INTRODUÇÃO AO GIT E GITHUB

AULA 01 - INTRODUÇÃO AO GIT:

**1.ENTENDENDO O QUE É GIT E SUA IMPORTÂNCIA:**

\*benefícios: controle de versão; armazenamento em nuvem; trabalho em equipe; melhorar seu código; reconhecimento

AULA 02 - NAVEGAÇÃO VIA COMMAND LINE INTERFACE E INSTALAÇÃO:

**1.COMANDOS BÁSICOS PARA UM BOM DESEMPENHO NO TERMINAL:**

\*o que vamos aprender: mudar de pastas, listar as pastas, criar pastas/arquivos, deletar pastas/arquivos, usando o line interface (terminal).

\*windows: terminal: cd, dir, mkdir, del/rmdir.

\*unix terminal: cd, ls, mkdir, rm-rf.

\*passo a passo usuário windows:

-para abrir terminal: botão windows, escrever cmd.

-para listar (responde com uma lista de diretórios dentro da pasta situada): dir

-navegar entre as pasta (change directory): cd /

-entrar na pasta windows: cd Windows

-voltar para pasta anterior: cd ..

-limpar tela (clear screen): cls

-função autocompletar: Tab

-criar pasta (make directory): mkdir

-printar de volta no terminal a palavra hello: echo hello

-criar arquivo hello.txt pelo terminal com a mensagem hello dentro do arquivo (a seta serve para direcionar) (caso não haja arquivo, o sistema irá criar): echo hello > hello.txt

-deletar arquivos dentro do diretório: del

- deletar o diretório/repositório workspace (remove directory): rmdir workspace /S /Q

-navegar entre os comandos utilizados (arrow): seta para cima

**2.REALIZANDO A INSTALAÇÃO DO GIT:**

AULA 03 - ENTENDENDO COMO O GIT FUNCIONA POR BAIXO DOS PANOS:

**1.TÓPICOS FUNDAMENTAIS PARA ENTENDER O FUNCIONAMENTO DO GIT:**

\*sha1: a sigla SHA significa secure hash algorithm, é um conjunto de funções hash criptográficas projetadas pela NSA.

-A encriptação gera conjunto de caracteres identificador de 40 dígitos.

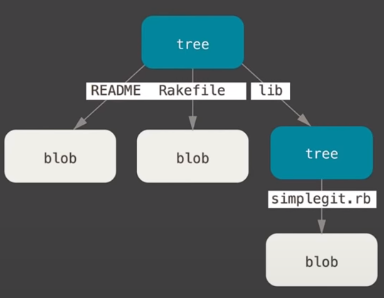
-é uma forma curta de representar um arquivo.

-echo “ola mundo” | openssl sha1

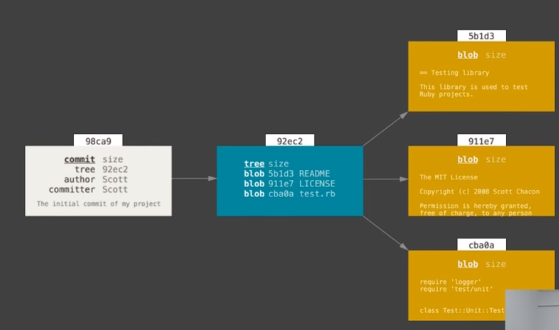
**2.OBJETOS INTERNOS DO GIT:**

\*bloobs: bloco básico de composição, armazena dados e metadados, como tipo do objeto, tamanho da string, tamanho do objeto, etc. contém o sha1 do arquivo.

\*trees: armazenam blobs e apontam para blobs diferentes ou outras árvores. também contém metadados. monta a estrutura de onde estão os arquivos. também tem um sha1, mas das árvores.



\*commits: junta tudo e dá sentido para a alteração. aponta para uma tree, parente, autor, mensagem (dando significado para os arquivos) e timestamp. também tem um sha1.



\*É um sistema distribuído seguro: pq se algo acontecer, como ele é distribuído, todos que tem acesso a ele tem a mesma versão, que pode ser restaurada.

**3.CHAVE SSH E TOKEN:**

\*chave ssh:

<https://web.dio.me/course/introducao-ao-git-e-ao-github/learning/7410b862-1989-421a-a48d-500db5857f53?back=/track/take-blip-web-developer>

\*token de acesso pessoal:

(xxxxxxxxx)

AULA 04 - PRIMEIROS COMANDOS GIT:

**1.INICIANDO O GIT E CRIANDO UM COMMIT**

\*iniciar o git: git init dentro da passta

\*iniciar o versionamento: git add (+ nome do arquivo ou ./\* para mandar tudo)

\*criar um commit: git commit

\*passo a passo criar e editar arquivo.md:

-entrar na pasta workspace

-criar pasta para trabalho com mkdir (ou criar dentro da pasta via windows)

-entrar na pasta criada

-iniciar o git dentro da pasta com o comando git init para criar repositório vazio (pasta .git oculta) (para ver a pasta oculta basta usar ls -a)

-para criar um arquivo dentro da pasta no formato markdown é possível criar dentro da pasta desejada, iniciando o processo como se fosse .txt mas renomeando e colocando o .md para que seja reconhecido como markdown, ou é possível também criar diretamente pelo typora (ou outro editor de markdown, como o vs code) e salvando dentro do local desejado.

-abrir arquivos de markdown pelo typora para editar.

\*comitar arquivo criado:

-no git, usar o comando git add \*

-git commit -m ”commit inicial”

\*download typora:

-para título: #

-para emoji: :

-para negrito: \*\* \*\*

-para itálico: \_ \_

AULA 05 - CICLO DE VIDA DOS ARQUIVOS GIT:

**1.PASSO A PASSO NO CICLO DE VIDA:**

\*git init: é usado para criar um repositório dentro de um diretório.

-tracked: unmodified, modified e staged

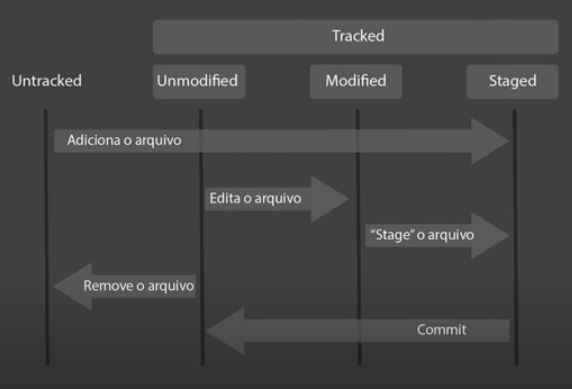
-untracked: arquivo recém criado.

\*ciclo:

-arquivos unmodified após edição se tornam modified pela comparação entre os sha1.

-git add move para staged.

- arquivos em stage estão sendo preparados para o commit, se tornando unmodified quando comitados.



-o git add faz o documento transitar entre o working directory e staging area

-o commit empurra o arquivo do staging area para o repositório local.



\*git status: esse comando devolve o status do documento.

\*comando para mover o arquivo strogonoff para pasta receitas: mv strogonoff.md ./receitas/

-quando se move um arquivo, o git vai entender que o arquivo foi removido.

-para que o git reconheça novamente, é necessário dar staged (git add) no arquivo e nova pasta para ser comitado:

git add strogonoff.md receitas/

git commit -m “cria pasta receitas, move arquivo para receitas”

\*criar arquivo como index para ser indexador da lista de receitas e para que as pessoas vejam todas as receitas o livro de receitas:

-echo > README.md (como foi criado agora, ele é untracked)

-editar o arquivo README para servir como uma espécie de capa para o livro de receitas. (título, descrição e listagem de receitas) e salvar.

-para mover de untracked para staged, usar comando git add \* (o \* ou . faz com que tudo que está no working directory vá para staging)

-agora que ele foi staged, para comitar, usar comando git commit -m “adiciona index”

AULA 06 - INTRODUÇÃO AO GITHUB:

**1.TRABALHANDO COM O GITHUB:**

\*empurrar do repositório local para o repositório remoto.

\*sempre usar o mesmo name e email no git e github. para alterar no git, basta dar unset nas configurações:

-git config –global –unset user.email

-git config –global –unset user.name

\*para setar novamente:

-git config –global user.mail “”

-git config –global user.name “”

(após essa alteração, os commits anteriores ficarão sem autor reconhecido).

\*criar repositório no github e empurrar arquivos:

-criar repositório remoto no github

-copiar url gerada para colar no git bash

-no git bash, adicionar origem de envio dos arquivos com o comando git remote add origin url https

-para verificar, usar o comando git remote -v e git status

-para empurrar: git push origin master

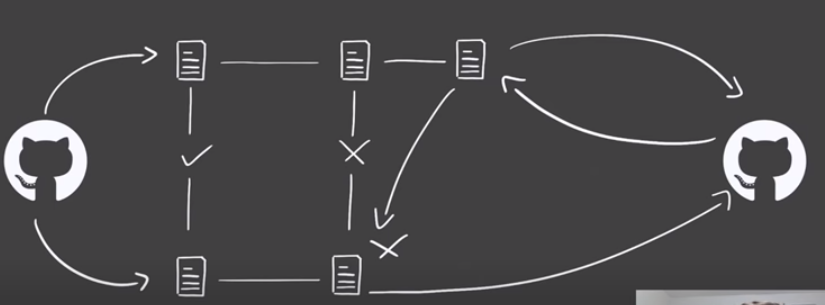
-digitar senha git para empurrar

-recarregar página no github para verificar push

AULA 07 - RESOLVENDO CONFLITOS:

**1.COMO OS CONFLITOS ACONTECEM NO GITHUB E COMO RESOLVÊ-LOS:**

\*conflito de merge: quando o github recebe dois arquivos com edição de duas ou mais pessoas com alterações na mesma linha. o github vai te dar a opção de escolher a edição correta



\*passo a passo edição de arquivo:

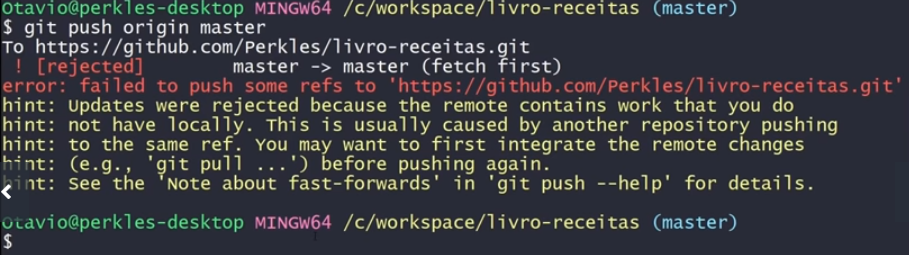
-edição do readme acrescentando uma nova receita, na lista de receitas, a de pavê. (status modified)

-para passar para staged: git add\* (agora está pronto para comitar)

-git commit -m “adiciona receita pavê” (comitado, pronto para push)

-git push origin master

-erro: as mudanças foram rejeitadas pq o repositório remoto contém algum trabalho que você não contém localmente. Devemos primeiro integrar para depois empurrar novamente. para isso devemos puxar para a máquina o arquivo do repositório remoto para que o git mostre os conflitos de merge.



-ao baixar o arquivo e o analisar, é possível perceber que alguém fez uma alteração no arquivo na mesma linha em que adicionamos o pavê, adicionando bolo de cenoura de forma colaborativa.

-para solucionar a questão, deve-se copiar o texto do typora para o bloco de notas, realizar as alterações desejadas (colocar o bolo de cenoura na linha abaixo do pavê) e posteriormente copiar o texto de volta novamente para o typora e salvar.

-status: both modified (os dois foram modificados, tanto o puxado quanto o que desejamos empurrar). agora falta comitar a edição importante.

-git add \*(para enviar para staging)

-git commit -m “resolve conflitos” (pronto para push)

-git push origin master

-conferir no github as alterações.

\*passo a passo para baixar códigos do github: (clonar pelo desktop)

-encontrar o repositório desejado

-copiar a url

-voltar ao git bash

-voltar para a pasta workspace

-para clonar: git clone url

-ls para identificar repositórios (o que difere é que ele vem de fábrica como repositório, contendo uma pasta .git oculta)

-para verificar se a pasta é repositório: entrar na pasta desejada: cd pasta, ls, ls-a(para mostrar repositórios ocultos), git remote -v (vai mostrar os repositórios remotos onde esse repositório baixado está apontado)