

Controle de Versão

Aula 01

<Módulo 02/>

Controle de Versão com Git

Introdução



Nesta aula, vamos falar sobre o Git, um sistema de controle de versão que nos ajuda a rastrear mudanças em nossos arquivos, reverter para versões anteriores quando necessário e manter organizado o progresso de nossos projetos. Veremos como instalar, configurar e fazer uso básico do Git.

O que é um sistema de controle de versão ?

Imagine que você está escrevendo um guia extenso sobre fotografia digital, com o objetivo de cobrir tudo, desde o básico até técnicas avançadas.

Conforme você avança com seu guia, frequentemente revisa e reestrutura seu conteúdo, às vezes removendo seções inteiras ou mudando a ordem dos tópicos.

Ocasionalmente, percebe que uma maneira anterior de explicar um conceito era melhor, ou deseja trazer de volta uma seção que deletou alguns dias atrás.

Atualmente, você está criando cópias de backup manuais de seus arquivos antes de fazer grandes mudanças, mas esse método está começando a se tornar incômodo e desorganizado.

Você também está preocupado em perder a noção de qual versão do arquivo é a mais atual ou perder conteúdo importante no processo.

Você precisa de um sistema que permita rastrear mudanças em diferentes versões de seus documentos, reverter facilmente para versões anteriores quando necessário e manter organizado o progresso de seu guia.

Isso é o que faz um sistema de controle de versão.

O que é o Git ?

Ele é um sistema de controle de versão projetado para lidar com tudo, desde projetos pequenos até muito grandes com velocidade e eficiência.

O Git permite que você mantenha um histórico completo e detalhado de todas as mudanças que você fez em seus arquivos, não apenas permitindo que você volte para versões anteriores, mas também que entenda exatamente o que mudou de uma versão para outra.

Repositório

Para iniciar com o Git, você cria um 'repositório', que é uma pasta que contém todo o seu projeto e o histórico de suas revisões.

Este repositório pode ser mantido localmente em seu computador, mas também pode ser sincronizado com um serviço remoto como o GitHub, GitLab ou Bitbucket.

Veremos sobre serviços remotos em aulas futuras.

Commit

Quando você usa o Git, cada vez que "salva" as alterações no seu guia de fotografia digital, você está criando um 'commit'.

Pense em um 'commit' como um ponto de checagem seguro: se algo der errado no futuro, você pode voltar a esse ponto a qualquer momento. É como ter uma máquina do tempo para seus arquivos.

Além disso, o Git não apenas armazena o commit mais recente do seu projeto, mas também todos os commits anteriores. Então, se você deletar uma seção do guia e decidir que precisa dela de volta, é fácil recuperá-la.

Instalação do Git

Introdução

Antes de utilizar o Git, precisamos instalá-lo no computador.

Há diferentes formas de instalar o Git, dependendo do sistema operacional que você utiliza.

No Windows, existem duas abordagens principais: a instalação nativa, que envolve baixar e executar o instalador do Git para Windows, e a instalação via Windows Subsystem for Linux (WSL), que fornece um ambiente semelhante ao Linux dentro do Windows.

Por outro lado, no Linux e no macOS, o processo de instalação é geralmente mais direto.

Exploraremos os passos para a instalação do Git em cada um desses sistemas operacionais.

Se você é usuário do Windows, verá como pode escolher entre uma instalação mais tradicional ou optar pela abordagem do WSL, que é particularmente útil se você deseja uma experiência mais próxima do ambiente Linux. Recomendamos que opte pelo WSL.

Para usuários de Linux e macOS, apresentaremos um guia simplificado que se adapta à natureza desses sistemas operacionais.

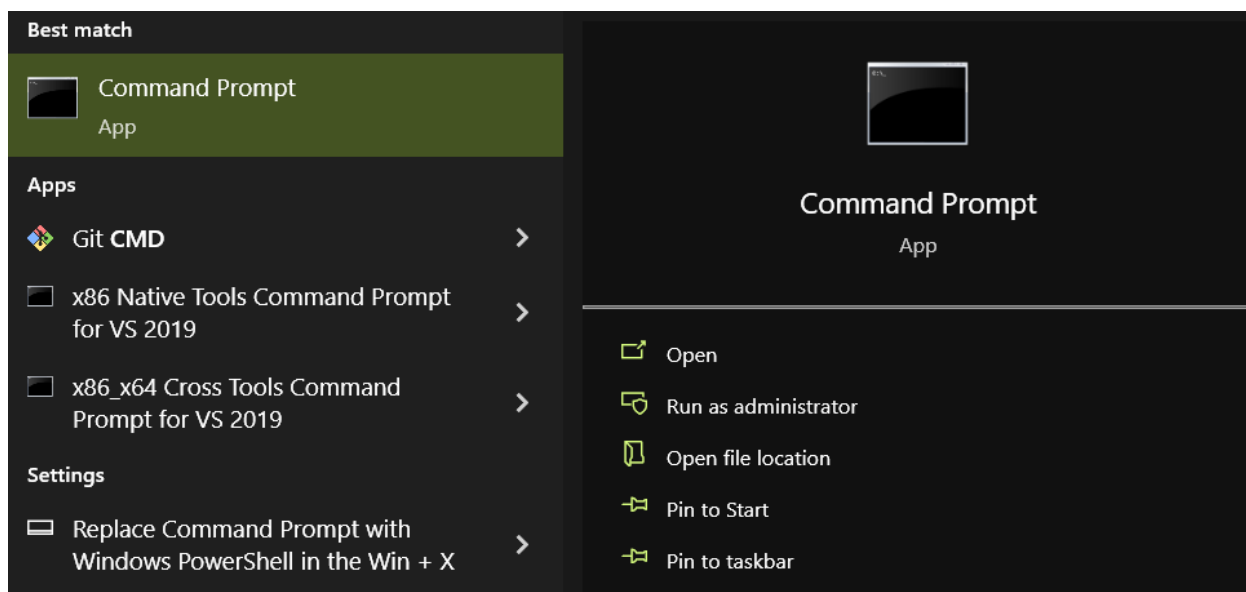
Instalação no Windows nativo

Baixe e execute o instalador para Windows no site oficial do Git: <https://git-scm.com/download/win>.

Recomendamos deixar todas as opções de instalação no seu valor padrão sugerido pelo instalador, exceto se você souber o que está fazendo.

Após a instalação, você pode verificar se o Git foi instalado corretamente abrindo um programa de terminal e executando o comando `git --version`, que vai mostrar a versão instalada do Git.

No Windows, existem dois terminais nativos: "cmd" (prompt de comando) e "powershell", e além deles o Git instala um terceiro chamado "Git Bash". Para abrir um deles, você pode digitar na pesquisa do Windows por um desses nomes:



Instalação do Git

Uma vez aberto, ao digitar `git --version` e apertar Enter no teclado, aparece a versão do Git, sinalizando que a instalação foi bem sucedida:

```
C:\> Command Prompt
Microsoft Windows [Version 10.0.19044.3086]
(c) Microsoft Corporation. All rights reserved.

E:\>git --version
git version 2.42.0.windows.2

E:\>_
```



O que é um terminal ?

Um terminal, também conhecido como terminal de linha de comando ou console, é uma interface de usuário que permite interagir com o sistema operacional ou software através de comandos de texto.

É como uma janela para o mundo digital, onde você pode digitar comandos específicos em vez de usar o mouse para clicar em ícones.

Imagine-o como uma conversa direta com o seu computador, onde você dá ordens e ele responde.

Essa ferramenta é incrivelmente poderosa para programadores e profissionais de TI. Com o terminal, você pode realizar tarefas como gerenciar arquivos, executar programas, acessar bancos de dados e muito mais.

Por exemplo, se você quer saber quais arquivos estão em uma pasta, em vez de abrir a pasta visualmente, você digita um comando no terminal e ele lista todos os arquivos para você.

Um ponto importante é que cada sistema operacional tem seu próprio tipo de terminal.

No Windows, é conhecido como Prompt de Comando ou PowerShell, enquanto no macOS e Linux, é chamado de Terminal.

Embora os comandos possam variar entre esses sistemas, a ideia básica é a mesma: você digita um comando de texto e o computador executa a tarefa correspondente.

O terminal é uma ferramenta fundamental para quem quer se aprofundar em programação e tecnologia da informação, pois oferece controle e flexibilidade muito maiores do que as interfaces gráficas.

É como ter um diálogo direto com as engrenagens que fazem seu computador funcionar.

Instalação do Git

Instalação no Windows via WSL

O WSL, ou Windows Subsystem for Linux, é uma funcionalidade do Windows 10 e Windows 11 que permite aos usuários executar um ambiente Linux diretamente no Windows, sem a necessidade de uma máquina virtual.

Isso significa que você terá dois “computadores” em um só: o Windows normal, e o WSL, que é um Linux.

Essa ferramenta é especialmente útil para desenvolvedores que precisam executar ferramentas, utilitários ou aplicações Linux no Windows.

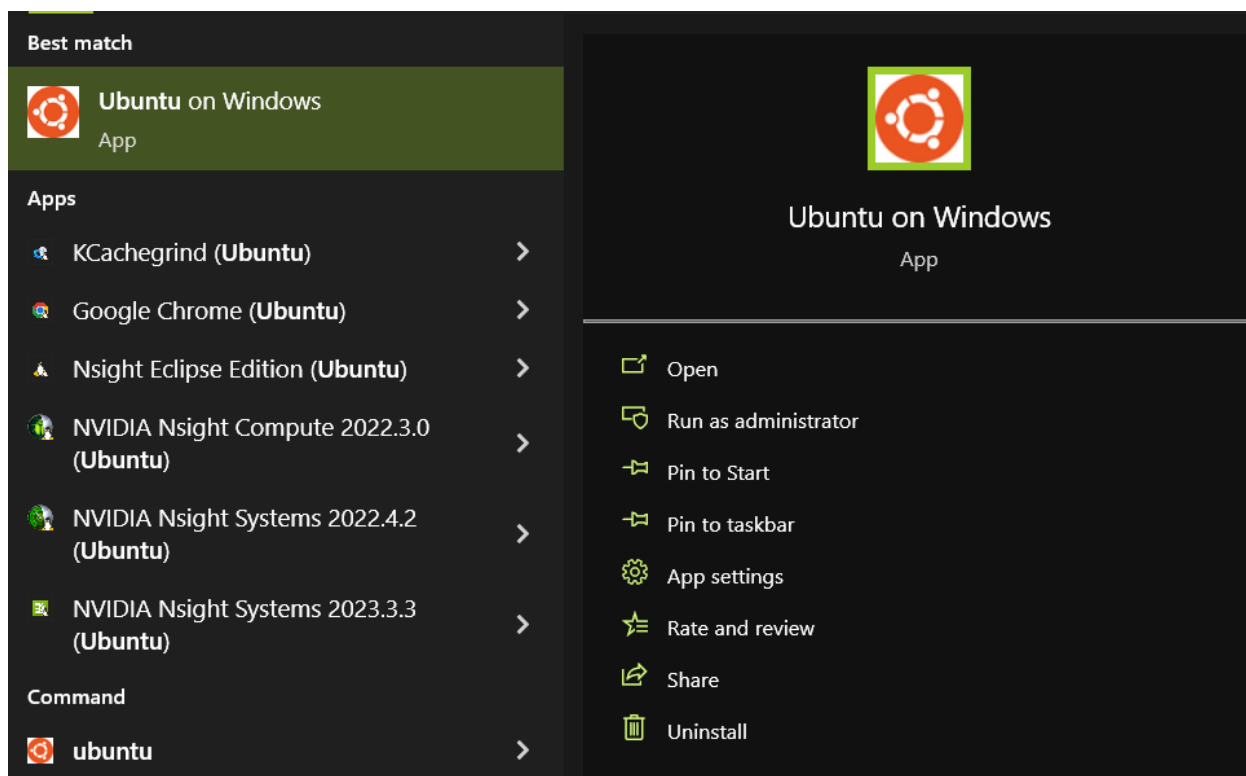
Para instalar o WSL, você pode seguir o passo a passo oficial da Microsoft: <https://learn.microsoft.com/en-us/windows/wsl/install>.

Em resumo, você precisará abrir um terminal em modo de administrador (basta digitar “cmd” na pesquisa do Windows e clicar em “Executar como Administrador”) e digitar o comando `wsl --install`.

Depois disso, reinicie o computador. Ao voltar, um terminal abrirá automaticamente pedindo para você criar um nome de usuário (username) e senha (password) para o ambiente Linux do WSL.

Por padrão, o WSL instala a distribuição Ubuntu do Linux, recomendamos que mantenha a distribuição padrão.

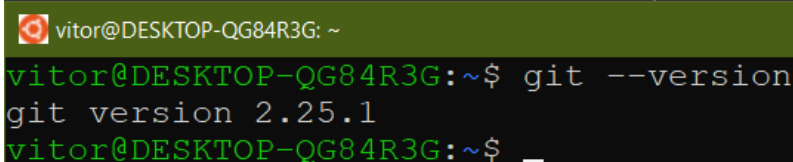
Nas próximas vezes que quiser abrir seu ambiente Linux, basta digitar “ubuntu” na pesquisa do Windows:



Uma vez que o WSL esteja funcionando, abra-o e execute `sudo apt update && sudo apt upgrade` para atualizar os pacotes do Linux. Depois, digite `sudo apt install git` para instalar o Git.

Instalação do Git

Para testar se a instalação foi bem sucedida, execute `git --version` no terminal do WSL. Se o Git foi instalado com sucesso, vai aparecer a versão dele:



```
vitor@DESKTOP-QG84R3G: ~  
vitor@DESKTOP-QG84R3G:~$ git --version  
git version 2.25.1  
vitor@DESKTOP-QG84R3G:~$
```



Qual forma de instalação no Windows devo escolher ?

Ao escolher entre instalar o Git de forma nativa no Windows ou dentro do Windows Subsystem for Linux (WSL), recomendamos a instalação no WSL pelos seguintes motivos:

1. **Padrão do Mercado:** A indústria de desenvolvimento de software frequentemente utiliza sistemas baseados em Linux. Ao usar o WSL, você ganha acesso a um ambiente Linux genuíno dentro do Windows. Isso significa que você pode trabalhar com as mesmas ferramentas, comandos e scripts que são comumente utilizados em ambientes de produção profissional. Assim, ao aprender e trabalhar dentro do WSL, você está se preparando melhor para os cenários encontrados no mercado de trabalho.
2. **Compatibilidade e Consistência:** Muitas ferramentas de desenvolvimento e scripts são projetados primariamente para Linux. Ao utilizar o WSL, você assegura uma maior compatibilidade com essas ferramentas, evitando problemas comuns de diferenças entre sistemas operacionais. Isso inclui não só o Git, mas também uma ampla gama de outras ferramentas de desenvolvimento.
3. **Ambientes Antigos vs. WSL:** Antes da introdução do WSL, os desenvolvedores no Windows recorriam a soluções como CMD, PowerShell ou Git Bash. Embora essas ferramentas ofereçam funcionalidades básicas, elas não proporcionam a experiência completa de um sistema Linux. O WSL, por outro lado, oferece uma experiência mais integrada e poderosa, trazendo o melhor do Linux para o Windows.
4. **Aprendizado e Crescimento Profissional:** Ao usar o WSL, você tem a oportunidade de se familiarizar com o Linux e seus comandos. Esta é uma habilidade valiosa, pois muitos servidores e ambientes de produção rodam em Linux. Ao adquirir experiência com Linux, você se torna mais versátil e adaptável como desenvolvedor.
5. **Flexibilidade e Desempenho:** O WSL proporciona uma integração suave com o Windows, permitindo que você utilize recursos do Linux e do Windows simultaneamente. Além disso, o desempenho do WSL tem melhorado constantemente, tornando-o uma escolha eficiente para o desenvolvimento de software.

Em resumo, optar pelo WSL com o Git instalado nele é uma escolha estratégica que alinha seu ambiente de desenvolvimento com as práticas padrão da indústria, melhora a compatibilidade com uma ampla gama de ferramentas, e contribui para seu crescimento e adaptabilidade como profissional de tecnologia da informação.

Instalação do Git

Instalação no Linux

Dependendo da sua distribuição de Linux, o processo de instalação do Git pode ser diferente. Siga as instruções do site oficial do Git: <https://git-scm.com/download/linux>.

O site tem instruções para instalar o Git em distribuições baseadas em Debian, Ubuntu, Fedora, Gentoo, Arch Linux e outras.

Para testar se a instalação foi bem sucedida, abra um terminal e execute `git --version`. Se o Git foi instalado com sucesso, vai aparecer sua versão.

Instalação no macOS

Siga as instruções do site oficial do Git: <https://git-scm.com/download/mac>.

Para testar se a instalação foi bem sucedida, abra um terminal e execute `git --version`. Se o Git foi instalado com sucesso, vai aparecer sua versão.

Configuração inicial do Git

Introdução

A primeira etapa da jornada com o Git envolve sua configuração inicial, um processo simples, mas fundamental para garantir que seu trabalho seja adequadamente rastreado e gerenciado.

Esta introdução visa orientá-lo através dos passos iniciais, focando nas configurações essenciais que todo usuário do Git deve conhecer e aplicar.

Nome e email

Futuramente, quando você já estiver usando o Git cotidianamente para criar "checkpoints" (pontos de salvamento) dos seus projetos, em cada checkpoint ficará registrado seu nome e email.

Por isso, logo após instalar o Git, você precisa configurar essas informações com os seguintes comandos:

- `git config --global user.name "<seu nome aqui>"`
- `git config --global user.email "<seu email aqui>"`

Substituindo o que está entre <> por seu nome e email reais, por exemplo:

- `git config --global user.name "Alice Smith"`
- `git config --global user.email "alice@gmail.com"`

Editor de texto

O Git usa um editor de texto para certos propósitos.

Em geral é um programa que funciona como o Bloco de Notas no Windows, porém dentro de um terminal, sem uma janela separada.

Podemos configurar qual programa editor de texto o Git usará, com o comando `git config --global core.editor <nome do editor>`.

Se você usa um computador Windows sem WSL, recomendamos que defina o editor de texto padrão para notepad, que é o nome do Bloco de Notas.

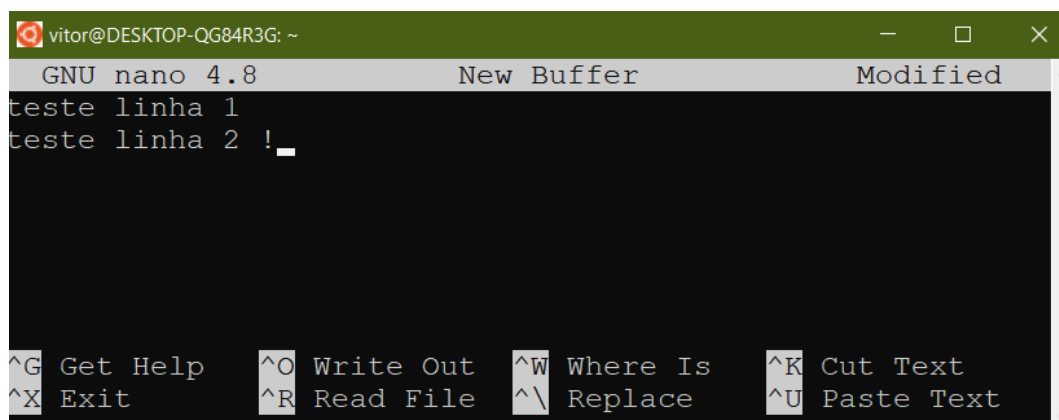
Já se você usa um computador Linux, macOS, ou WSL no Windows, sugerimos que defina o editor de texto padrão para nano, que é fácil de usar para principiantes:

- `git config --global core.editor nano`

Você precisa ter o programa nano instalado. Para isso, execute `sudo apt install nano` caso use Linux Ubuntu (incluindo WSL), `brew install nano` caso use macOS, ou o equivalente em outros sistemas Linux, a depender do gerenciador de pacotes do sistema.

Para testar se o programa nano está funcionando, digite `nano` no terminal e aperte Enter.

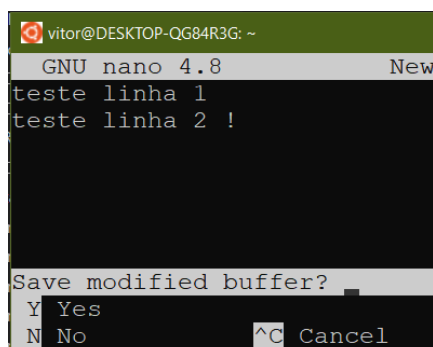
Se tudo estiver certo, o editor será iniciado e você poderá digitar o que quiser:



```
vitor@DESKTOP-QG84R3G: ~
GNU nano 4.8      New Buffer      Modified
teste linha 1
teste linha 2 !
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text
```


Configuração inicial do Git

Conforme os textos de ajuda na parte inferior do editor, aperte Ctrl+X para sair. Vai aparecer uma mensagem perguntando se você quer salvar o que digitou:



```
vitor@DESKTOP-QG84R3G: ~  
GNU nano 4.8 New  
teste linha 1  
teste linha 2 !  
  
Save modified buffer?  
Y Yes  
N No ^C Cancel
```

Aperte N para "não". Se você apertasse Y (sim), ele pediria para definir um nome ao arquivo que será salvo, você teria que digitar o nome e apertar Enter.

Outras configurações

Além das configurações básicas, o Git oferece uma ampla gama de opções de configuração para personalizar sua experiência de trabalho.

Para quem está começando, é importante saber que essas configurações ajudam a tornar o processo mais adaptado às suas necessidades.

Não se preocupe em entender exatamente o que todas elas fazem agora, pois muitos conceitos ainda serão vistos.

Algumas das configurações mais comuns incluem:

- **Alias:** Você pode criar atalhos para comandos longos. Por exemplo, configurar um alias para um comando frequente pode economizar tempo e esforço.
- **Color UI:** Esta opção ativa ou desativa o uso de cores na interface do Git, tornando mais fácil identificar diferentes tipos de informações, como mudanças, estados, etc.
- **Merge Tool:** Define a ferramenta padrão usada para resolver conflitos de mesclagem. Isso é especialmente útil em projetos grandes onde conflitos são comuns.
- **Push Default:** Configura o comportamento padrão do comando `git push`, determinando para qual branch remoto suas mudanças serão enviadas.
- **Pull Rebase:** Esta configuração define se o Git deve fazer um rebase (reaplicar commits em cima de outra base) ao invés de um merge (mesclar) quando fazendo um `pull`. Isso pode ajudar a manter um histórico de commits mais limpo.
- **FileModes:** Define se o Git deve ou não considerar as mudanças nas permissões de arquivos.
- **Ignore Case:** Útil em sistemas que não diferenciam maiúsculas de minúsculas, essa configuração diz ao Git como tratar o caso dos nomes de arquivo.
- **AutoCrlf:** Converte automaticamente as terminações de linha LF (usadas em sistemas UNIX) para CRLF (usadas em Windows) e vice-versa. Isso é importante para evitar problemas em projetos que são desenvolvidos em diferentes sistemas operacionais.

Configuração inicial do Git

Checando as configurações

Você pode verificar se as configurações foram salvas com o comando `git config --list`. Isso vai listar todas as configurações que você já definiu:

```
vitor@DESKTOP-QG84R3G:~$ git config --list
user.name=megatron0000
user.email=
```

Se quiser consultar o valor de uma configuração específica, digite `git config <nome da configuração>`, por exemplo:

```
vitor@DESKTOP-QG84R3G:~$ git config user.name
megatron0000
```

Configurações globais e locais

Ao usar o comando `git config --global`, você está definindo valores globais para as configurações do Git.

Isso significa que esses valores serão aplicados a todos os seus projetos, independentemente de onde eles estejam.

Por outro lado, pode ser que em projetos específicos você não queira usar o valor global de uma ou mais configurações.

Nesse caso, dentro da pasta do projeto, você poderá usar `git config` (sem a parte `--global`) para redefinir qualquer configuração.

Isso é chamado de configuração local e fará com que, dentro desse projeto, valha a configuração local que você criou em detrimento da global.

E em outros projetos, como não há configuração local (exceto se você criar com `git config`), valerá a global.

Fluxo de trabalho básico com Git

Introdução

Vamos manter o exemplo do guia de fotografia digital e mostrar como você cria seu projeto e usa o Git para fazer checkpoints.

Assumimos que você está usando o sistema operacional Linux, macOS, ou WSL no Windows com distribuição Ubuntu.

Criando uma pasta para o projeto

Primeiro precisamos criar uma pasta para abrigar o projeto do guia sobre fotografia digital.

Você pode criar uma pasta pelo terminal usando o comando `mkdir <nome da pasta>`, por exemplo `mkdir guia-fotografia`.

O comando `mkdir` significa "make directory"/"criar diretório", onde diretório é sinônimo de pasta.

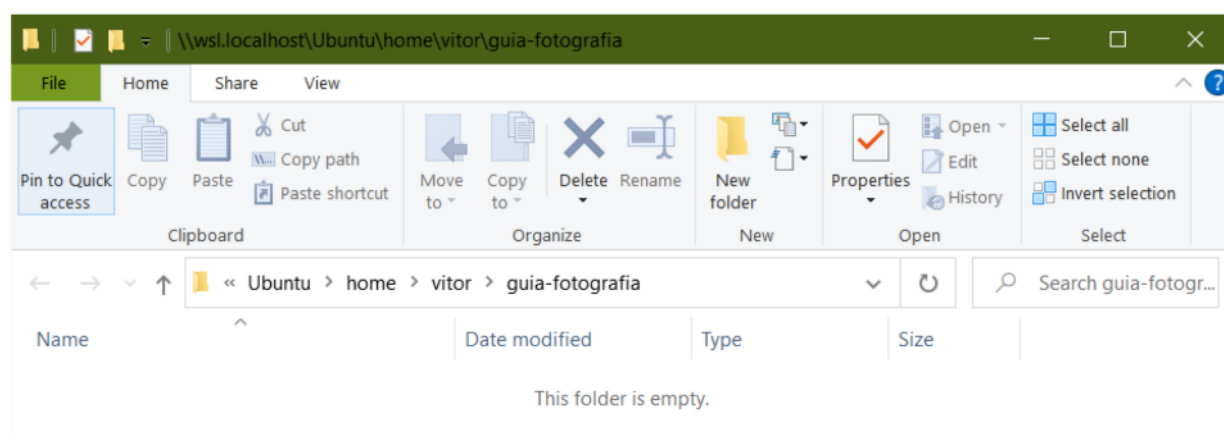
Se o nome da sua pasta tem espaços (não recomendado), precisará de aspas em torno do nome, por exemplo `mkdir "guia fotografia"`.

Depois, pode entrar na pasta usando o comando `cd <nome da pasta>`, por exemplo `cd guia-fotografia`. De novo precisará de aspas caso haja espaços no nome da pasta.

O comando `cd` significa "change directory"/"mudar diretório".

Caso você esteja usando Windows com WSL, poderá abrir a pasta com a interface gráfica familiar do Windows, o Explorador de Arquivos, digitando no terminal `explorer.exe .`, onde o ponto é uma abreviação para o diretório atual. Veja:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ explorer.exe .
```



Se estiver usando um sistema Linux ou macOS, também é possível abrir a interface gráfica nativa desses sistemas.

Fluxo de trabalho básico com Git

Git init

Agora que você está na pasta do projeto, pode inicializar o repositório Git com o comando `git init`.

Isso vai criar uma pasta oculta chamada `.git`, que é onde o git armazena todos os commits (backups) que você fará no futuro.

Para enxergar que foi criada a pasta oculta pelo terminal, digite `ls -a`

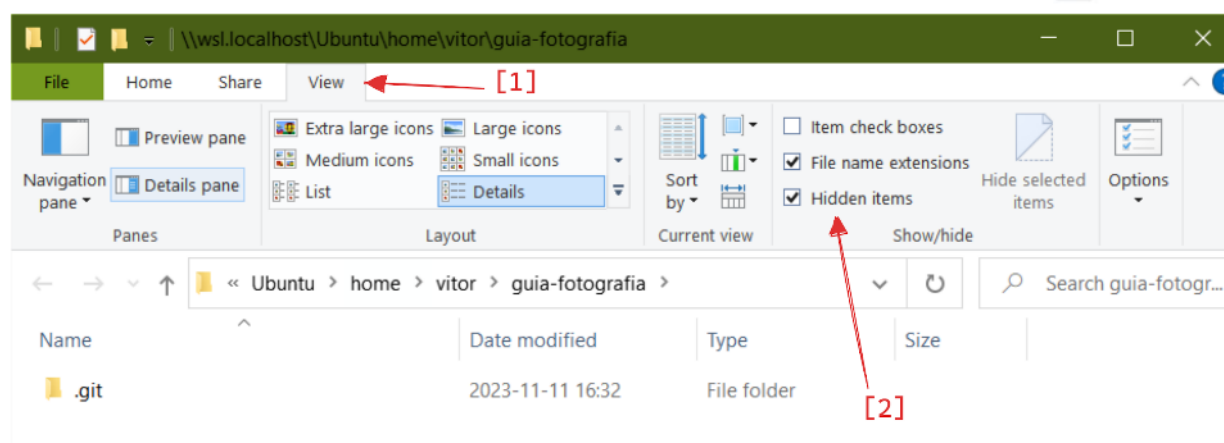
Esse é o comando de listagem de arquivos, ele mostra todos os arquivos e pastas que estão dentro da pasta atual (guia-fotografia).

O `-a` serve para incluir na listagem os arquivos e pastas ocultos, que de outro modo não seriam mostrados.

Veja como fica abaixo (os símbolos `.` e `..` não são arquivos propriamente ditos, eles representam a pasta atual e a pasta acima da atual, respectivamente):

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ ls -a
.  ..  .git
```

No Explorador de Arquivos do Windows também é possível ver a pasta oculta. Se você não está vendo, vá em "View"/"Visualização" e marque a opção "Hidden items"/"Ítems ocultos":



No nosso exemplo, a pasta do projeto estava vazia antes do `git init`.

Se você já tinha uma pasta com arquivos dentro e depois resolveu usar o Git nela, não tem problema.

O `git init` ainda vai funcionar. Ele vai somente criar a pasta oculta `.git` sem alterar nenhum dos arquivos ou pastas existentes.

Fluxo de trabalho básico com Git

O que é a pasta .git ?

A pasta .git é a “cozinha” do Git, onde ele armazena todas as informações de backup do seu projeto.

Nunca mexa nessa pasta, você pode quebrar o histórico do Git !

Mesmo se tentar abri-la para ver o que tem lá dentro, não entenderá muita coisa, porque o Git armazena informações de maneira que é fácil para a máquina compreender, mas difícil para humanos.

Nos próximos passos, ao executar certos comandos como `git add` e `git commit`, o Git vai alterar o conteúdo dessa pasta oculta automaticamente.

Working Tree

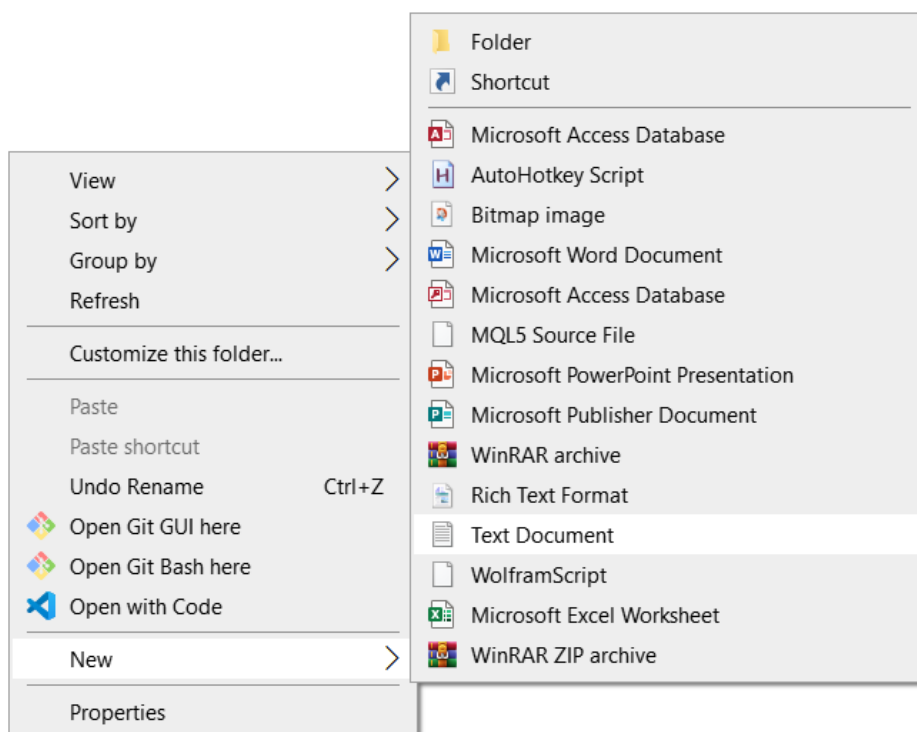
Suponha que, para iniciar seu guia de fotografia, você decide escrever um arquivo `guia.txt`.

Isso pode ser feito tanto pelo terminal quanto pela interface gráfica da pasta.

Pelo terminal, o comando `touch guia.txt` cria o arquivo, daí `nano guia.txt` abre o arquivo para edição, como já vimos.

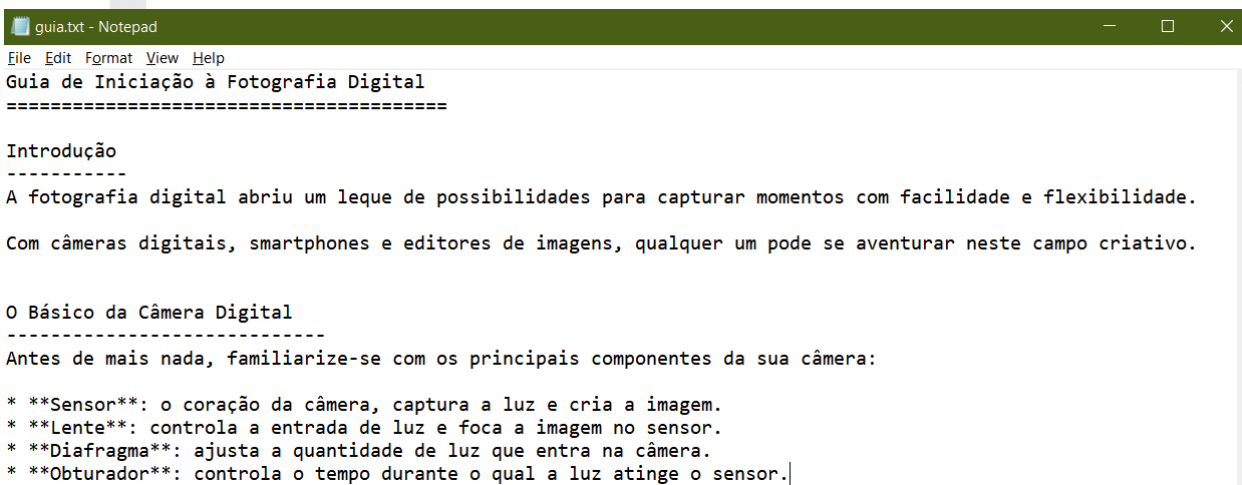
Pela interface gráfica, o procedimento varia de acordo com o sistema: Linux, macOS ou Windows.

No caso do Windows, você pode clicar com o botão direito do mouse, então ir em “New”/“Novo”, e então em “Text Document”/“Documento de Texto”:



Fluxo de trabalho básico com Git

Seja como for, imagine que escreveu o seguinte conteúdo no arquivo, e o salvou:



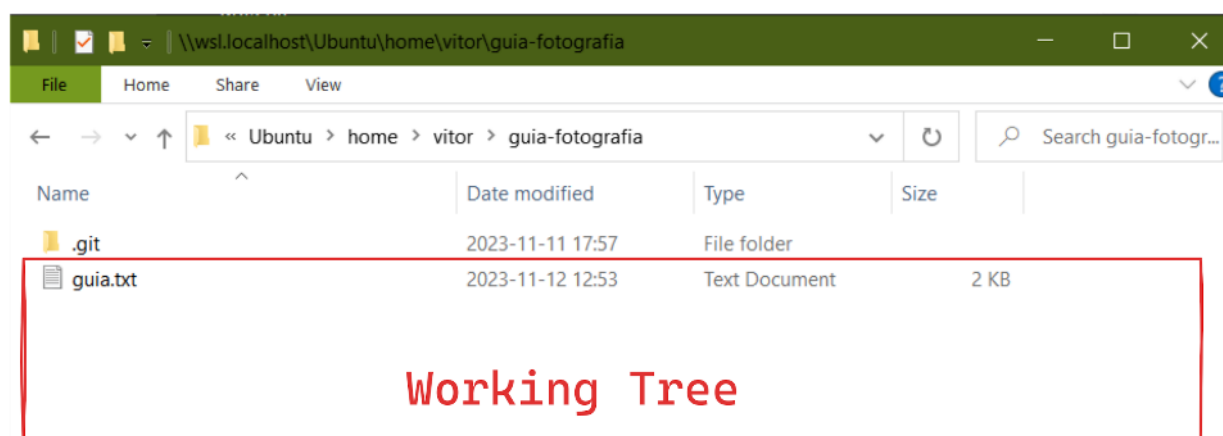
```
guia.txt - Notepad
File Edit Format View Help
Guia de Iniciação à Fotografia Digital
=====

Introdução
-----
A fotografia digital abriu um leque de possibilidades para capturar momentos com facilidade e flexibilidade.
Com câmeras digitais, smartphones e editores de imagens, qualquer um pode se aventurar neste campo criativo.

O Básico da Câmera Digital
-----
Antes de mais nada, familiarize-se com os principais componentes da sua câmera:

* **Sensor**: o coração da câmera, captura a luz e cria a imagem.
* **Lente**: controla a entrada de luz e foca a imagem no sensor.
* **Diafragma**: ajusta a quantidade de luz que entra na câmera.
* **Obturador**: controla o tempo durante o qual a luz atinge o sensor.
```

O arquivo fica salvo na pasta do projeto. Todos os arquivos e pastas fora de `.git` são chamados de **Working Tree** (árvore de trabalho):



Fluxo de trabalho básico com Git

Git status

Agora que o arquivo do guia foi criado, o Git percebe que o arquivo existe.

Para verificar o que o Git está enxergando no projeto, você pode usar o comando `git status`:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master

No commits yet

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

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

A mensagem diz "No commits yet" porque nunca criamos um commit (backup) do projeto até agora.

E o `guia.txt` aparece dentro de "Untracked files" em vermelho. Isso significa que o Git percebe que o arquivo é novo e não está sendo rastreado ainda.

Git add

Para rastrear o arquivo com o Git, você precisa usar o comando `git add <nome do arquivo>`. Neste caso `git add guia.txt`.

Depois disso, o `git status` passa a mostrar outra mensagem:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   guia.txt
```

Agora o `guia.txt` está dentro de "Changes to be committed" em verde.

Isso significa que ele foi rastreado pelo Git. Mas ainda não temos nenhum commit (backup) criado.

Fluxo de trabalho básico com Git

Index / Staging Area

Suponha que você resolveu adicionar mais uma seção ao `guia.txt` chamada "Modos de Disparo" no final do arquivo:

```
guia.txt - Notepad
File Edit Format View Help
Guia de Iniciação à Fotografia Digital
=====

Introdução
-----
A fotografia digital abriu um leque de possibilidades para capturar momentos com facilidade e flexibilidade.
Com câmeras digitais, smartphones e editores de imagens, qualquer um pode se aventurar neste campo criativo.

O Básico da Câmera Digital
-----
Antes de mais nada, familiarize-se com os principais componentes da sua câmera:

* **Sensor**: o coração da câmera, captura a luz e cria a imagem.
* **Lente**: controla a entrada de luz e foca a imagem no sensor.
* **Diafragma**: ajusta a quantidade de luz que entra na câmera.
* **Obturador**: controla o tempo durante o qual a luz atinge o sensor.

Modos de Disparo
-----
Explore os modos de disparo para entender melhor o controle criativo:

* **Automático**: a câmera ajusta todas as configurações.
* **Manual**: controle total sobre exposição, abertura, e velocidade do obturador.
* **Prioridade de Abertura**: você escolhe a abertura, a câmera ajusta a velocidade do obturador.
* **Prioridade de Velocidade**: você define a velocidade do obturador, a câmera cuida da abertura.
```

Se executar `git status` agora, verá o seguinte:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   guia.txt

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:   guia.txt
```

Estranho ! Agora o `guia.txt` está tanto em "Changes to be committed" em verde, quanto numa nova seção "Changes not staged for commit" em vermelho. Por quê ?

Precisamos entender exatamente o que acontece quando você executa `git add`. Vamos recorrer a uma metáfora:

Fluxo de trabalho básico com Git

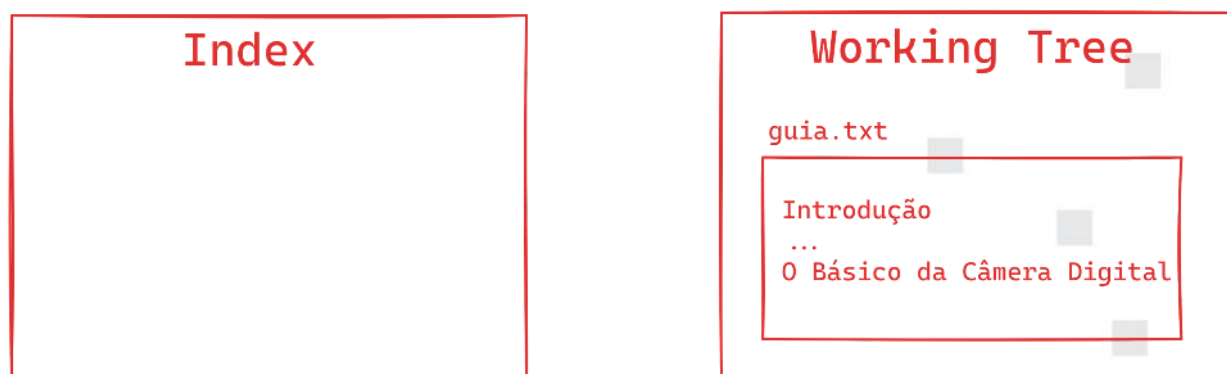
Se você está escrevendo um livro no computador, seu rascunho privado das páginas é equivalente ao Working Tree no Git.

Quando você acha que o livro está pronto para ser publicado, imprime uma cópia dele e dá para algumas pessoas lerem e checar se está bem escrito. No Git, essa "cópia impressa" é chamada de Index, ou Staging Area. Quando você executa `git add`, o Git manda uma cópia do arquivo como está para o Index.

Finalmente, quando você confirma que a cópia impressa não tem erros e está adequada para publicação, você manda para uma editora, e aí o livro adquire sua forma final, com uma encadernação bonita, uma capa, etc. No git, esse é o commit (que ainda vamos tratar), o commit é criado a partir do Index.

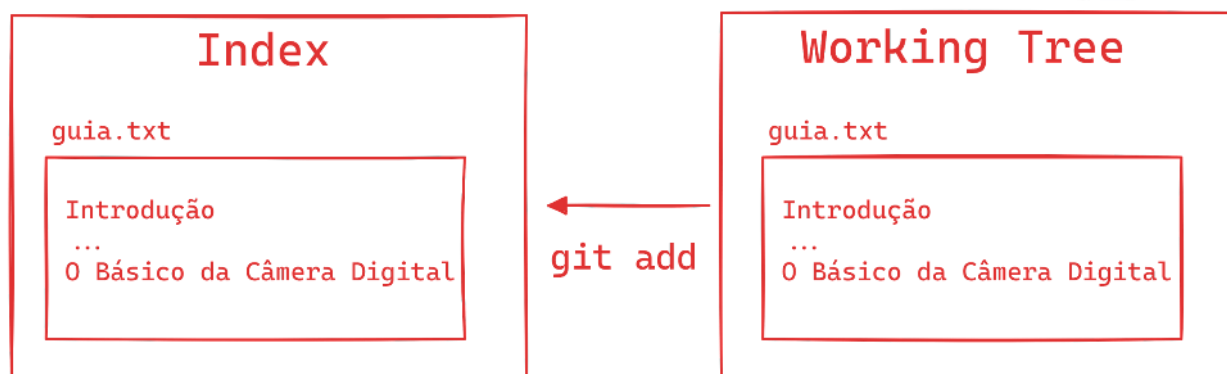
Então vamos recapitular o que fizemos até agora, na luz desta metáfora:

Quando você criou o `guia.txt` pela primeira vez, antes do primeiro `git add`, o Git estava assim:



O Index estava vazio, e o Working Tree tinha a primeira versão do arquivo. Por isso o `git status` mostrava "Untracked File", já que o arquivo não estava no Index.

Ao executar `git add`, o que ocorreu foi:



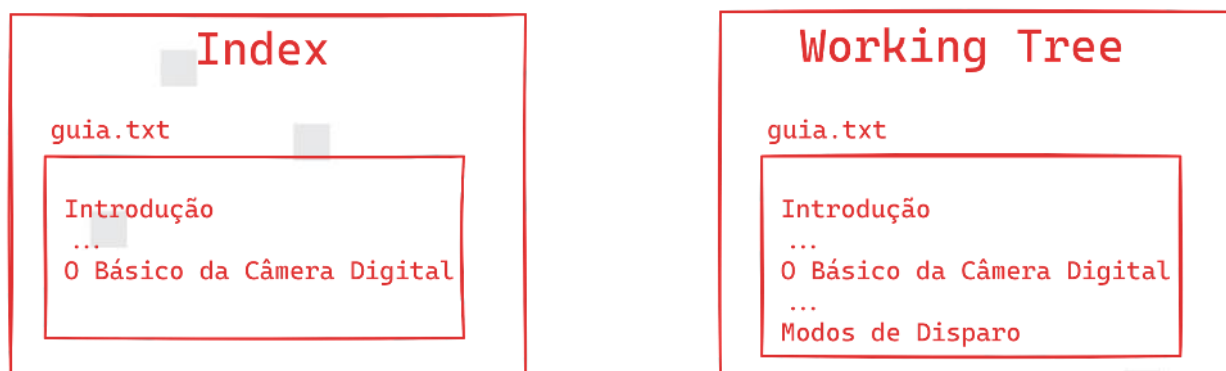
Ou seja, o Git copiou o arquivo do Working Tree para o Index. Por isso o `git status` passou a mostrar "Changes to be committed" em verde, já que o Index tinha a versão mais recente do arquivo.



Você já sabe que a Working Tree são seus arquivos como vistos no explorador de arquivos. Já o Index fica dentro da pasta `.git`.

Fluxo de trabalho básico com Git

Depois, quando você editou o `guia.txt` para inserir a nova seção "Modos de Disparo" no final, o Git ficou assim:

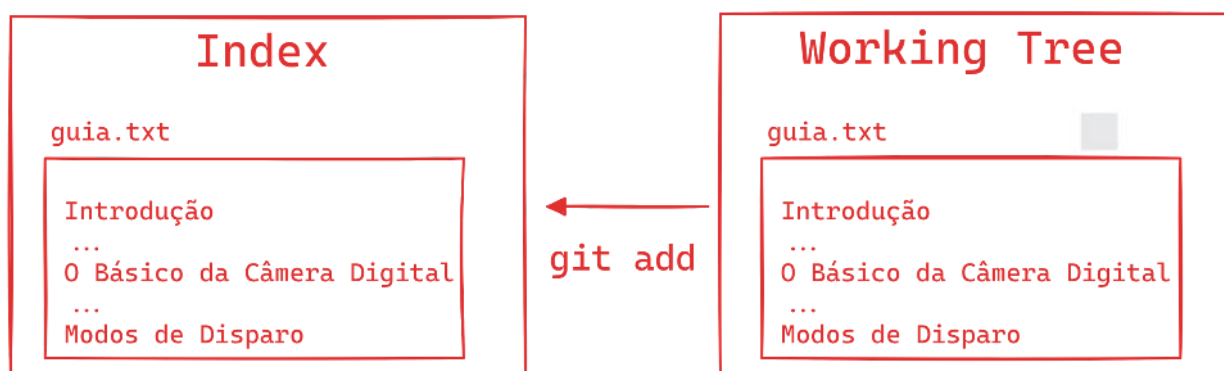


O Index não tem mais a versão mais recente do arquivo. Foi nesse momento que o `git status` passou a mostrar tanto em vermelho quanto em verde.

O verde é a versão que está no Index. E o vermelho é a versão que está na Working Tree mas não no Index.

O `git status` intitula a parte vermelha de "Changes not staged for commit" (mudanças não preparadas para commit) justamente porque são as mudanças que não estão no Index.

Para atualizar o Index, basta executar `git add guia.txt` de novo:



E agora o `git status` vai mostrar tudo em verde de novo:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   guia.txt
```

Só falta fazer o commit para guardar em definitivo uma cópia do projeto !

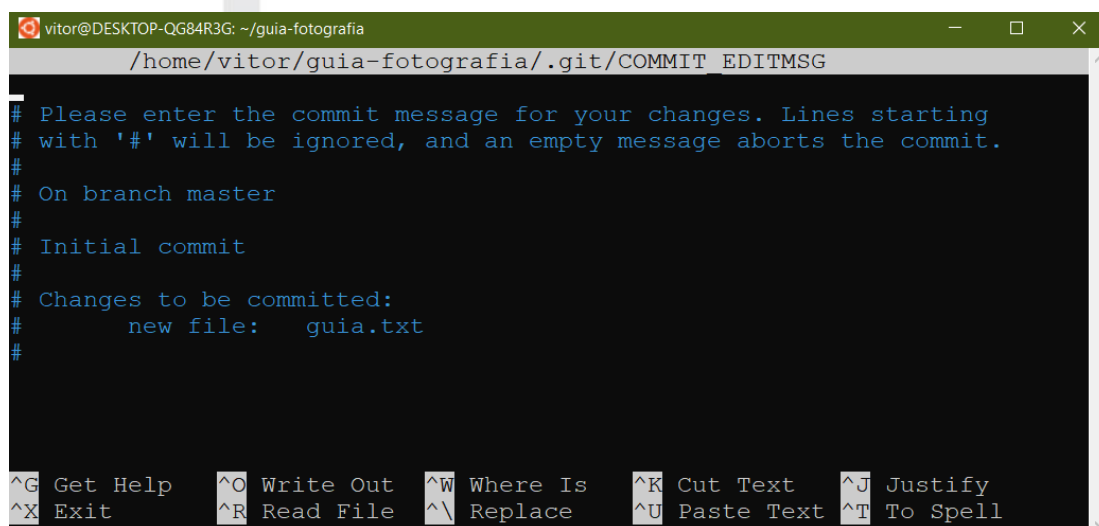
Fluxo de trabalho básico com Git

Git commit

Para criar um backup definitivo do projeto, você precisa usar o comando `git commit`.

Ele vai salvar uma cópia do Index como backup do projeto, não uma cópia da Working Tree ! Tudo que você quer salvar precisa ser antes adicionado ao Index com `git add` !

Ao digitar `git commit` e apertar Enter, o Git vai abrir o editor de texto configurado (lembrando que configuramos o nano):



```
vitor@DESKTOP-QG84R3G: ~/guia-fotografia
/home/vitor/guia-fotografia/.git/COMMIT_EDITMSG
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:   guia.txt
#
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell
```

Conforme as instruções escritas no editor, digite um título para o commit, por exemplo "Initial commit". É prática de mercado escrever o título em inglês, apesar de que o Git não se importa sobre o conteúdo do título.

Depois, pressione `Ctrl+X` para sair, `Y` para confirmar que você quer salvar, e então Enter (sem mudar o nome de arquivo gerado pelo Git).

Agora o editor vai fechar, e foi criado o primeiro commit:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git commit
[master (root-commit) aef0f34] Initial commit
1 file changed, 27 insertions(+)
create mode 100644 guia.txt
```

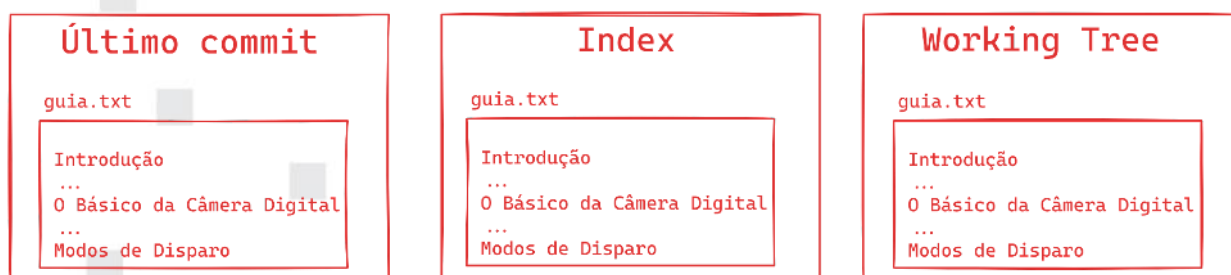


Se você não quiser abrir um editor de texto para digitar a mensagem de commit, pode optar pelo comando `git commit -m "<mensagem do commit>"`.

Por exemplo `git commit -m "Initial commit"`.

Fluxo de trabalho básico com Git

Após o commit, o Git fica esquematicamente assim:



Ou seja, o `git commit` cria um commit, sem alterar o Index ou a Working Tree.

O `git status` agora não mostra nenhum arquivo, porque o conteúdo do Index é idêntico ao commit, então não tem nada para ser salvo:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
nothing to commit, working tree clean
```

Ele diz "nothing to commit" porque o conteúdo do Index é idêntico ao último commit.

E diz "working tree clean" porque o conteúdo da Working Tree é idêntico ao Index.



No Git, quando o conteúdo do Index ou da Working Tree (ou dos dois) é diferente do conteúdo do último commit, dizemos que você tem "*mudanças locais*".



O "commit" é como salvar um jogo de videogame.

Ele salva o estado atual do jogo, incluindo o que você fez.

Você pode salvar o jogo em qualquer momento, e se algo der errado, pode voltar para o estado salvo (ainda veremos como faz isso em outras aulas).

O git salva todos os commits que você fizer (não só o mais recente), e você pode restaurar o projeto para qualquer um deles.

Eles ficam salvos dentro da pasta `.git`, assim como o Index.

Fluxo de trabalho básico com Git



Boas práticas para mensagens de commit

As boas práticas para mensagens de commit no desenvolvimento de software são cruciais para manter um histórico de mudanças claro e útil. Aqui estão algumas diretrizes:

1. **Seja Claro e Conciso:** A mensagem de commit deve ser suficientemente informativa para que alguém que a leia entenda rapidamente o que foi alterado e por quê.
2. **Escreva em Inglês:** Escrever mensagens de commit em inglês facilita a compreensão e a colaboração entre desenvolvedores de diferentes partes do mundo.
3. **Use o Tempo Presente:** É comum usar o tempo presente imperativo, como "Add", "Correct", "Update". Isso cria um padrão consistente e lê como uma instrução sobre o que o commit faz.
4. **Primeira Linha como um Resumo:** A primeira linha da mensagem deve ser um resumo curto (idealmente menos de 50 caracteres) do commit. Detalhes adicionais devem ser fornecidos após uma linha em branco.
5. **Separe Assuntos com Linhas em Branco:** Se a mensagem de commit contém múltiplos parágrafos, separá-los com uma linha em branco melhora a legibilidade.
6. **Explique o Contexto e a Razão, Não Apenas a Mudança:** Além de descrever o que foi alterado, é útil explicar por que essas mudanças foram feitas. Isso é particularmente importante para mudanças complexas ou não óbvias.
7. **Evite Mensagens Genéricas:** Evite mensagens vagas como "Correção de bugs" ou "Atualizações". Elas não oferecem informações úteis sobre o conteúdo do commit.
8. **Revise Antes de Commitar:** Antes de finalizar o commit, revise a mensagem para erros de digitação, clareza e aderência às diretrizes da equipe.
9. **Consistência com as Normas da Equipe:** Cada equipe ou projeto pode ter suas próprias convenções para mensagens de commit. É importante seguir essas normas para manter a consistência.

Fluxo de trabalho básico com Git

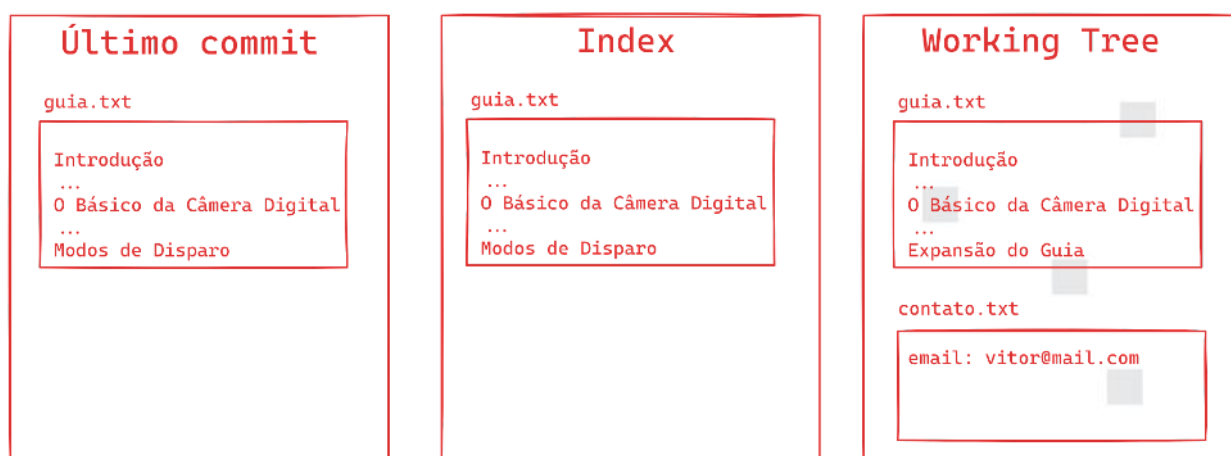
Fazendo mais commits

Agora que você criou o primeiro commit, vamos criar mais um para reforçar a maneira como o Git funciona.

Logo após o primeiro commit, suponha que você fez as seguintes três modificações:

- Removeu a seção "Modos de Disparo" que estava no final do `guia.txt`, porque estava muito grande, e você achou que seu público não teria interesse.
- Adicionou uma seção "Expansão do Guia" ao final do `guia.txt`.
- Adicionou um novo arquivo `contato.txt` contendo suas informações de contato.

Com isso, o Git ficou diagramaticamente assim:



Como o `guia.txt` está diferente do último commit, e essas mudanças não estão no Index, ele será classificado como "Changes not staged for commit" pelo `git status`.

E como o `contato.txt` não existe no Index, será classificado como "Untracked files" pelo `git status`.

Vamos comprovar isso:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ 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:   guia.txt

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

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

Fluxo de trabalho básico com Git

Se usarmos `git add guia.txt` e `git add contato.txt`, o diagrama passará a ser:



Há outras maneiras alternativas de adicionar os dois arquivos:

- `git add guia.txt contato.txt` adiciona os dois de uma só vez
- `git add .` adiciona todos os arquivos que estiverem presentes. O `.` significa "todos os arquivos"

Como as duas mudanças foram copiadas ao Index, elas serão classificadas como "Changes to be committed" pelo `git status`:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   contato.txt
    modified:   guia.txt
```

Fluxo de trabalho básico com Git

Ao fazer `git commit -m "Add guide expansion section and contact information"`, o Git vai criar um novo commit, e o diagrama ficará assim:



De novo o `git status` voltará a estar vazio.

Note também que o primeiro commit não foi perdido. o Git guarda todos os commits passados.

Um resumo, o processo de trabalhar no seu projeto de fotografia digital é um ciclo contínuo de edição, adição e commit.

Lembre-se: cada commit é um marco importante a cada vez que você faz mudanças significativas, para que possa sempre reverter seu projeto para uma versão anterior se necessário.

Fluxo de trabalho básico com Git

Git log

Onde estão seus commits já feitos ? Você pode usar o comando `git log` para vê-los:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git log
commit 503517dcadbccd0255110bdald4f7c6014a45ef7 (HEAD -> master)
Author: megatron0000 <[redacted]>
Date:   Sun Nov 12 16:03:08 2023 -0400

    Add guide expansion section and contact information

commit aef0f34ecdd92ae2e722f8137e61c1c1f94e6bc7
Author: megatron0000 <[redacted]>
Date:   Sun Nov 12 14:17:15 2023 -0400

    Initial commit
```

O `git log` mostra todos os commits que você fez, do mais recente para o mais antigo.

Em cada commit, aparecem:

- Autor: nome e email, por isso que tivemos que configurar nome e email no início. Na figura, encobrimos o email
- Data: quando o commit foi feito
- Mensagem de commit: o que você escreveu no editor de texto, ou no `-m`, quando fez o commit
- Um código chamado "commit hash" ou "checksum", por exemplo 503517... no caso do nosso commit mais recente na figura. Esse código é um identificador único para o commit, e você pode usá-lo para restaurar o commit mais tarde (ainda veremos sobre isso em outras aulas)

Na figura também apareceram as palavras `HEAD` e `master`. Não se preocupe com elas por enquanto, ainda introduziremos o que elas são.

Por enquanto, entenda-as como sinônimos de "último commit" (commit mais recente).

O comando `git log` tem várias opções que mudam quais commits são mostrados e como são mostrados.

Você pode ler sobre essas opções no manual do Git, digitando `git log --help`.

Algumas opções possíveis, a título de ilustração:

Para mostrar somente o hash e o título do commit, digite `git log --oneline`:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git log --oneline
503517d (HEAD -> master) Add guide expansion section and contact information
aef0f34 Initial commit
```

[illegible]

Fluxos de trabalho adicionais

Introdução

O fluxo de trabalho que vimos até agora é o mais básico.

Mas ao usar o Git, talvez você precise conhecer algumas outras funcionalidades ou limitações, que vamos introduzir aqui.

Não necessariamente você vai precisar imediatamente desses assuntos. Pode usar como um manual de consulta, recorrendo a esta seção quando necessário.

Git não rastreia pastas

Se você criar uma pasta, por exemplo contato, verá que ela não aparece no `git status`.

Isso ocorre porque o Git não rastreia pastas, somente arquivos.

Uma vez que você crie algum arquivo dentro dessa nova pasta, por exemplo teste.txt, o `git status` vai passar a marcar.

Veja o passo a passo:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ mkdir contato
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
nothing to commit, working tree clean
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ touch contato/teste.txt
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    contato/

nothing added to commit but untracked files present (use "git add" to track)
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git add contato/teste.txt
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   contato/teste.txt
```

Deletando um arquivo

Digamos que você tem um arquivo contato.txt que já está sendo rastreado pelo Git.

Diagramaticamente, estamos como na figura abaixo (estamos representando somente o arquivo contato.txt, mas na prática pode haver outros arquivos):



E você deseja remover esse arquivo do projeto.

Fluxos de trabalho adicionais

Ao excluí-lo da Working Tree, por exemplo com o comando de terminal `rm contato.txt` ou usando a interface gráfica do sistema operacional (como o Explorador de Arquivos no Windows), o diagrama fica:



Nesse momento, o `git status` vai mostrar algo que talvez você não tenha visto antes:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ 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:    contato.txt

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

Ele está dizendo que, como o arquivo está presente no Index mas não na Working Tree, a mudança que ocorreu foi que ele foi deletado.

Para registrar essa deleção no próximo commit, você precisa antes atualizar o Index, como já sabe.

Então o comando para atualização do Index ainda é `git add contato.txt`, apesar de parecer estranho, já que não existe mais um arquivo `contato.txt` na Working Tree.

Veja como fica o `git status`:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git add contato.txt
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    contato.txt
```

No diagrama, estamos agora assim:



Fluxos de trabalho adicionais

Agora, se você executar `git commit -m "Remove contact"`, o diagrama ficará assim:



Que é o desejado ! O novo commit não inclui o arquivo.

Renomeando um arquivo

Quando você renomeia um arquivo em um repositório, o Git não rastreia essa ação explicitamente.

Em vez disso, ele usa uma abordagem heurística para detectar renomeações.

Por exemplo, se você tem um arquivo `contato.txt` que já está sendo rastreado pelo Git, e o renomeia para `contatos.txt`, o `git status` vai mostrar o seguinte:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ 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:    contato.txt

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

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

Ou seja, o Git não entendeu que foi uma renomeação do arquivo. Para ele, o arquivo `contato.txt` foi excluído e um novo arquivo `contatos.txt` foi criado.

Mesmo assim, ao adicionar as mudanças com `git add contato.txt contatos.txt`, o `git status` passa a mostrar o seguinte:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:   contato.txt -> contatos.txt
```


Fluxos de trabalho adicionais

Portanto agora o Git reconheceu que foi somente uma renomeação.

Isso ocorre porque, ao adicionar as mudanças, o Git compara o conteúdo do arquivo deletado contato.txt e do arquivo adicionado contatos.txt. Como são iguais, o Git entende que é uma renomeação.

Mesmo que os conteúdos não fossem iguais, mas parecidos, ainda seria reconhecido como uma renomeação.

Esse processo é chamado de "heurístico" porque, se você renomear o arquivo mas também modificar bastante seu conteúdo, o Git pode não reconhecer que foi uma renomeação.

Nesse caso, o `git status` ficaria assim:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    contato.txt
    new file:   contatos.txt
```

Se você quer evitar isso, faça a renomeação e o commit em seguida, sem modificar o conteúdo do arquivo.

Movendo um arquivo

Mover um arquivo funciona da mesma maneira que renomear um arquivo.

Ao executar `git add`, o Git vai comparar os conteúdos do arquivo deletado e do arquivo criado.

Se forem conteúdos suficientemente parecidos, será reconhecido como uma movimentação em vez de como a criação de um arquivo novo.

Fluxos de trabalho adicionais

Ignorando arquivos

Normalmente, qualquer arquivo criado dentro da pasta do seu projeto será percebido pelo `git status` como "Untracked file".

Provavelmente você desejará adicionar esse arquivo com `git add`. Mas há situações específicas em que na verdade você não quer que ele seja rastreado pelo Git.

Por exemplo, se você criou um arquivo `anotacoes.txt` para fazer anotações rápidas de algumas coisas que precisa fazer, mas não quer que elas sejam rastreadas pelo Git.

Nesse caso, não é estritamente um problema que o `anotacoes.txt` apareça no `git status`, mas isso aumenta a chance de você adicionar o arquivo ao Git sem querer.

Por exemplo, se executar o comando `git add .`, que adiciona todas as mudanças ao Git, esse arquivo vai junto.

Felizmente, é possível dizer ao Git para ignorar totalmente o `anotacoes.txt`, assim ele não aparecerá no `git status` e não será incluído no caso do comando `git add ..`

Para isso, crie um arquivo chamado exatamente `.gitignore`, e escreva dentro dele uma linha com o nome do arquivo que você quer ignorar.

Ao executar `git status` de novo, o `anotacoes.txt` não aparecerá mais.

Note que o `.gitignore` tem um nome "mágico": ao ver que existe um arquivo com esse nome, o Git lê o conteúdo dele, e ignora todos os arquivos cujo nome consta lá.

Fora isso, o `.gitignore` é um arquivo como qualquer outro, e deve ser adicionado ao Git com `git add .gitignore`.

O `.gitignore` é bem flexível, seguem algumas maneiras de usá-lo:

- Se há vários arquivos que você quer ignorar, escreva um por linha.
- Se quiser ignorar todos os arquivos dentro de uma pasta chamada `anotacoes`, escreva uma linha `anotacoes/`
- Se quiser ignorar todos os arquivos na pasta principal que terminam em `.txt`, escreva uma linha `*.txt`
- Se quiser ignorar todos os arquivos com final `.txt` que estejam dentro de uma pasta chamada `anotacoes`, **mas não** dentro de subpastas dessa pasta, escreva uma linha `anotacoes/*.txt`
- Se quiser ignorar todos os arquivos com final `.txt` que estejam dentro de uma pasta chamada `anotacoes`, **e também** dentro de subpastas dessa pasta, escreva uma linha `anotacoes/**/*.*.txt`

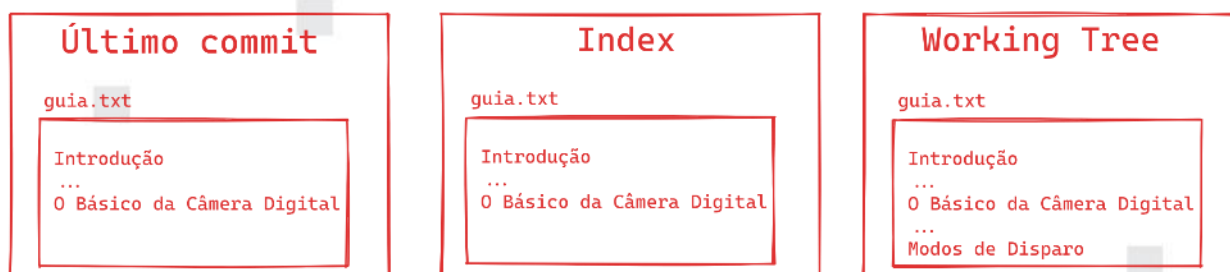
E ainda há outras funcionalidade possíveis. Você pode ler sobre elas na documentação oficial do Git: <https://git-scm.com/docs/gitignore>

Fluxos de trabalho adicionais

Retirando mudanças do Index

Se você adicionou as mudanças de algum arquivo ao Index com `git add`, mas não quer mais que essas mudanças sejam incluídas no próximo commit, você pode usar o comando `git reset` para retirá-las do Index.

Por exemplo, digamos que, logo após um commit, o projeto está assim:

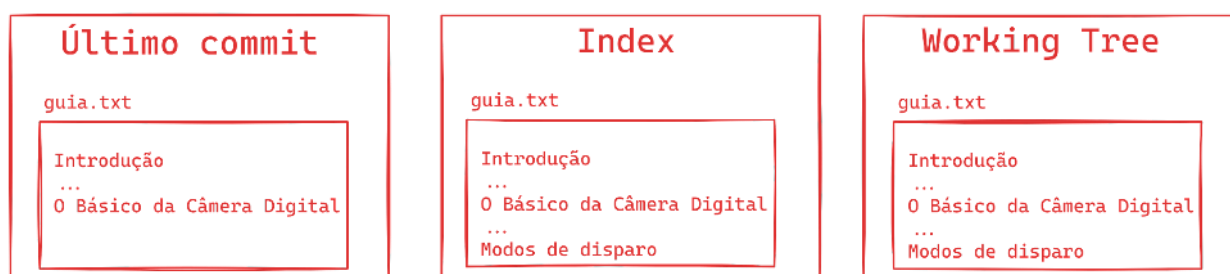


Correspondentemente, o `git status` mostra:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ 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:   guia.txt

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

Então você executa `git add`, ficando com o diagrama:



E com o `git status`:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   guia.txt
```

Fluxos de trabalho adicionais

Logo, caso execute `git commit`, a seção "Modos de Disparo" do `guia.txt` será incluída no commit.

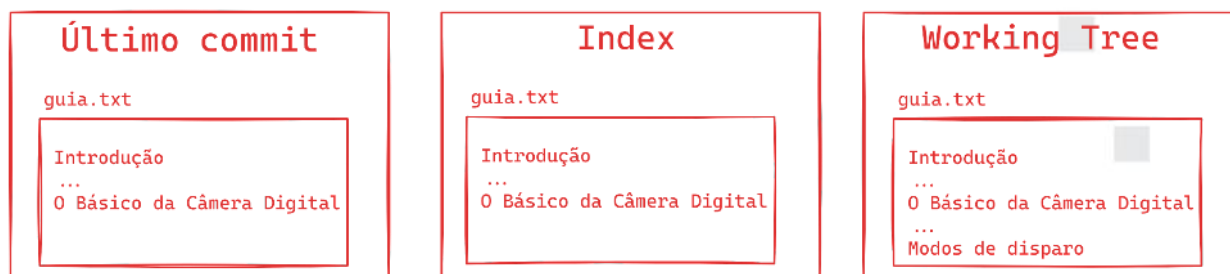
Se você não quer isso, pode fazer o contrário do `git add`.

Apesar de não intuitivo, o contrário de `git add guia.txt` é `git reset HEAD guia.txt`. Veja:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git reset HEAD guia.txt
Unstaged changes after reset:
M   guia.txt
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ 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:   guia.txt

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

Ou seja, voltamos ao que tínhamos antes do `git add`, e o diagrama é:



Agora o próximo commit não vai incluir as mudanças do `guia.txt`, porque essas mudanças não estão mais no Index.

Explicando o nome do comando, `git reset HEAD guia.txt` fez o `guia.txt` no Index ser resetado para coincidir com seu conteúdo no último commit, que é chamado de HEAD no Git (ainda veremos sobre HEAD detalhadamente em outras aulas).



O comando `git reset` é poderoso.

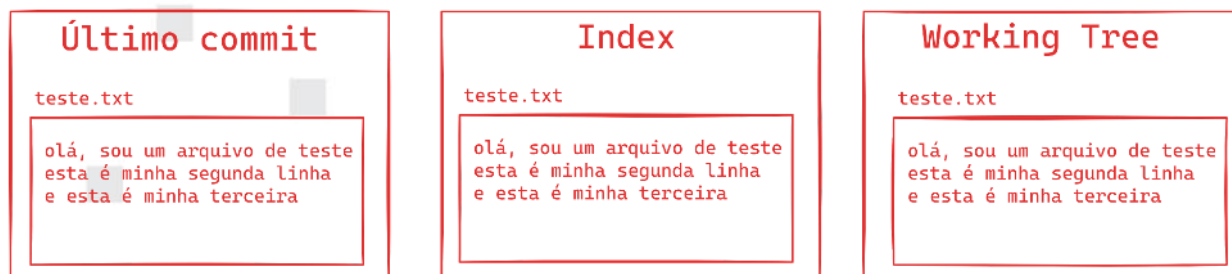
Ele tem muitas variações que podem fazer coisas bem diferentes do que somente retirar uma mudança do Index.

Veremos sobre outros usos do `git reset` em outras aulas.

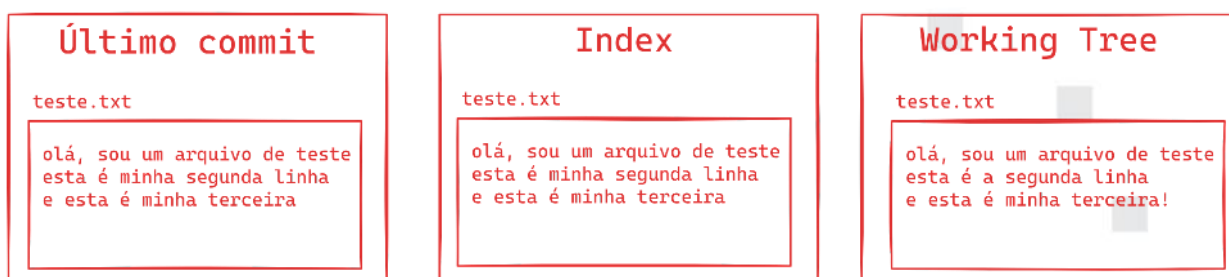
Fluxos de trabalho adicionais

Vendo o conteúdo das mudanças

Para começar, suponha que você acaba de realizar um commit, então o git status está vazio, e o projeto está diagramaticamente assim:



Depois disso, você modifica o teste.txt e o diagrama passa a ser:



Note que as únicas mudanças foram a troca de "minha" por "a" na segunda linha, e a adição de um ponto de exclamação no fim da terceira linha.

Se o tempo passar, você eventualmente vai esquecer exatamente como estava o arquivo no último commit, e o que foi exatamente que você modificou desde então.

Aqui entra o comando `git diff`. Com ele, você pode comparar duas versões do mesmo arquivo para ver quais são as diferenças.

No nosso cenário, queremos comparar o conteúdo do arquivo teste.txt que está na Working Tree com o mesmo arquivo no último commit.

Para isso, executamos o comando `git diff HEAD -- teste.txt` (HEAD significa o último commit), que vai mostrar o seguinte:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git diff HEAD -- teste.txt
diff --git a/teste.txt b/teste.txt
index ab29630..383fe69 100644
--- a/teste.txt
+++ b/teste.txt
@@ -1,3 +1,3 @@
 olá, sou um arquivo de teste
-esta é minha segunda linha
-e esta é minha terceira
+esta é a segunda linha
+e esta é minha terceira!
```


Fluxos de trabalho adicionais

Essa é a visão da diferença do último commit para a Working Tree.

Linhas com um sinal de - na frente (vermelhas) são linhas que o Git entende que você removeu.

Já as linhas com + na frente (verdes) são aquelas que o Git entende que você adicionou.

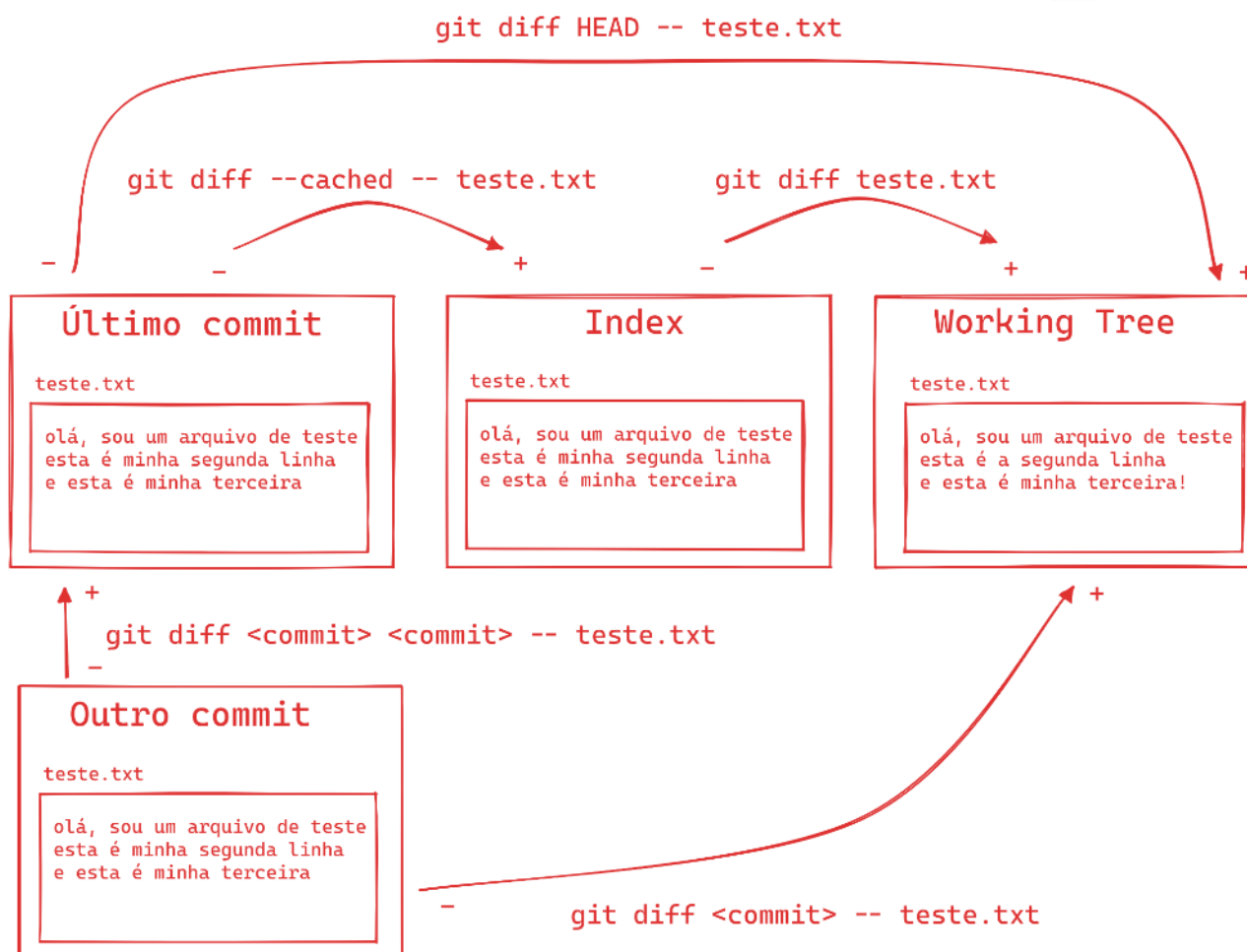
Olhando a figura, o Git entende que nós removemos as duas últimas linhas do arquivo e adicionamos outras duas linhas no lugar (não está errado ! Apesar de que seria mais fácil dizer que modificamos somente alguns caracteres nessas linhas).

O comando `git diff` tem várias opções que você pode usar para comparar diferentes versões do mesmo arquivo.

Por exemplo:

- `git diff --cached -- teste.txt` compara o Index contra o último commit
- `git diff teste.txt` compara a Working Tree contra o Index
- `git diff <commit> <commit> -- teste.txt` compara dois commits quaisquer, onde <commit> é ou o hash de um commit ou a palavra HEAD (último commit). O hash de um commit é longo, mas normalmente os 4 primeiros caracteres já são suficientes para o Git.
- `git diff <commit> -- teste.txt` compara a Working Tree contra um commit qualquer

Vamos colocar essas variações numa figura para ficar mais fácil de visualizar:



Mas essas não são as únicas variações possíveis !

Você pode consultar o manual oficial do Git para ver mais sobre o `git diff`: <https://git-scm.com/docs/git-diff>

Fluxos de trabalho adicionais

Remendando o último commit

Suponha que você tem algumas mudanças que deseja colocar no próximo commit (não importa quais são as mudanças exatamente, pode ser qualquer coisa):

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    teste1.txt
    teste2.txt

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

Por engano, você dá `git add teste1.txt` mas esquece de `git add teste2.txt`, aí faz o commit:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git add teste1.txt
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git commit -m "add test files"
[master d24d5c1] add test files
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 teste1.txt
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git log --oneline
d24d5c1 (HEAD -> master) add test files
503517d Add guide expansion section and contact information
aef0f34 Initial commit
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    teste2.txt

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

O commit foi feito mas o teste2.txt ficou de fora, que não era o que você queria ! E agora ?

O Git tem uma ferramenta para resolver isso.

Primeiro dê `git add teste2.txt` como se fosse fazer outro commit.

Mas agora digite `git commit --amend`

Veja o que acontece:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git add teste2.txt
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git commit --amend
[master 852bd96] add test files
 Date: Mon Nov 13 19:22:52 2023 -0400
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 teste1.txt
 create mode 100644 teste2.txt
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git log --oneline
852bd96 (HEAD -> master) add test files
503517d Add guide expansion section and contact information
aef0f34 Initial commit
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git status
On branch master
nothing to commit, working tree clean
vitor@DESKTOP-QG84R3G:~/guia-fotografia$
```

Fluxos de trabalho adicionais

O `--amend` refaz o último commit em vez de fazer um novo.

Ele incluiu no último commit tanto `teste1.txt` quanto `teste2.txt`, que era o que você queria.

Em resumo: se você acaba de fazer um commit mas esqueceu de adicionar algumas mudanças, basta adicioná-las depois e executar `git commit --amend`.

Observe que o hash do último commit mudou quando você fez o `--amend`.

Isso significa que essa opção não **modifica** o commit, mas sim faz **outro** commit no lugar.

Por último, infelizmente o `--amend` só consegue refazer o último commit.

Então, se você quer refazer um commit que não é o último, precisa de outra ferramenta.

Fluxos de trabalho adicionais

Tags

Uma tag é um rótulo que você pode dar a um commit, como se fosse um marcador de página de livro. Geralmente é usada por desenvolvedores de software para marcar as versões prontas do software, como "v1.0", "v2.0" etc.

Então os commits que não são marcados por nenhuma tag ficam implicitamente entendidos como versões "desenvolvimento" ou "beta" que não estão prontas para publicação.

Para criar uma tag, suponha que comecemos com o seguinte histórico:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git log --oneline
5d35265 (HEAD -> master) teste.txt
503517d Add guide expansion section and contact information
aef0f34 Initial commit
```

Para marcar uma tag com nome "v2.0" ao último commit, basta executar `git tag v2.0`:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git tag v2.0
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git log --oneline
5d35265 (HEAD -> master, tag: v2.0) teste.txt
503517d Add guide expansion section and contact information
aef0f34 Initial commit
```

Se você quiser marcar um commit que não é o último, o comando é `git tag <nome da tag> <hash do commit>`, por exemplo `git tag v1.0 5035`:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git log --oneline
5d35265 (HEAD -> master, tag: v2.0) teste.txt
503517d (tag: v1.0) Add guide expansion section and contact information
aef0f34 Initial commit
```

Para mostrar todas as tags existentes, você pode digitar `git tag`, isso vai mostrar uma lista das tags. E para deletar uma tag, o comando é `git tag -d <nome da tag>`, por exemplo:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git tag -d v2.0
Deleted tag 'v2.0' (was 5d35265)
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git log --oneline
5d35265 (HEAD -> master) teste.txt
503517d (tag: v1.0) Add guide expansion section and contact information
aef0f34 Initial commit
```

Essas tags que exemplificamos acima são chamadas de "lightweight tags".

Existem ainda as "annotated tags", que são tags onde você pode adicionar uma mensagem e uma assinatura.

Para saber mais, consulte a documentação oficial do Git: <https://git-scm.com/book/en/v2/Git-Basics-Tagging>

Fluxos de trabalho adicionais

Alias

A palavra "alias" significa "apelido".

No Git, serve para criar "apelidos" para comandos, para que você não precise digitar o nome inteiro do comando.

Por exemplo, se você gosta de ver seu histórico de commits com o comando `git log --oneline` mas não quer ter que digitar tudo isso toda vez que quer ver o histórico, você pode criar um alias para ele.

Isso é feito com o comando `git config --global alias.<nome do apelido> "<nome do comando real>"`.

Por exemplo, `git config --global alias.l "log --oneline"`. Veja:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git config --global alias.l "log --oneline"
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git l
5d35265 (HEAD -> master) teste.txt
503517d (tag: v1.0) Add guide expansion section and contact information
aef0f34 Initial commit
```

Ou seja, agora `git l` é o mesmo que `git log --oneline`.

Só precisamos executar o `git config` uma vez. Depois disso, o alias fica registrado nas configurações globais, e sempre `git l` funcionará.

Note que `git l` não é um comando nativo do Git. Ele só existe no nosso sistema porque fizemos a configuração do alias.

Se você quiser mudar o comando real, basta re-executar o `git config`, por exemplo `git config --global alias.l "log --oneline --graph"`:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git config --global alias.l "log --graph"
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git l
* commit 5d3526583042849f3b99e46541d1cf9bc2f94332 (HEAD -> master)
| Author: megatron0000 <vitor_pimenta97@hotmail.com>
| Date: Sun Nov 12 19:58:33 2023 -0400
|
|     teste.txt
|
+---+
* commit 503517dcadbccd0255110bda1d4f7c6014a45ef7 (tag: v1.0)
| Author: megatron0000 <vitor_pimenta97@hotmail.com>
| Date: Sun Nov 12 16:03:08 2023 -0400
|
|     Add guide expansion section and contact information
|
+---+
* commit aef0f34ecdd92ae2e722f8137e61c1c1f94e6bc7
| Author: megatron0000 <vitor_pimenta97@hotmail.com>
| Date: Sun Nov 12 14:17:15 2023 -0400
|
|     Initial commit
```

E se quiser remover o alias, use `git config --global --unset alias.l`:

```
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git config --global --unset alias.l
vitor@DESKTOP-QG84R3G:~/guia-fotografia$ git l
git: 'l' is not a git command. See 'git --help'.

The most similar commands are
    log
    lfs
```

Como esperado, `git l` deixa de funcionar.