicionados ao container utilizando

`git add <nome_do_arquivo>`

Após utilizar o `git add`, o comando `git status` mostrará os arquivos em verde, indicando que estes estão dentro do container. Resta, então, identificar o container e armazená-lo no repositório. Para isso, utiliza-se

`git commit -m "<comentário>"`

Ao fazer isso, é possível verificar utilizando `git status` que não há mais pendências. Para verificar os registros de mudanças, utiliza-se um dos comandos

```
git log
```

```
git log --oneline
```

Dica: é possível, em vez de usar `git add` e depois `git commit -m`, usar um único comando que equivale aos dois juntos. Para isso, utiliza-se

```
git commit -am "<comentário>"
```

EXEMPLO 1

1. Temos um diretório "Projeto" já com a pasta `.git` e alguns *commits* feitos. Deseja-se adicionar um novo arquivo no projeto. Abriu-se o terminal no diretório
2. No diretório "Projeto", criou-se um arquivo "d.txt"
3. No terminal, utilizou-se o comando `git status` para ver a alteração

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    d.txt

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

4. Para adicionar o arquivo ao container, utiliza-se

```
git add d.txt
```

Mas **para adicionar todas as mudanças no container de uma vez**, utilizou-se

```
git add *
```

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git add *

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   d.txt
```

5. O arquivo está no container, que deve ser identificado e armazenado no repositório. Utilizou-se

```
git commit -m "Criado o arquivo d.txt"
```

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git commit -m "Criado o arquivo d.txt"
[master d98adda] Criado o arquivo d.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 d.txt

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
nothing to commit, working tree clean
```

6. Utilizando `git log` e `git log --oneline`, é possível ver as alterações feitas (vale perceber que antes de criar “d.txt”, outras mudanças já tinham sido feitas no Projeto. Elas continuam registradas nos *logs*)

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git log
commit d98adda61fdb670883d7fd1e2279f13677ce660d (HEAD -> master)
Author: Vinicius Passareli <vi.passareli@hotmail.com>
Date: Thu May 6 04:14:31 2021 -0300

    Criado o arquivo d.txt

commit 02e31c5b6904468dee0cb4418c26e68694fdad3a
Author: Vinicius Passareli <vi.passareli@hotmail.com>
Date: Thu May 6 03:20:48 2021 -0300

    modificados a.txt e b.txt. Criado c.txt

commit 78e3e5cc3065d0529d4639bef5bacad40c282b24
Author: Vinicius Passareli <vi.passareli@hotmail.com>
Date: Thu May 6 03:16:50 2021 -0300

    Criado os arquivos a.txt e b.txt
```

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git log --oneline
d98adda (HEAD -> master) Criado o arquivo d.txt
02e31c5 modificados a.txt e b.txt. Criado c.txt
78e3e5c Criado os arquivos a.txt e b.txt
```

Pelo Exemplo 1, é possível ver que cada *commit* criado possui um código de identificação (em amarelo) que chamamos de **hash**. Também é possível ver que o último *commit* feito (versão atual do projeto salvo) sempre conterà o **HEAD**.

DESFAZER ALTERAÇÕES DO PROJETO

Para verificar as últimas alterações feitas e que ainda não foram salvas, utiliza-se o comando

```
git diff
```

a) Caso não deseja-se salvar as mudanças feitas, e elas **não tenham sido adicionadas ao container**, basta utilizar os dois comandos a seguir

```
git status
```

```
git restore <nome_do_arquivo>
```

EXEMPLO 2

1. A partir do Exemplo 1, editou-se o arquivo “d.txt”, adicionando uma linha com o texto “> Estou alterando d.txt <”. Para verificar a mudança, utilizou-se

```
git diff
```

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git diff
diff --git a/d.txt b/d.txt
index e69de29..957675d 100644
--- a/d.txt
+++ b/d.txt
@@ -0,0 +1 @@
+> Estou alterando d.txt <
\ No newline at end of file
```

2. Para descartar essa mudança e não salvá-la, utilizou-se

```
git status
```

seguido de

```
git restore d.txt
```

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   d.txt

no changes added to commit (use "git add" and/or "git commit -a")

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git restore d.txt
```

3. As alterações em “d.txt” foram desfeitas com sucesso. Para verificar, utilizou-se os comandos a seguir

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
nothing to commit, working tree clean

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git diff
```

b) Caso as mudanças indesejadas **tenham sido adicionadas ao container** utilizando o comando `git add` (mas sem fazer o **commit**), para desfazer, utiliza-se os comandos a seguir

```
git status
git restore --staged <nome_do_arquivo>
git status
git restore <nome_do_arquivo>
```

EXEMPLO 3

1. A partir do Exemplo 1, editou-se o arquivo “d.txt” e utilizou-se o `git add` para adicioná-lo ao container

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git add d.txt

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   d.txt
```

2. Para desfazer essas alterações e descartar a mudança, utilizou-se a sequência de comandos a seguir

```
git status
git restore --staged d.txt
git status
git restore d.txt
```

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   d.txt

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git restore --staged d.txt

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   d.txt

no changes added to commit (use "git add" and/or "git commit -a")

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git restore d.txt
```

3. As alterações em “d.txt” foram desfeitas com sucesso

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
nothing to commit, working tree clean

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git diff
```

c) Caso as mudanças **tenham sido adicionadas ao container e realizou-se o *commit***, para desfazer, utiliza-se o comando

```
git reset --hard <hash>
```

Onde o campo *hash* deve conter **o *hash* da versão anterior que deseja-se retornar**, para desfazer as alterações indesejadas da versão atual.

Dica: não é necessário utilizar o *hash* inteiro, apenas a versão resumida exibida pelo comando `git log --oneline` já é suficiente.

EXEMPLO 4

1. A partir do Exemplo 1, editou-se o arquivo “d.txt”, utilizou-se o comando simplificado (`git commit -am`) para adicioná-lo diretamente ao container, realizando a identificação, e utilizou-se `git log` para obter os *hashes* dos *commits*

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   d.txt

no changes added to commit (use "git add" and/or "git commit -a")

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git commit -am "Esse commit sera removido"
[master 128a53e] Esse commit sera removido
1 file changed, 1 insertion(+)

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git log --oneline
128a53e (HEAD -> master) Esse commit sera removido
d98adda Criado o arquivo d.txt
02e31c5 modificados a.txt e b.txt. Criado c.txt
78e3e5c Criado os arquivos a.txt e b.txt
```

2. Para remover o último *commit*, realiza-se um *reset* para o *hash* do *commit* anterior desejado. Nesse caso, deseja-se voltar para a versão de *hash* **d98adda**

`git reset --hard d98adda`

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git reset --hard d98adda
HEAD is now at d98adda Criado o arquivo d.txt
```

3. As alterações em “d.txt” foram desfeitas com sucesso

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git status
On branch master
nothing to commit, working tree clean

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git diff
```

```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git log --oneline
d98adda (HEAD -> master) Criado o arquivo d.txt
02e31c5 modificados a.txt e b.txt. Criado c.txt
78e3e5c Criado os arquivos a.txt e b.txt
```


TRANSITANDO ENTRE VERSÕES / COMMITS

Para ver o grafo de *commits* e o *hash* de cada versão do projeto, se utiliza

```
git log --graph  
git log --oneline --graph --all
```

É importante ter em mente que o Git não gera uma nova cópia, ou container, a cada mudança salva. Ele apenas altera o container já existente e atribui a ele um novo *commit hash*. Mas ele ainda mantém todos os *hashes* das versões anteriores, e é com isso que ele consegue rastrear as alterações realizadas.

Para rastrear as mudanças entre as versões, é necessário mover o *HEAD* da versão presente para a versão desejada. Para isso, utiliza-se o comando abaixo, inserindo o *hash* da versão que deseja-se rastrear

```
git checkout <hash_desejado>
```

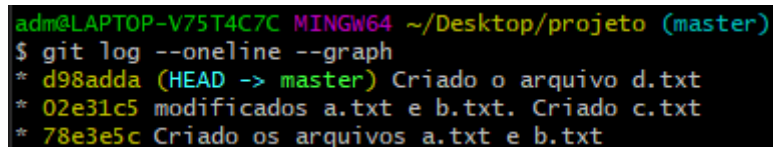
Ao utilizar o comando, todo o diretório do Projeto e seus arquivos **irão voltar ao estado que estavam quando essa versão antiga foi criada**. Para voltar para a última versão criada, utiliza-se

```
git checkout master
```

EXEMPLO 5

1. A partir do que foi feito no Exemplo 1, utilizando

```
git log --oneline --graph
```



```
adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)  
$ git log --oneline --graph  
* d98adda (HEAD -> master) Criado o arquivo d.txt  
* 02e31c5 modificados a.txt e b.txt. Criado c.txt  
* 78e3e5c Criado os arquivos a.txt e b.txt
```

2. Deseja-se voltar para a versão com *hash* **02e31c5**, onde “d.txt” ainda não teria sido criado. Utiliza-se, então,

```
git checkout 02e31c5
```

```

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git checkout 02e31c5
Note: switching to '02e31c5'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 02e31c5 modificados a.txt e b.txt. Criado c.txt

```

3. Nessa versão, não existe o arquivo “d.txt” criado no Exemplo 1. Isso pode ser confirmado utilizando `git log` ou até mesmo observando o diretório do Projeto, onde o arquivo não estará mais presente

```

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto ((02e31c5...))
$ git log --oneline
02e31c5 (HEAD) modificados a.txt e b.txt. Criado c.txt
78e3e5c Criado os arquivos a.txt e b.txt

```

4. Para retornar à versão final original, onde “d.txt” tinha sido criado, basta utilizar o comando

`git checkout master`

```

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto ((02e31c5...))
$ git checkout master
Previous HEAD position was 02e31c5 modificados a.txt e b.txt. Criado c.txt
Switched to branch 'master'

adm@LAPTOP-V75T4C7C MINGW64 ~/Desktop/projeto (master)
$ git log --oneline
d98adda (HEAD -> master) Criado o arquivo d.txt
02e31c5 modificados a.txt e b.txt. Criado c.txt
78e3e5c Criado os arquivos a.txt e b.txt

```

5. Nesse ponto, o arquivo “d.txt” volta a ser exibido no diretório do Projeto