**Git**

**Terminal Windows**

* cd (Entra dentro das pastas)
* cd .. (Volta para pasta anterior)
* dir (Lista de diretórios dentro da pasta)
* cls (Limpa o terminal)
* Apertar tecla table (Autoincremento)
* mkdir (Cria uma pasta)
* echo (Printa na tela do terminal)
* echo hello > hello.txt (Cria o arquivo dentro da pasta)
* del (Deleta arquivos)
* rmdir workspace /S /Q (Deleta a pasta com todo seu conteúdo)

**Terminal Linux**

* cd (Entra dentro das pastas)
* ls (Lista de diretórios dentro da pasta)
* mkdir (Cria uma pasta)
* rm -rf (Deleta a pasta com todo seu conteúdo)

**Tópicos Fundamentais**

**SHA1**

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

A encriptação gera conjunto de caracteres identificador de 40 dígitos.

É uma forma curta de representar um arquivo.

Exemplo:



**Objetos Fundamentais**

BLOBS (Armazena Metadados)

* tipo de objeto
* tamanho
* \0
* Conteúdo
* Sha



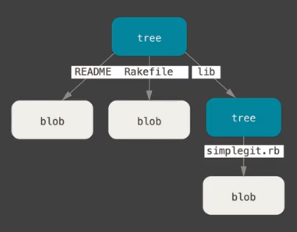
Exemplo:



TREES (Armazena Blobs, também metadados)

* Aponta para um blob
* Tamanho
* \0
* Sha
* Nome do arquivo
* Aponta tanto para blobs ou outras árvores
* Se muda um arquivo muda toda estrutura





COMMITS

* Objeto que junta tudo
* Aponta para uma árvore
* Aponta para um parente
* Aponta para um autor
* Aponta para uma mensagem
* Timestamp (data e hora de quando foi criado)

Também possui SHA1 dos seus metadados, isso significa que se alterar um arquivo irá gerar um SHA1 da blob, que se por sua vez pode interferir sua árvore e no seu commit, ou sea, altera toda estrutura do commit (o commit é único para cada autor).



O SHA1 desse commit é o hash de toda essa informação.

**Sistema Distribuído e Seguro**

É um sistema distribuído e seguro pelo fato de várias pessoas possuírem versões seguras dos códigos fontes, os quais possuem em suas máquinas.

**Primeiros Comando com GIT**

**Iniciar o GIT**

* git init (Inicializa o Git no repositório)
* ls -a (Mostra arquivos ocultos)
* git add \* (Add arquivo no Git)
* git commit -m “commit inicial” (Realiza commit)

**Tracked ou Untracked**

Dentro de Tracked (arquivos rastreáveis) tem-se 3 subdivisões:

* Unmodified (arquivo não foi modificado)
* Modified (arquivo que sofreu modificação)
* Staged (onde fica os arquivos que está preparando para fazer parte de outro tipo de agrupamento)

Git add (Tem um arquivo que estava Untracked (acabou de ser criado no repositório com o git init) e quando utiliza o comando git add, move direto para staged (aguardando)).

Quando o arquivo tem modificação ele muda de unmodified para modified, faz isso comparando com a SHA1 dos arquivos. Depois de rodar de novo o git add esse arquivo vai para staged.

Se tiver um arquivo unmodified e remove ele, então vai para untracked.

No staged está esperando para realizar um commit, envelopa todas as modificações e aguarda. O commit retorna os arquivos para unmodified, para aguardar novas modificações, no commit tem uma “foto” do código naquele momento.

No ambiente de desenvolvimento tem o ambiente de trabalho, a área de staging, como realiza um commit tem-se um repositório local que por sua vez pode-se ter um repositório remoto.

git status monitora o status do arquivo

Move arquivo para outra pasta:



Move todas os arquivos com modificações para staged:



Commit com mensagem:





Verificar se as configurações do servidor remoto e locais estão iguais:



Configurar e-mail:



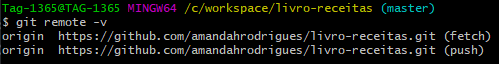
Configurar nickname:



Configurar origem repositório:



Lista os repositórios remotos cadastrados:



Leva código do repositório local para repositório remoto:

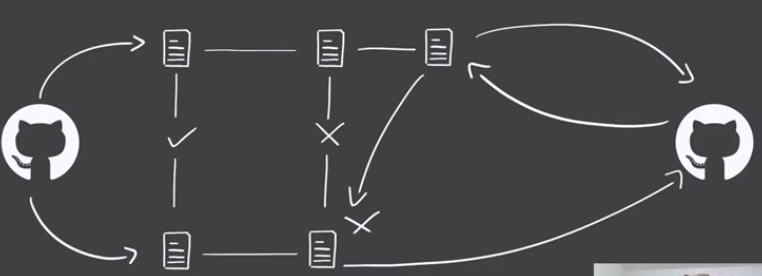


Leva código do repositório remoto para repositório local:



Resolvendo Conflitos

Conflito de merge:



Clona um repositório remoto para o repositório local:



**PRATICANDO COM O GIT**

**GIT INIT –** inicializa o git na pasta que estiver e ela passa a ser main.

**GIT STATUS –** Utiliza para ver a situação atual da pasta.

**TOUCH LISTA-ALUNOS –** Adiciona arquivo de texto a pasta atual.

**GIT ADD –** Utiliza para filtrar arquivos, ou seja, para adicionar os arquivos com alterações e poder realizar o commit. Com o git add . adiciona todos os arquivos.

**GIT COMMIT –M “MENSAGEM” -** Realiza commit com mensagem para futura identificação.

**GIT LOG** – Utiliza pra verificar o histórico de commits.

**GIT LOG --GRAPH** – Utiliza pra verificar o caminho que as branchs fizeram.

**GIT BRANCH “NOME BRANCH”** – Utiliza para criar uma nova branch.

**GIT BRANCH** – Utiliza para listas as branchs.

**GIT CHECKOUT “NOME BRANCH”** – Utiliza para trocar de branch.

**GIT CHECKOUT –B “NOME BRANCH”** – Utiliza para criar uma nova branch e já fazer a troca pra essa branch.

**NANO “NOME ARQUIVO”** – Utiliza para abrir editor de texto na linha de comando.

**CAT “NOME ARQUIVO”** – Utiliza para listar o que está dentro do arquivo.

**GIT MERGE “NOME BRANCH”** – Utiliza para enviar modificações de uma branch para outra.

**GIT MERGE --ABORT** – Utiliza para abortar o merge.

**GIT DIFF** – Utiliza para verificar a diferença entre os arquivos que estão dando conflitos.

**Arquivo no editor nano**

Se você deseja salvar as alterações feitas, pressione Control + O. Para sair do nano, digite Control + X. Se você pedir ao nano para sair de um arquivo modificado, ele perguntará se você deseja salvá-lo. Basta pressionar N, caso contrário, ou Y, caso contrário. Ele solicitará um nome de arquivo. Basta digitar e pressionar Enter.

Se você acidentalmente confirmou que deseja salvar o arquivo, mas na verdade não o faz, sempre pode cancelar pressionando Control + C quando for solicitado um nome de arquivo.