

Vídeo Aula:

<https://app.rocketseat.com.br/node/o-guia-estelar-de-git>

Introdução

- Porque entender o Git?
 - Para podermos mudar algo no nosso projeto sem perder o que já tínhamos feito;
 - Para arquivar/salvar o nosso código em algum lugar;
 - Criar versões do projeto;
 - Para voltar uma versão do nosso projeto;
 - Etc.

O que é GIT

Controle de Versão

- VCS:
 - Version Control System - Controle de Versão.
- Controle:
 - Controla o nosso projeto;
 - Registra as alterações feitas em nosso projeto em um arquivo;
 - Permite reverter para um estado anterior um arquivo ou até mesmo um projeto inteiro;
 - Permite comparar um arquivo atual com o mesmo de uma determinada versão;
 - Permite identificarmos quem modificou pela última vez;
 - Permite identificarmos quando um determinado problema começou a acontecer.
- Versão:
 - Versões do nosso projeto.

Tipos de Controles de Versão

- Sistemas Locais:
 - Como funciona:
 - Copiamos um arquivo de um diretório para outro;
 - Vantagens:
 - Muito simples;
 - Desvantagens:
 - Propenso de erros, pois, podemos facilmente sobrescrever arquivos. Exemplo: salvei a versão 3 na versão 2 e continuo escrevendo na versão 3, cuja não contém as alterações que foram feitas na versão 2;
 - É mais difícil compartilhar, pois, está na nossa máquina local;
 - Se o disco rígido for corrompido perdemos praticamente tudo do nosso projeto.

**O MAC até possui uma extensão para DEV que possui um controle de versão, porém, não permite o compartilhamento.*

- Sistemas Centralizados:
 - Exemplos:
 - VCS, Subversion, Perforce.
 - Como funciona:
 - Fica em um único servidor onde vários desenvolvedores acessam e mantém suas versões.
 - Vantagens:

- Permite o compartilhamento;
 - Controle das atividades dos colaboradores (quem pode fazer o que).
- Desvantagens:
 - Se o servidor cair ou não termos acesso à internet;
 - Se o disco rígido for corrompido perdemos praticamente tudo do nosso projeto.
- Sistemas Distribuídos:
 - Exemplo:
 - GIT;
 - Mercurial;
 - Bazar;
 - Darcs.
 - Como funciona:
 - Ele vai duplicar/clonar os repositórios localmente. Ou seja, se três pessoas trabalham em um mesmo projeto, as três terão um clone local do repositório remoto;
 - Seguindo o exemplo acima, temos três máquinas com uma cópia do projeto, ou seja, três backups, portanto, se um servidor cair ainda temos duas máquinas funcionando;
 - Ele une os dois mundos, o servidor local e o servidor remoto (ex: GitHub);
 - Se a internet cair, conseguimos continuar trabalhando com o repositório local e depois basta submeter para o repositório remoto para que os demais desenvolvedores tenham acesso à versão mais recente.

O que é o GIT

- É um sistema de controle de versão distribuído;
- É Open-source, ou seja, de código gratuito e aberto;
- O que ele faz?
 - Pontos na história: mantém o histórico de alterações no código e permite voltarmos para qualquer ponto da história;
 - Controla o fluxo de novas funcionalidades: branch (pontos na história em uma linha da história paralela da linha principal), vários devs no mesmo projeto, análise e resolução de conflitos.

Instalando Git

Instalando

- Acesse o link para download: <https://git-scm.com/downloads>.

Configuração Inicial

- Vamos fazer uma única vez por computador;
- Pode ser alterada rodando os comandos novamente;
- Identidade:
 - Serve para podermos identificar quem está fazendo as alterações em determinado ponto da história;
 - Comandos:
 - `git config --global user.name "Seu Nome";`
 - `git config --global user.email email@email.com;`
 - Essas informações podem ser alteradas para um projeto específico, basta tirarmos o `--global`.
- Para usar um editor diferente do vim, por exemplo, o VS Code, basta rodar o seguinte comando: `git config --global core.editor "code -w";`

- Para ver as configurações que acabamos de fazer basta rodar o seguinte comando: `git config --list`;
- Onde fica o arquivo de configurações: `C:\Users\Usuário\.gitconfig`;
- Para ver o conteúdo do arquivo de configuração basta rodar o seguinte comando: `cat ~/.gitconfig`;
- Para alterar as informações para um projeto específico basta executar os seguintes comandos dentro da pasta do projeto:
 - `git init`: cria um repositório chamado `.git`;
 - `cd .git`: entrar no diretório;
 - `git config --list`: ver o que temos de configuração para esse projeto (configurações globais e do projeto).

**Devemos manter sempre o mesmo e-mail utilizado no GitHub.*

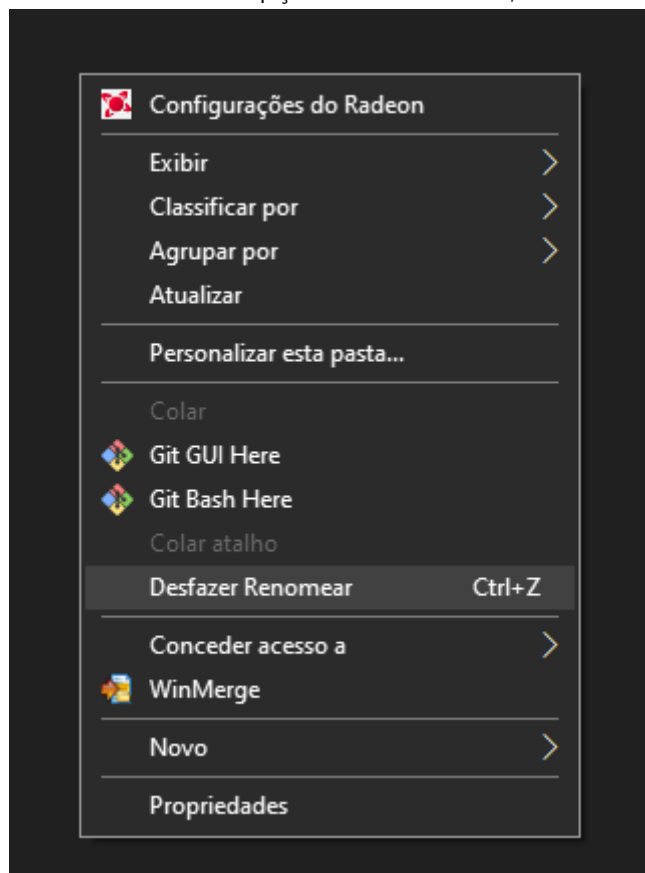
Git Help

- `git help`: exibe diversos comandos mais básicos;
- `git help log`: exibe um manual do git com as informações do git log;
- `:q` : Para sair do manual.

Começando

Iniciando um Repositório

- Crie um diretório chamado `git-aula` na pasta Documentos do seu computador;
- Vá com o git até o diretório criado: `cd Documents/git-aula/` ou com o botão direito dentro do diretório selecione a opção `Git bash here`;



- Liste tudo o que temos dentro do repositório: `ls`;
- Crie um repositório chamado `.git`: `git init`;
- Acessando a pasta `git-aula` não é possível visualizar o diretório `.git` criado, mas através do terminal com o seguinte comando é possível: `ls -a`;

O .git Guarda o Histórico do Projeto

- O diretório .git guarda todos os pontos da história do nosso projeto;
- Para visualizar tudo o que tem dentro do diretório: `ls -al .git:`

Resultado do comando:

```
total 11
drwxr-xr-x 1 Vanessa 197121  0 Apr 10 21:00 ./
drwxr-xr-x 1 Vanessa 197121  0 Apr 10 21:00 ../
-rw-r--r-- 1 Vanessa 197121 23 Apr 10 21:00 HEAD
-rw-r--r-- 1 Vanessa 197121 130 Apr 10 21:00 config
-rw-r--r-- 1 Vanessa 197121 73 Apr 10 21:00 description
drwxr-xr-x 1 Vanessa 197121  0 Apr 10 21:00 hooks/
drwxr-xr-x 1 Vanessa 197121  0 Apr 10 21:00 info/
drwxr-xr-x 1 Vanessa 197121  0 Apr 10 21:00 objects/
drwxr-xr-x 1 Vanessa 197121  0 Apr 10 21:00 refs/
```

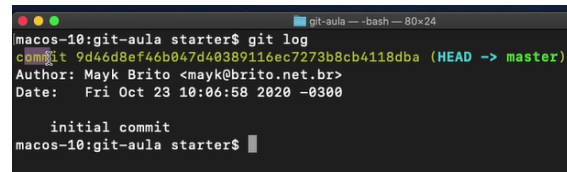
**HEAD: ponto da história;*

**Se deletarmos essa pasta perdemos todo o histórico do nosso projeto.*

- Para visualizar as configurações para o projeto que ficará na pasta aula-git: `cat .git/config`

Git Log

- Para visualizar os pontos da história: `git log`;

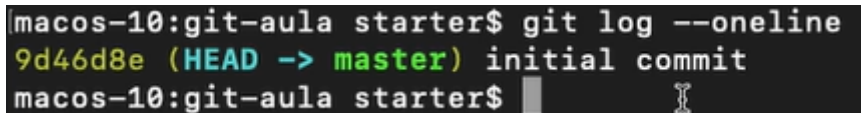


```
macos-10:git-aula starter$ git log
commit 9d46d8ef46b047d40389116ec7273b8cb4118dba (HEAD -> master)
Author: Mayk Brito <mayk@brito.net.br>
Date:   Fri Oct 23 10:06:58 2020 -0300

    initial commit
macos-10:git-aula starter$
```

commit
nome_commit hash(sha1-1)
(HEAD -> master) aponta o ponto da história, nesse caso estamos na branch master, nossa linha do tempo principal
autor e
data

- Para visualizar os pontos da história de forma resumida: `git log --oneline`;



```
macos-10:git-aula starter$ git log --oneline
9d46d8e (HEAD -> master) initial commit
macos-10:git-aula starter$
```

- Para visualizar o log dos últimos 5 commit: `git log -n 5`;
- Para visualizar o log desde uma determinada data: `git log --since=2020-10-01`;
- Para visualizar o log anteriormente a uma determinada data: `git log --until=2020-10-01`;
- Para visualizar o log por autor: `git log --author=Vanessa`;
- Para visualizar o log por expressão regular. Nesse exemplo estamos buscando pela mensagem do commit: `git log --grep="init"`.

O Primeiro Commit

- Primeiro vamos criar um arquivo dentro do repositório git-aula para que possamos ter o que versionar. Comandos:
 - `vim file.txt`;
 - Inserir o texto: `i`;
 - Digite o texto de exemplo: texto;
 - Escrevendo o arquivo e saindo: `:wq`;
 - Preparando o arquivo para adicionar um ponto na história (o ponto significa que é pra preparar tudo/todos arquivos que tem dentro do diretório git-aula): `git add .`;

- Adicionando o ponto na história ([-m "initial commit"] representa a mensagem do meu commit): `git commit -m "initial commit"`.

Conceitos

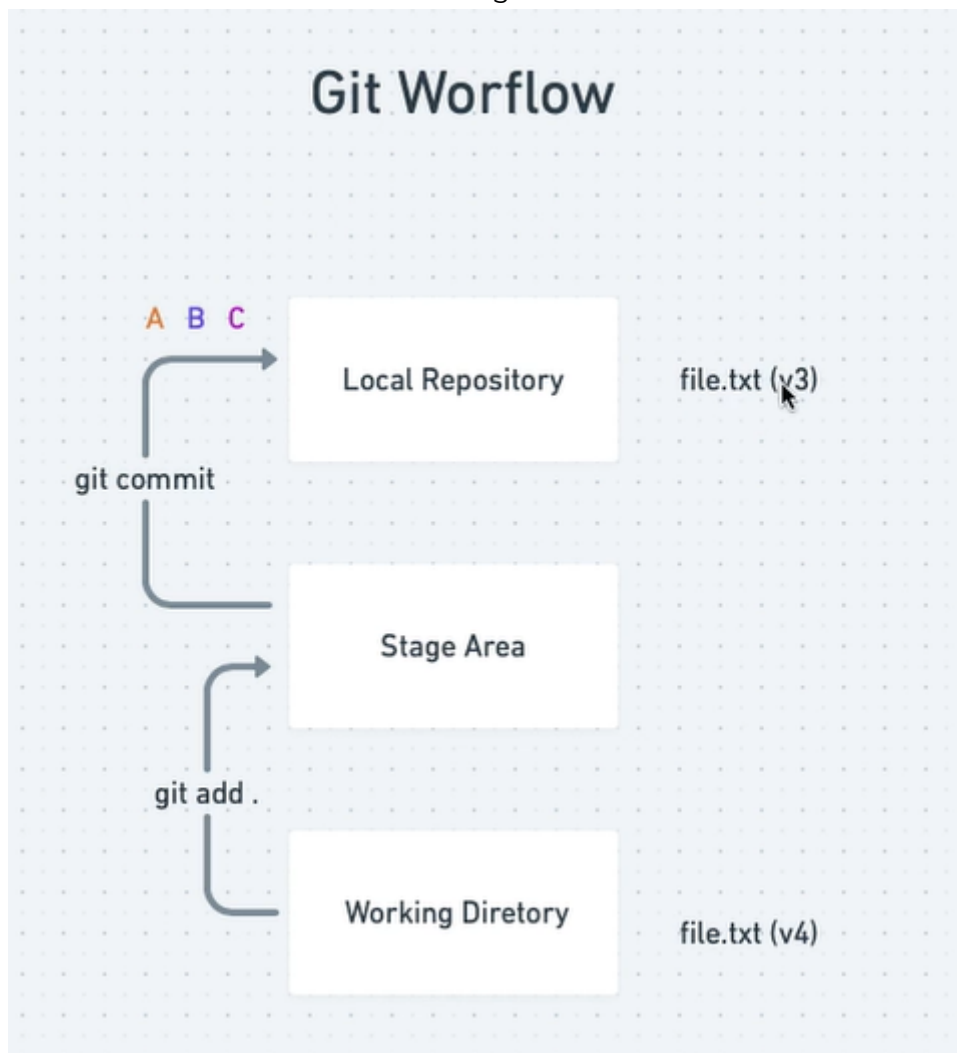
Estágios do Arquivo

- Três estados do arquivo dentro do fluxo do git:
 - Iniciar um repositório dentro do nosso projeto: `git init`. Nesta etapa os arquivos estão no estado Working Directory;
 - Git Add: Nesta etapa os arquivos estão no estado Stage Area (arquivo está preparado para ser comitado);
 - Git Commit: Nesta etapa os arquivos estão no estado Local Repository (arquivo está no repositório local e o ponto da história é criado e evitamos perder esse arquivo).

**Todo arquivo modificado deve passar por esses três estados acima!*

Git Workflow

- Como funciona o fluxo de trabalho do git:



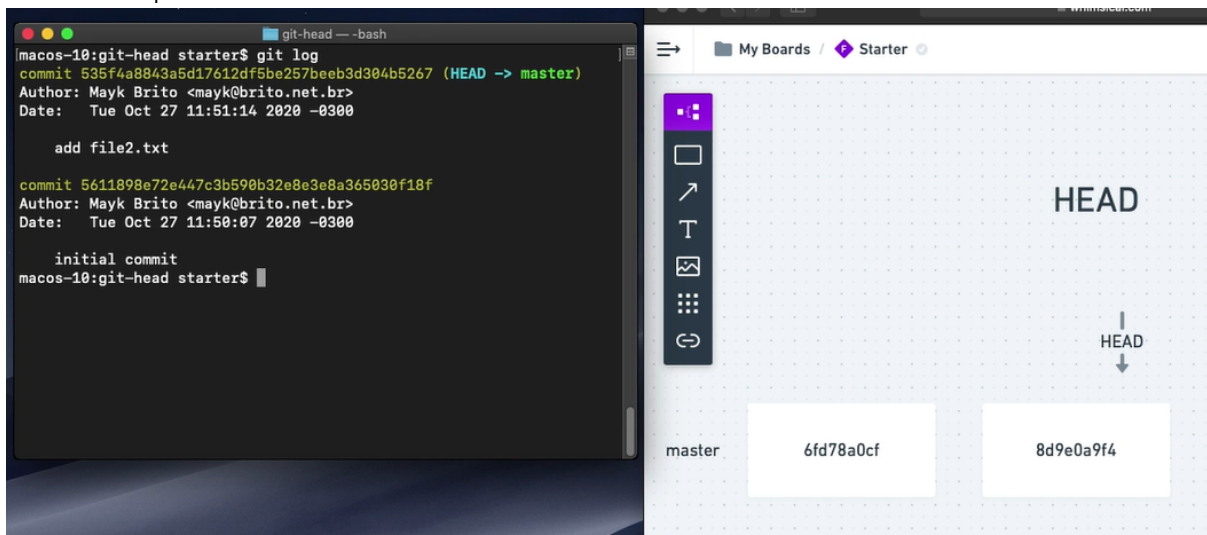
**A, B e C representam as versões 1, 2 e 3, porém, já posso ir trabalhando uma versão 4 no meu diretório local (minha máquina).*

Hash SHA-1

- Nome que é dado a cada commit é criado através de um valor HASH chamado Hash Values (SHA-1). Esse hash contém 40 caracteres é hexadecimal (vai de 0 a 9 e de A a F);
- A cada commit gera um checksum (soma que vai checar/conferir alguma coisa). Ele vai converter os dados da nossa alteração em números no formato SHA-1;
- Os dados são convertidos para garantir uma integridade de dados, ou seja, os mesmos dados gerados da mesma maneira irá gerar o mesmo número, porém, é difícil fazermos um commit sem alterações, portanto, dificilmente commits terão o mesmo nome;
- Dentro do nome temos as seguintes informações:
 - Snapshot: Uma fotografia de como estava o meu projeto naquele momento;
 - Hash dos metadados (dados convertidos);
 - Informações adicionais:
 - Autor;
 - Mensagem e;
 - Parent: hash do commit anterior.

HEAD

- HEAD representa um ponteiro que irá apontar em que ponto da história estamos, além de permitir navegarmos para pontos diferentes da história em nossa linha do tempo:



**Master é a nossa linha do tempo principal. Repare que a cada commit o HEAD aponta para um hash diferente, ou seja, para um outro ponto da história.*

Revisão Prática

Ação	Comando	Workflow Git
Criar um diretório na pasta Documentos chamado de git-aula	mkdir git-aula	
Entrar no diretório criado	cd git-aula	
Criar o repositório .git	git init	Working Directory
Criar um arquivo dentro do diretório	touch file1.txt	

Visualizar melhor o que eu fiz até agora:

git status

Fala em qual branch eu estou

Quantos commits

Se existem arquivos untracked (que não foram comitados) ainda

Se existe algo que deve ir para o commit

```
Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

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

Criar um novo arquivo dentro do diretório	touch file2.txt	
Preparar o arquivo para ser comitado	git add file1.txt	Stage Area
Preparar o arquivo para ser comitado (a diferença do de cima é que nesse eu estou enviando todo o meu repositório e não apenas 1 arquivo específico)	git add .	Stage Area
Removendo um arquivo da Stage Area para que o mesmo não suba no commit	git rm --cached file2.txt	
Dando o commit. O ponto da história é criado e evitamos perder esse arquivo.	git commit -m 'initial commit'	Local Repository

```
Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git commit -m 'initial commit'
[master (root-commit) 2e6d28d] initial commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file1.txt
```

root: primeiro commit

2e6d28d: parte do hash

initial commit: primeiro commit

1 file changed: foi um arquivo modificado

0 insertions(+): não tem nenhuma inserção nesse arquivo

0 deletions(-): não tem nenhuma deleção nesse arquivo (quantidade de linhas deletadas do arquivo)

create mode 100644 file1.txt: a numeração representa o tipo de permissão do arquivo e temos também o nome do arquivo.

Visualizando o LOG	git log	
<pre>\$ git log commit 2e6d28dc54221702cae9f2a43aafb8d5bf6dfaa9 (HEAD -> master) Author: Vanessa Ichikawa <vanessavallarini@gmail.com> Date: Sat Apr 10 22:36:49 2021 -0300 initial commit</pre>		
Alterando um arquivo	vim file1.txt Digite qualquer coisa :wq	
Verificando se hou alteraçõs	git status	
<pre>vanessa@VanessaNote MINGW64 ~/Documents/git-aula (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: file1.txt no changes added to commit (use "git add" and/or "git commit -a")</pre>		
Preparar o arquivo para ser comitado - versão 2 do arquivo	git add file1.txt	Stage Area
Dando o commit. O ponto da história é criado e evitamos perder esse arquivo - versão 2 do arquivo	git commit -m 'version 2'	Local Repository
Agora a HEAD está apontando para a versão 2 do arquivo		


```

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git log
commit 8517d167628465a635a8e4aecb76f3110e2ef419 (HEAD -> master)
Author: Vanessa Ichikawa <vanessavallarini@gmail.com>
Date: Sat Apr 10 22:50:31 2021 -0300

    version 2

commit 5c4edc4aadbcb0a22c53a91f09b7ceb76ccd87f
Author: Vanessa Ichikawa <vanessavallarini@gmail.com>
Date: Sat Apr 10 22:45:30 2021 -0300

    second commit

commit 2e6d28dc54221702cae9f2a43aafb8d5bf6dfaa9
Author: Vanessa Ichikawa <vanessavallarini@gmail.com>
Date: Sat Apr 10 22:36:49 2021 -0300

    initial commit

```

Alterando Arquivos

Adicionando Arquivos com Git Add

Criar um novo arquivo dentro do diretório	touch README.md	
Preparar o arquivo para ser comitado	git add file1.txt	Stage Area
Preparar o arquivo para ser comitado (a diferença do de cima é que nesse eu estou enviando todo o meu repositório e não apenas 1 arquivo específico)	git add README.md	Stage Area
Preparar o arquivo para ser comitado (a diferença dos de cima é que nesse eu estou enviando apenas arquivos com uma extensão específica)	git add *.md	Stage Area

Editando Arquivos

Alterando um arquivo	vim file1.txt Digite qualquer coisa :wq	Working Directory
----------------------	---	-------------------

Modificações com Git Diff

- Supondo que realizei diversas alterações no meu projeto e não me recordo exatamente do que foi alterado. Nesse caso o git pode auxiliar exibindo o que tem de diferente entre o Working Directory e o Local Repository. Por exemplo:

Deletando arquivos	Delete os dois arquivos de nome file1.txt e file2.txt, mantendo apenas o README.md
Crie 3 novos arquivos	touch working_directory.md touch stage_area.md touch repository_local.md
Adicione os arquivos para o Stage Area	git add .
Faça alterações nos arquivos	adicione linhas/ remova linhas
Pedir para o git informar o que foi alterado	git diff

```
Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git diff
diff --git a/README.md b/README.md
index e69de29..d61b5d7 100644
--- a/README.md
+++ b/README.md
@@ -0,0 +1 @@
+dssff
\ No newline at end of file
diff --git a/stage_area.md b/stage_area.md
index e69de29..7cba086 100644
--- a/stage_area.md
+++ b/stage_area.md
@@ -0,0 +1 @@
+asdsdsadsd
\ No newline at end of file
diff --git a/working_directory.md b/working_directory.md
index da1abbc..e69de29 100644
--- a/working_directory.md
+++ b/working_directory.md
@@ -1 +0,0 @@
-der ewr ewr
\ No newline at end of file
```

```
$ git diff
diff --git a/README.md b/README.md
index e69de29..d61b5d7 100644
--- a/README.md
+++ b/README.md
@@ -0,0 +1 @@
+dssff
\ No newline at end of file
//no arquivo README foi adicionado uma linha "+dssff"
diff --git a/stage_area.md b/stage_area.md
index e69de29..7cba086 100644
--- a/stage_area.md
+++ b/stage_area.md
@@ -0,0 +1 @@
+asdsdsadsd
\ No newline at end of file
```

```
//no arquivo stage_area.md foi adicionado uma linha "+asdsdsadsd"
diff --git a/working_directory.md b/working_directory.md
index da1abbc..e69de29 100644
--- a/working_directory.md
+++ b/working_directory.md
@@ -1,0,0 @@
-derewrewr
\ No newline at end of file
//no arquivo working_directory.md foi deletada uma linha "-derewrewr"
```

Git Diff Staged

- Supondo que realizei diversas alterações no meu projeto e não me recordo exatamente do que foi alterado. Nesse caso o git pode auxiliar exibindo o que tem de diferente entre o Stage Area e o Local Repository. Por exemplo:

Crie novo arquivo	touch teste.txt
Adicione os arquivos para o Stage Area	git add .
Faça alterações nos arquivos	adicione linhas/ remova linhas
Pedir para o git informar o que foi alterado	git diff --staged

```
Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git diff --staged
diff --git a/file2.txt b/README.md
similarity index 100%
rename from file2.txt
rename to README.md
diff --git a/file1.txt b/file1.txt
deleted file mode 100644
index 837c454..0000000
--- a/file1.txt
+++ /dev/null
@@ -1,0,0 @@
-linha
diff --git a/repository_local.md b/repository_local.md
new file mode 100644
index 0000000..8fab842
--- /dev/null
+++ b/repository_local.md
@@ -0,0 +1 @@
+dsfsdff
\ No newline at end of file
diff --git a/stage_area.md b/stage_area.md
new file mode 100644
index 0000000..e69de29
diff --git a/working_directory.md b/working_directory.md
new file mode 100644
index 0000000..da1abbc
--- /dev/null
+++ b/working_directory.md
@@ -0,0 +1 @@
+derewrewr
\ No newline at end of file
```

```
$ git diff --staged
diff --git a/file2.txt b/README.md
```

```
similarity index 100%
rename from file2.txt
rename to README.md
diff --git a/file1.txt b/file1.txt
deleted file mode 100644
index 837c454..0000000
--- a/file1.txt
+++ /dev/null
@@ -1 +0,0 @@
-linha
//arquivo file1.txt foi deletado
diff --git a/repository_local.md b/repository_local.md
new file mode 100644
index 0000000..8fab842
--- /dev/null
+++ b/repository_local.md
@@ -0,0 +1 @@
+dsfsdff
\ No newline at end of file
//arquivo repository_local.md foi criado e adicionada a linha +dsfsdff
diff --git a/stage_area.md b/stage_area.md
new file mode 100644
index 0000000..e69de29
diff --git a/working_directory.md b/working_directory.md
new file mode 100644
index 0000000..da1abbc
--- /dev/null
+++ b/working_directory.md
@@ -0,0 +1 @@
+derewrewr
\ No newline at end of file
//arquivo repository_local.md foi criado e adicionada a linha +derewrewr
```

Deletando Arquivos

- Considerando que nosso repositório esteja da seguinte forma (**deletamos manualmente os arquivos** file1 e file2(estes foram renomeados, mas suponhamos que tenham sido deletados)):

```

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:   file2.txt -> README.md
    deleted:   file1.txt
    new file:   repository_local.md
    new file:   stage_area.md
    new file:   working_directory.md

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:  README.md
    modified:  stage_area.md
    modified:  working_directory.md

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

```

Temos 5 alterações em nossa Stage Area para serem comitadas:

```

renamed: file2.txt -> README.md
deleted: file1.txt
new file: repository_local.md
new file: stage_area.md
new file: working_directory.md

```

Temos 3 alterações que não estão na stage area:

```

modified: README.md
modified: stage_area.md
modified: working_directory.md

```

Temos 1 alteração que não está sendo rastreada:

```

teste.txt

```

Vamos restaurar as alterações na Stage Area, pois, no momento vamos manter apenas a alteração de deleção do arquivo file2.txt para que essa possa ser comitada

```

git restore --staged README.md
git restore --staged repository_local.md
git restore --staged stage_area.md
git restore --staged working_directory.md
git restore --staged file1.txt

```

Visualizando novamente as alterações:

```

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git status
on branch master
changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    file2.txt

changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    file1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md
    repository_local.md
    stage_area.md
    teste.txt
    working_directory.md

```

Comitar a alteração feita em nossa Stage Area

git commit -m 'delete file2.txt'

- Agora **deletando com o git:**

Certifique-se de que o arquivo a ser deletado já foi dado commit e está no Local Repository

```

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git status
on branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    testeRemocao.txt

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

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git add .

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git commit -m 'add testeRemocao.txt'
[master b27bc70] add testeRemocao.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 testeRemocao.txt

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git rm testeRemocao.txt
rm 'testeRemocao.txt'

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$

```

Renomeando Arquivos

- Manualmente

Renomeie o arquivo manualmente

working_directory
para
1-working_directory

git status

```
Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git status
on branch master
changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    working_directory.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        1-working_directory.md

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

Remova o working_directory

git rm working_directory.md

Adicione o 1-working_directory na Stage Area

git add 1-working_directory.md

git status

```
Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git status
on branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    working_directory.md -> 1-working_directory.md
```

- Com o git

Renomeando stage_area.md para 2-stage_area.md

git mv stage_area.md 2-stage_area.md

git status

```
Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git status
on branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    working_directory.md -> 1-working_directory.md
        renamed:    stage_area.md -> 2-stage_area.md
```

Movendo Arquivos

- Manualmente

Crie um novo diretório dentro de git-aula

mkdir sub

Mova o arquivo README manualmente para dentro do novo diretório

git status

```

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    README.md

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

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

```

- Com o git

Movendo o arquivo README para dentro do novo diretório sub	<code>git mv README.md sub/README.md</code>
--	---

git status

```

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:   README.md -> sub/README.md

```

***A maior diferença entre fazer manualmente e utilizando o GIT é que as alterações já ficam preparadas para o COMMIT (Stage Area).*

Desfazendo Mudanças

Desfazendo Modificações

Editar o arquivo repository_local.md	Antes: dsfsdff
	Depois:

```

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (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:   repository_local.md

```

Restaurando as alterações	<code>git restore repository_local.md</code>
---------------------------	--

Agora a linha que eu havia apagado do meu arquivo voltou:	Antes:
	Depois: dsfsdff

Trazendo de Volta do Staged

- No exemplo abaixo estamos restaurando um arquivo (teste-Trazendo-de-Volta-do-Staged.txt) que foi inserido no Stage Area:

Comando	git restore --staged teste-Trazendo-de-Volta-do-Staged.txt
<pre>Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master) \$ git status on branch master changes to be committed: (use "git restore --staged <file>..." to unstage) new file: teste-Trazendo-de-Volta-do-Staged.txt Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master) \$ git restore --staged teste-Trazendo-de-Volta-do-Staged.txt Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master) \$ git status on branch master Untracked files: (use "git add <file>..." to include in what will be committed) teste-Trazendo-de-Volta-do-Staged.txt nothing added to commit but untracked files present (use "git add" to track) Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master) \$</pre>	
Comando para restaurar todos os arquivos que estão na Stage Area	git restore --staged .
Comandos mais antigos	git reset HEAD teste-Trazendo-de-Volta-do-Staged.txt git reset HEAD .

Corrigindo o Último Commit (--amend)

- Alterando a mensagem do último commit:

<pre>Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master) \$ git log commit 3f1d8a305eb82338ba10607a5d96f5b61f64d107 (HEAD -> master) Author: Vanessa Ichikawa <vanessavallarini@gmail.com> Date: Sun Apr 11 14:37:30 2021 -0300 alter local readme commit ecf447bdeb9fcc233368d737b33ca3e08e4a1979 Author: Vanessa Ichikawa <vanessavallarini@gmail.com> Date: Sun Apr 11 13:59:34 2021 -0300 alter filename</pre>	
Comando para alterar a mensagem do último commit	git commit --amend -m 'insert readme to sub directory'

```

Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master)
$ git log
commit 1330bc34e121cdf8d58103114299d2335bab9653 (HEAD -> master)
Author: Vanessa Ichikawa <vanessavallarini@gmail.com>
Date: Sun Apr 11 14:37:30 2021 -0300

    insert readme to sub directory

commit ecf447bdeb9fcc233368d737b33ca3e08e4a1979
Author: Vanessa Ichikawa <vanessavallarini@gmail.com>
Date: Sun Apr 11 13:59:34 2021 -0300

    alter filename

```

Recuperando Arquivos

- Seguindo as boas práticas, é recomendado criar novos pontos na história ao invés de ficar fazendo alterações em commit (--amend);
- Puxando uma alteração feita em um arquivo:

Visualize o histórico de commit	git log
Copie o nome do commit que deseja restaurar	8c0ae56bf2179371a3c76b639e89bb577b3d7221
Trazendo determinado arquivo para o estado que estava no commit selecionado	git checkout 8c0ae56bf2179371a3c76b639e89bb577b3d7221 -- README.md
Fazendo um novo commit com o arquivo recuperado no estado desejado	git commit -m 'recovery README.md'

Removendo Arquivos não Rastreados

Crie dois arquivos para teste desta aula	touch trash1.txt trash2.txt
Removendo os arquivos	git clean -n ou git clean -f

***Essa remoção não permite recuperar os arquivos posteriormente, pois, os arquivos não estão em um ponto da história, estão em nosso Working Directory e são não rastreáveis**

Revertendo um Commit

- Reverter múltiplos arquivos de um determinado ponto da história:

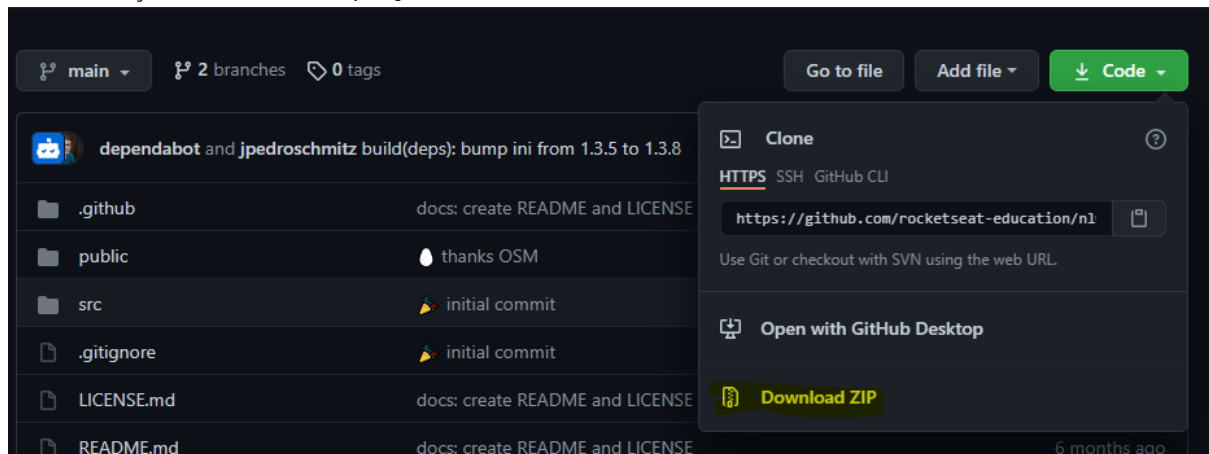
Garanta que seu working directory esteja limpo	git status
<pre> Vanessa@VanessaNote MINGW64 ~/Documents/git-aula (master) \$ git status On branch master nothing to commit, working tree clean </pre>	
Visualize o histórico de commit e selecione um (ponto da história).	8c0ae56bf2179371a3c76b639e89bb577b3d7221

A partir do commit mais recente o anterior é o -1, antes desse temos o -2 e assim por diante. O que eu escolhi é o -6	
Retornando para o ponto da história selecionado	git revert HEAD~6
<ul style="list-style-type: none"> • Outra forma de reverter: 	
Visualizar o log de forma resumida Escolha o que vai reverter e copie o hash	git log --oneline
Digite o seguinte comando para reverter para o commit selecionado	git revert 023680c <i>git revert +hash selecionado</i>

Usando Git em um Projeto Real

Adicionando e Verificando Alterações

- Acesse o seguinte link: <https://github.com/rocketseat-education/nlw-03-discovery>;
- Faça download do projeto:



- Cole a pasta do projeto no seu diretório Documents e edite o nome para nlw-03-discovery;
- Abra um terminal do Git Bash dentro da pasta nlw-03-discovery;
- Verifique se o projeto já não possui o diretório do git: git status. Se não tiver vai aparecer a seguinte mensagem: ***fatal: not a git repository (or any of the parent directories): .git***;
- Inicie o repositório do Git: git init;
- Adicione todos os arquivos do projeto para o Stage Área: git add .;
- Envie todos os arquivos da Stage Area para o Local Repository: git commit -m 'initial commit'.

Staging e Commits com atalho

- Abra o arquivo index.hbs, cujo se encontra dentro da pasta src/views e faça uma alteração. No exemplo abaixo apenas demos alguns ENTER e adicionamos a palavra “agora” (repare que o VS Code já percebeu que o arquivo está sendo rastreado pelo GIT ao exibir as barra em roxo mais claro):

- Dê o comando para visualizar se houve alterações: `git status`;
- Dê o comando para visualizar as alterações: `git diff`;
- Dê o comando para visualizar exatamente qual linha do código foi alterada: `git diff --color-words`;

***Esse símbolo @@ -30,7 +30,11 @@ representa em qual linha do código foi feita a alteração**

- Agora que todos os arquivos do projeto já estão rastreados, não precisamos dar o comando `git add`. + `git commit` para submeter as alterações, basta dar o seguinte comando: `git commit -am 'modify index'`.

Ver modificações em Diversos Pontos da História

- Visualizando as mudanças que já ocorreram no projeto: `git log`;
- Copie o hash de um ponto da história;
- Ver a mudança nesse ponto da história: `git show 8c7194d24`; (`git show` + hash do ponto da história selecionado)
- Obter o mesmo resultado do comando acima, porém, com as alterações em cores diferentes: `git show 8c7194d24 --color-words`;
- Obter o mesmo resultado do comando acima, porém, para um determinado arquivo: `git show 6296bcc545 -- src/views/index.hbs`;
- Obter o mesmo resultado do comando acima, porém, para uma determinada pasta e os arquivos dentro dela: `git show 6296bcc545 -- src/views/*`.

***Repare que o que está rodando por trás do `git show` é o `diff --git a/src/views/orphanages.hbs b/src/views/orphanages.hbs`, cujo está comparando o mesmo arquivo na versão a com a versão b.**

****Git diff: traz a diferença entre o `working directory` e `stage area`, enquanto o `git show` traz o que já tem no `Local Repository` em um determinado ponto da história.**

Ignorando Arquivos e Diretórios Indesejados

- Quando baixamos um projeto da internet alguns arquivos que vem nele não nos interessa, porém, existe um que é muito importante, o `.gitignore`;
- De o comando para visualizar os arquivos do projeto: `ls -al`;
- De o comando para visualizar o que contém no arquivo `.gitignore`: `cat .gitignore`;
- Esse arquivo ajuda a ignorar alguns arquivos que tem no nosso projeto, porém, não queremos adicionar em nosso `Local Repository`;
- Supondo que eu submeti um arquivo para o `Local Repository`, porém, este arquivo deveria ser ignorado. Podemos fazer os seguintes comandos:

Abra o terminal do Git na pasta do projeto <code>git-aula</code>	
Crie o arquivo <code>.gitignore</code>	<code>vim .gitignore</code>
Insira no arquivo <code>.gitignore</code> o que deve ser ignorado	<code>i.file2</code> (o arquivo que irei ignorar. i permite eu digitar, o nome do arquivo ficará <code>.file2</code>) <code>esc</code> (finalizar) <code>:wq</code> (salvar o que eu escrevi)
Adicionar o arquivo <code>.gitignore</code> na stage	<code>git add .gitignore</code>

area	
Fazer o commit do arquivo .gitignore no Local Repository	git commit -m 'add .gitignore'
Limpar tudo o que estava sendo rastreado pelo git	git rm -r --cached .
Adicionando todos os arquivos novamente para serem rastreados, porém, agora eu tenho o .gitignore, cujo está ignorando o arquivo file2.txt	git add .
Fazendo o commit da alteração	git commit -m 'ignore files'