Introdução ao Git e ao GitHub

# Introdução ao Git

## Entendendo o que é o Git e sua importância

### Por que eu estou aprendendo essa tecnologia e por que ela se faz relevante na minha carreira?

Para entender, vamos a uma simples analogia: Vamos supor que você é um jornalista, você trabalha em um jornal e no final da sua semana você tem que entregar uma reportagem pronta, e esse processo de escrita envolve o seu conhecimento próprio de escrita, técnico e envolve outras pessoas que fazem parte da sua equipe, revisando, fazendo correções, captando aqueles pequenos erros que você como mente criativa por trás do texto talvez não tenha conseguido identificar. Então você começou a escrever essa reportagem e salva como rascunho, logo você submete esse arquivo para um editor olhar e pegar esses pequenos erros e fazer algumas correções e ele te sugere um monte de coisas para alterar.

Então você vai nesse arquivo e renomeia para antigo, então você agora tem uma versão antiga contendo algumas informações do rascunho inicial dessa reportagem que você queria e o arquivo novo que é o arquivo que você vai trabalhar atualmente.

Então você realizou todas as alterações que o editor solicitou e enviou para ele novamente e mais uma vez ele retornou com outras alterações, então o arquivo atual agora será renomeado para o antigo2, ou seja, agora você já tem 3 arquivos.

Seu editor aprovou a última versão que você enviou para ele, agora você gera um PDF para enviarmos para o editor chefe.

Agora imagina isso no contexto de programador, no desenvolvimento de um software. Que as vezes manter na sua mente o estado do arquivo não é o suficiente, você tem que ter informações especificas sobre esse arquivo, distribuir em pastas diferentes, aplicações diferentes, e a gente como desenvolvedor temos “editores” que são muito exigentes, que são os interpretadores de código. Esse é o foco principal desse curso de Git e GitHub.

### GIT

O Git foi criado em 2005 e é um sistema de versionamento de código distribuído e ele foi criado por Linus Torvalds.

Linus Torvalds é um nome muito famoso além de ser muito inteligente é um dos criadores do Linux.

Um software não é feito se não for colaborativo. O Linus precisava de um software que criasse versionamento do trabalho que ele está desenvolvendo e que fosse complexo suficiente para que suportasse pessoas do mundo inteiro trabalhando, as vezes no mesmo arquivo, as vezes alterando a mesma linha.

É isso que o GIT soluciona, ele é um software de versionamento de código, que ajuda a gente a criar e monitorar diferentes versões do nosso código dentro da nossa máquina, embora o GIT hoje seja um padrão de mercado ele não é uma invenção do Linus, na verdade já existia no sistema o versionamento de código, e o GIT é apenas uma criação do Linus a partir do descontentamento com os sistemas de versionamento de código da época, para poder trabalhar e versionar utilizando o kernel do linux.

“Linus: Na verdade eu nunca usei CVS no kernel. Para os primeiros dez anos mantendo o Kernel nós, literalmente usávamos tarballs e patches o que é um sistema de controle de versão muito superior ao da CVS, mas, eu acabei usando CVS por sete anos em uma empresa e eu odiei com paixão, quando eu digo que odiei CVS com paixão também tenho de dizer que se tiver algum usuário do Subversion na plateia, talvez seja melhor você ir embora, porque meu ódio de CVS significa que eu vejo a Subversion como o projeto mais sem sentido já criado. Porque o slogan da Subversion por um tempo foi ‘CVS do jeito certo’, ou algo do tipo, se você começa com esse slogan não tem mais para onde ir.”

### GitHub

Depois que você já tem um software para cuida dessa parte de versionamento do seu código onde você o guarda? Então outras empresas surgiram para solucionar demanda.

Hoje em dia nós temos o GitHub que é uma empresa da Microsoft. Git e GitHub não são a mesma coisa.

O GitHub é o repositório online, onde vamos armazenando o nosso código, é onde a competição fica um pouco mais acirrada, o Git ele é OpenSource (aberto), mas o GitHub e as empresas que vão fazer o host desse repositório de código elas podem ter planos pagos, então é ai que as coisas podem ficar um pouco mais nubladas.

O GitHub não é o único, tem outros sistemas que oferecem serviços parecidos, como o bitpucket, o GitLab, entre outros.

### Benefícios do Git e do GitHub

* Controle de Versão
* Armazenamento em nuvem
  + Uma vez que seu código estiver utilizando esse método, você estará utilizando toda a estrutura da Microsoft na nuvem para poder armazenar o seu código.
* Trabalho em equipe
  + O próprio versionamento de código te permite trabalhar em equipe e o GitHub potencializa isso, porque ele melhora o seu código também, você vai expor o seu código para o mundo, então você vai ter pessoas de diferentes nacionalidades, de diferentes locais olhando para o seu código e podendo interagir com ele.
* Melhorar seu código
* Reconhecimento
  + O GitHub funciona como uma rede social, e vem cada vez mais caminhando para uma abrangência social, uma interação social maior e se você cria um projeto que é bem aceito na comunidade com certeza você vai ganhar reconhecimento por ele.

# Navegação via command line interface e instalação

## Comandos básicos para um bom desempenho no terminal

### GUI x CLI

A maioria dos sistemas operacionais tem programas que possuem uma interface gráfica, ou seja, a forma como o usuário interage com esses programas é de forma gráfica, clicar, arrastar, tudo é responsivo, ou seja, responde aos comandos dos usuários.

O software que vamos interagir que é o GIT, o desing dele é voltado para outro tipo de programa, então ele é um CLI, ele não tem uma interface gráfica. Hoje em dia nós temos programas que automatizam isso e que pegam o GIT e acrescentam uma Grafic User Interface.

### O que vamos aprender?

* Mudar de pastas
* Listar as pastas
* Criar pastas/arquivos
* Deletar pastas/arquivos

**Windows**

* cd
* dir
* mkdir
* del / rmdir

**Unix**

* cd
* ls
* mkdir
* rm -rf

### Abrindo o cmd

Windows > digitar: cmd e abrir a primeira opção (Prompt de comando).

Ele vai abrir o terminal do Windows na pasta padrão: c:\Users\enascimj>

Texto

Descrição gerada automaticamente com confiança média

Uma coisa muito interessante que o Windows 10 tem é Windows Subsystem For Linux, ou seja, é um sistema Linux rodando dentro do sistema Windows.

### Comando DIR

Esse comando lista todas as pastas que contém dentro do diretório informado. Se for Linux você irá utilizar o comando LS.

Todos esses comandos que serão utilizados, eles possuem variâncias, então eles possuem flags que são complementos que passamos para esses comandos, que eles acrescentam, modificam ou formatam a forma que esses comandos são devolvidos para a gente.

Texto

Descrição gerada automaticamente

### Comando CD

Esse comando vai possibilitar que a gente navegue entre as pastas e ele é igual para todos os sistemas operacionais.

**cd/** - Vai te levar para a base do diretório C:



Se após eu utilizar esse comando eu utilizar o comando DIR ele vai listar todas as pastas que têm dentro do diretório C:

Texto

Descrição gerada automaticamente

**cd Windows** – Vai te levar para a pasta Windows.



Se após eu utilizar esse comando eu utilizar o comando DIR ele vai listar todas as pastas que têm dentro do diretório Windows.

Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa

**cd ..** – Retorna para o diretório anterior.

Texto

Descrição gerada automaticamente com confiança média

**cls –** Limpa o terminal (No Linux o atalho é clear ou crtl + L)

Forma

Descrição gerada automaticamente com confiança média

### Comando MKDIR

Comando para criar uma pasta

**mkdir** (nome da pasta)

Texto

Descrição gerada automaticamente

### Comando ECHO

Comando utilizado para imprimir uma frase no terminal, porém, se eu digitar: echo hello > hello.txt ele vai verificar se existe esse arquivo na pasta se não tiver ele vai criar esse arquivo.

**echo hello** – Só vai printar na tela o que eu escrever na frente do echo.

**Texto

Descrição gerada automaticamente**

**echo hello > hello.txt**

****

### Comando DEL

Comando utilizado para deletar arquivos dentro das pastas selecionadas.

**del workspace**

**Texto

Descrição gerada automaticamente**

Verifique que você não encontra mais o arquivo hello.txt

### Comando RMDIR

Comando utilizado para excluir uma pasta.

**rmdir workspace /S /Q** (/S /Q se não utilizado vai informar que a pasta não está vazia, logo você não vai conseguir excluir)



### Comando MV

Comando utilizado para mover um arquivo para outra pasta.

### Ressaltando as principais diferenças entre os sistemas operacionais

Para cada sistema operacional existe uma forma de instalar o Git. Foi realizada a instalação do Git para Windows pois é o sistema operacional que utilizamos.

## Realizando a instalação do Git

git-sm.com

Caso tenha alguma dúvida de instalação é só acessar o link abaixo:

[Cliquei aqui](https://web.digitalinnovation.one/course/introducao-ao-git-e-ao-github/learning/014fe14a-dc5a-41ec-9965-755a79694f27)

# Entendendo como Git funciona por baixo dos panos

## Tópicos fundamentais para entender o funcionamento do Git

Entender alguns conceitos fundamentais da tecnologia para que a gente consiga lá na frente, quando começarmos a executar o nosso primeiro comando ter a ciência do que acontece por baixo dos panos.

* SHA1
* Objetos fundamentais
* Segurança

### SHA1

A sigla SHA1 significa Secure Hash Algorithm (Algoritmo de Hash Seguro), é um conjunto de funções hash criptográficas projetadas pela NSA (Agência de Segurança Nacional dos EUA).

A encriptação gera conjunto de caractere identificador de 40 dígitos.

Se você tem um arquivo de texto muito grande, e você vai nesse arquivo de texto e roda esse algoritmo, ele vai gerar um conjunto de caracteres de 40 dígitos. Se eu for nesse arquivo novamente e alterar uma única vírgula e gerar esse algoritmo novamente ele vai gerar outro conjunto de caracteres de 40 dígitos. Se eu for no arquivo e remover essa vírgula e rodar o algoritmo ele vai gerar o conjunto de 40 dígitos anterior.

Ele é uma forma culta de representar um arquivo.

## Objetos internos do Git

Existem 3 tipos básicos de objetos interno no Git.

* BLOBS
* TREES
* COMMITS

### BLOBS

Os arquivos ficam armazenados dentro do objeto chamado BLOB, e esse objeto contém meta dados dentro dele.

Então o objeto BLOB vai ter dentro dele o tipo de objeto, o tamanho da String ou do arquivo, uma barra ao contrário com o numeral zero e o conteúdo de fato desse arquivo.

### TREE

A TREE armazena os BLOB, então é uma crescente, o BLOB sendo o bloco base de composição, a TREE armazenando e apontando para tipos de BLOBS diferentes.

A TREE também contém meta dados, ela aponta para um BLOB, que por sua vez tem um SHA1, e ela também guarda o nome desse arquivo. A TREE vai ser responsável por montar toda a estrutura de onde estão localizados esses arquivos, então a TREE pode apontar tanto para a BLOB ou para outras TREE, e por que isso acontece? Porque os diretórios dentro de um sistema operacional podem conter outros diretórios. Então faz sentido que o Git use um tipo de objeto que é cursivo. A TREE tem o seu próprio SHA1.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

### COMMIT

É o objeto que vai juntar tudo, é o objeto que vai dar sentido para a alteração que você está fazendo. Então o COMMIT aponta para uma árvore, aponta para um parente, ou seja, o último COMMIT utilizado antes dele, aponta para o autor e para uma mensagem também. Então o autor e a mensagem fazem parte desse sentido que citamos no início. Você pode escrever uma mensagem nesse COMMIT que vai explicar todas as alterações que foram realizadas nesse arquivo. O nome do autor é quem fez o COMMIT.

O COMMIT tem um time, que é um carimbo de tempo, ou seja, ele tem a data e hora certinha de quando ele foi criado. O COMMIT também tem um SHA1 próprio.

### Sistema Distribuído Seguro

Imagina que você tem o seu código hosteado em um servidor na nuvem, o código que você tem lá, representa o resultado do seu código, sistema ou software, é a versão mais recente, mais atualizada. Vamos supor que esse repositório tem 40 pessoas contribuindo nele e em cada máquina desses 40 também tem uma versão desse código, então pelo fato de os COMMITS serem tão difíceis de serem alterados, ou até mesmo impossíveis, tanto a versão mais recente que está na máquina do servidor quanto qualquer uma dessas 40 versões que estão distribuídas também são versões confiáveis. Se der um problema na nuvem do GitHub e o código não existir mais também irá acontecer algo com as 40 versões, porque as versões disponíveis deles também são super confiáveis por causa da estrutura que o Git mantém, da forma que o Git foi projetado, por isso que o Git é seguro e um sistema distribuído.

## Chave SSH e Token

Quando você for subir o seu código para o GitHub você vai precisar se autenticar, criar um usuário e senha, porém, em agosto de 2021, o GitHub informou que para você subir o seu código na plataforma você vai precisar fazer alguns outros processos mais seguros para poder se autenticar.

### Chave SSH

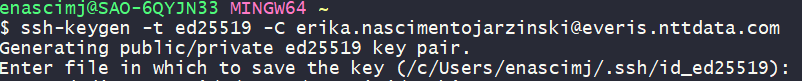
É uma forma de estabelecer uma conexão segura e encriptada entre duas máquinas, no caso vamos nos conectar com o servidor do GitHub, vamos configurar a nossa máquina local com uma máquina confiável pelo GitHub, estabelecendo essa conexão com duas chaves, vai ter sempre uma chave publica e uma chave privada, vamos pegar essa chave pública e colocar ela lá no GitHub e a partir do momento que fizermos isso o GitHub vai conhecer nossa máquina, então todos os repositórios que tivermos na nossa máquina por esse processo SSH e a gente for enviar código para lá o GitHub já vai reconhecer a assinatura da nossa máquina, já vai ter uma conexão pré-estabelecida e a gente vai ser capaz de enviar código sem nem precisar colocar senha. Então a nossa máquina vai ficar configurada propriamente para poder levar esse código para lá.

Para criar essa chave você vai abrir o Git Bash e colocar o seguinte comando:

ssh-keygen -t ed25519 -c [erika.nascimentojarzinski@everis.nttdata.com](mailto:erika.nascimentojarzinski@everis.nttdata.com)



Sempre utilizando o e-mail que você utilizou para cadastrar no GitHub.

Após o entender ele vai perguntar se você deseja criar na pasta que ele está informando

Caso você dê enter, ele vai prosseguir e pedir para você colocar uma senha:

Texto

Descrição gerada automaticamente

Após feitos isso, você pode entrar na pasta ssh:

Texto

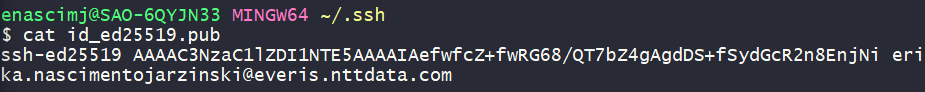
Descrição gerada automaticamente com confiança média

E listar para verificar as chaves pública e privada:

Texto

Descrição gerada automaticamente

Feito isso você vai dar o comando cat na nossa chave publica para poder mostrar o conteúdo dela.



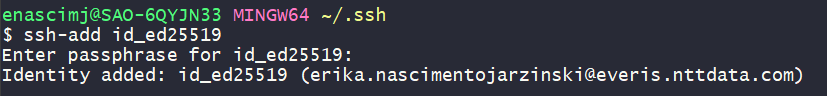
Você vai copiar todo o resultado e ir lá no GitHub para configurar o SSH. Após inserir a chave no GitHub você vai precisar inicializar o agente e entregar chave para ele no GitBash.

Para inicializar o agente você vai fazer o seguinte comando:

Texto

Descrição gerada automaticamente

E para entregar a chave para o agente você vai fazer o seguinte comando:



Lembrando que, você vai entregar a chave privada para o agente, ou seja, aquela chave que não tem o pub no final.

### Token de acesso

Ele tem um início bem parecido também é necessário ir até o GitHub configurar ele, mas ele se assemelha mais ao processo de digitar seu nickname e sua senho do que ter essa vantagem que o SSH tem de não requerer senha nem nada.

Então nesse caso você vai gerar um token de acesso no GitHub, vai guardar ele na sua máquina ou em algum lugar e sempre que você for fazer um commit o Git vai pedir o seu usuário e sua senha e você vai colocar o seu usuário normal e na hora da senha você vai usar o seu token de acesso pessoal.

# Primeiros comandos com Git

## Iniciando o Git e criando um commit

* Iniciar o Git
  + Git init
* Iniciar o versionamento
  + Git add
* Criar um commit
  + Git commit

Quando estamos lidando com terminal colocamos sempre o nome do programa na frente do comando.

### Criando um repositório

Abra o GitBash no repositório workspace criado lá no início.

Criar uma pasta livro-receitas:

**mkdir livro-receitas**

Inicializar o git dentro da pasta:

**git init**

Ao inicializar o Git cria uma pasta oculta dentro da pasta livro-receitas. Para localizar essa pasta dentro do nosso terminal é necessário acrescentar o comando de listar (ls) mais (-a).:

Se eu entrar nessa pasta .git e listar para verificar o que tem lá dentro, vai retornar uma lista que é uma estrutura do próprio Git.

### Adicionando um arquivo

Poderíamos utilizar um arquivo txt, porém vamos utilizar um arquivo chamado Markdown, ele nada mais é uma forma mais humana de se escrever um arquivo html.

Instalar Typora para editar e ler arquivos Markdown. [Link para download](https://typora.io/#windows)

### Criando um COMMIT

git commit -m “Commit Inicial”

O que vai dentro das aspas é a descrição do que foi realizado.

# Ciclo de vida dos arquivos no Git

## Passo a passo no ciclo de vida

### Git Init

Esse comando além de criar a pasta de .git, ele inicializa um conceito do Git chamado de repositório. Quando usado o Git Init de fato estamos criando um repositório no Git dentro daquela pasta.

### Tracked ou Untracked

Tracked são arquivos que nós temos ciência deles e o Untracked são os arquivos que ainda não temos ciência deles.

Dentro dos arquivos que são rastreados dentro do Git ele pode subdividir em três estágios diferentes, **UNMODIFIED, MODIFIED, STAGED,** e claro o **UNTRACKED**.

#### UNMODIFIED

É o arquivo que ainda não foi modificado.

#### MODIFIED

É o arquivo que já sofreu modificação e ele muda automaticamente de UNMODIFIED para MODIFIED. Ele identifica o SHA1 modificado, por isso ele muda de UNMODIFIED para MODIFIED automaticamente.

#### STAGED

É onde fica os arquivos que estão se preparando para poder fazer parte de um outro tipo de agrupamento.

Quando você adiciona um arquivo que ele era UNTRACKED, ou seja, o Git ainda não tinha ciência dele e você dá um Git Add, ele é movido para STAGED, quando você tem um arquivo modificado e você dá um Git Add ele também é movido para a área de STAGED.

### Git Status

Esse comando vai ajudar a monitorar os status dos arquivos, vai dizer se aquele arquivo está UNTRACKED, MODIFIED ou STAGED.

# Introdução ao GitHub

## Trabalhando com o GitHub

# Resolvendo conflitos

## Como os conflitos acontecem no GitHub e como resolvê-los

Quando você tem seu código lá no GitHub e você o baixa para a sua máquina, ou você acabou de enviar ele para lá, então quer dizer que o código que está lá no GitHub é exatamente igual o que está na sua máquina.

Vamos supor que uma outra pessoa vai lá no GitHub, baixe seu código. Nesse momento ninguém mexeu no código ainda, os dois estão sincronizados. É muito difícil duas pessoas fazerem a mesma formatação, as vezes uma vírgula diferente, um espaço diferente, uma quebra de linha diferente, então o código que está no GitHub e o que está com a pessoa já não são mais iguais, então a pessoa empurrou para o GitHub essa nova versão, então o que está na sua máquina está desatualizado. Então na hora que você empurrar o seu código, o GitHub vai te informar que o seu código está desatualizado, nesse momento é quando acontece o conflito que é chamado de conflito de merge, que é quando o GitHub tenta juntar duas alterações e existem duas alterações na mesma linha, então o GitHub não vai fazer nenhuma alteração drástica e não vai resolver nada, ele vai deixar que você abra aquele arquivo, na qual ocorreu o conflito e depois empurre o código para o GitHub, ai você vai estar empurrando o código sem conflitos, você disse para ele qual versão é a versão correta daquela alteração específica daquele arquivo.