Introdução ao Git e ao GitHub

Aula 01: Entendendo o que é Git e sua importância

- Git (Linus Torvalds): software de versionamento de código distribuído
    - Open source
- GitHub (Microsoft): repositório on-line
    - Tem planos pagos
- Benefícios de aprender Git/GitHub:
    - 1. Controle de versão
    - 2. Armazenamento em nuvem
    - 3. Trabalho em equipe
    - 4. Melhorar seu código
    - 5. Reconhecimento

Aula 02: Comandos básicos para um bom desempenho no terminal

- GUI (interface gráfica de usuário) x CLI (interface de linha de comando)
- O que vamos aprender?
    - Mudar de pastas
    - Listar as pastas
    - Criar pastas/arquivos
    - Deletar pastas/arquivos
- Tabela

Ação	Windows	Unix-like
Mudar de pasta	cd	cd
Listar as pastas	dir	ls
Criar pastas/arquivos	mkdir	mkdir
Deletar pastas/arquivos	del (apaga só arquivos) rmdir /S /Q (apaga tudo)	rm -rf (-r apaga as pastas de maneira recursiva; f força a não confirmação do processo)
Limpar o terminal	cls	clear ou Ctrl + L

- Terminal Windows (derivado do Shell) x Terminal Linux (derivado do Bash)
- Atalhos:
    - Windows:
        - Tab: autocompleta
    - Unix:
        - Ctrl + L: limpa a tela do terminal
- Terminal: “silence on sucess”
- Comandos extra:
    - Windows:
        - echo frase: printa frase no terminal
        - echo hello > hello.txt: redireciona frase para um arquivo txt
    - Linux:
        - (echo funciona igual no Windows)

Aula 03: Ressaltando as principais diferenças entre os sistemas operacionais

- Instalando o Git:
    - Windows explorer: aplicação de gerenciamento de pastas
    - Observar os tipos de quebra de linha (existe diferenças entre Windows e Unix)
    - Para instalar o Git no Linux: apt-get install git
    - Para verificar se o Git está instalado no Linux/Windows: git --status
    - Para verificar a versão do Git no Linux/Windows: git --version

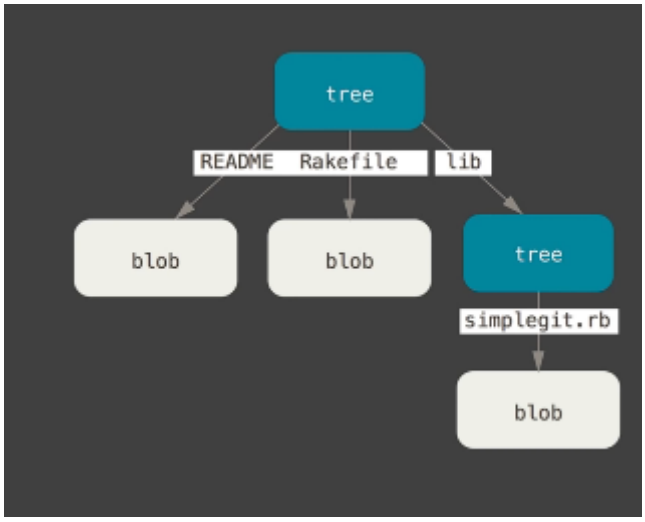
Aula 04: Tópicos fundamentais para entender o funcionamento do Git

- Entendendo como o Git funciona por baixo dos panos:
    - SHA1:
        - A sigla SHA (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 caracteres identificador de 40 dígitos.
        - Qualquer mudança no arquivo gera um novo identificador.
        - Caso o texto volte para a formatação original, o código antigo é retornado:
            - Ex.: 2b4c (sem alteração)
            - 1bc6 (primeira alteração)
            - 2b4c (volta a formatação original)
        - É uma forma curta de representar um arquivo.
        - Modo de usar: echo “ola mundo” | **openssl sha1** → mostra que queremos encriptar em sha1 o comando.
        - Como usar o Git Bash:
            - Para mudar o tema: botão direito do mouse no cabeçalho do programa > options
            - Para acessar o Git Bash direto da pasta: botão direito do mouse > Git Bash Here
            - Para encriptar um arquivo com SHA1: openssl sha1 texto.txt
    - Objetos fundamentais
    - Sistema distribuído
    - Segurança

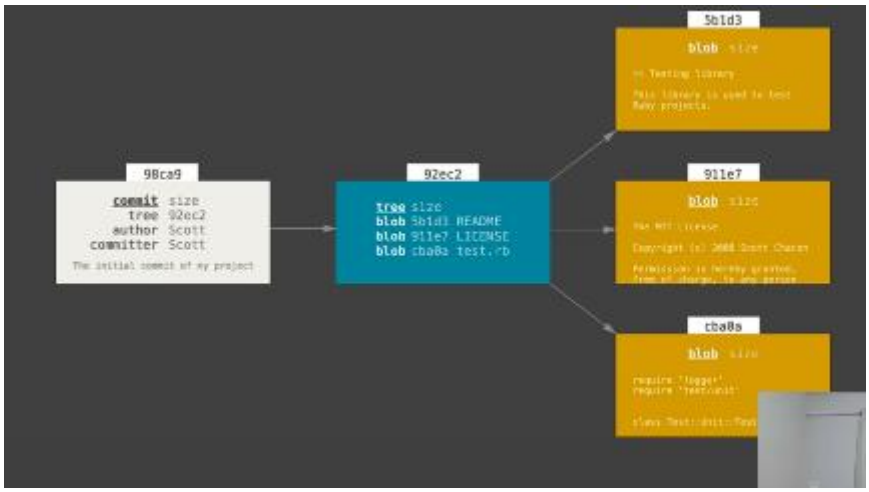
Aula 05: Objetos internos do Git

- Blobs:
    - echo ‘conteudo’ | git hash-object --stdin → devolve o sha1 do ‘conteudo’
        - hgf46e6566ie474nhg (1)
    - echo -e ‘conteudo’ | openss1 sha1
        - wsetdryfguy567879809w4vfcgvhg (2)
    - Blob tem o tipo do objeto (blob), o tamanho da string/arquivo, uma barra ao contrário com o zero (\0) e o conteúdo do arquivo.
    - echo -e ‘blob 9\0conteudo’ | openss1 sha1
        - hgf46e6566ie474nhg (1)

- Trees:
  - Trees armagenam blobs e apontam para tipos de blobs
  - Tree tem o tipo do objeto (tree), uma barra ao contrário com o zero (\0)
  - Tree guarda o nome do arquivo, diferente dos blobs
  - Tree podem apontar tanto para blobs quanto para outras trees (por causa da recursividade)
  - Tree também tem o sha1 dos metadados das trees



- Commits:
  - Objeto que vai juntar tudo, vai dar sentido a alteração que estamos fazendo
  - Commit aponta para uma arvore (tree), aponta para o ultimo commit feito (parente), aponta para um autor, aponta para a mensagem e também tem um timestamp (carimbo de tempo)
  - Mensagem e autor do commit dá significado as alterações feitas
  - Os commits também possuem um sha1 (encriptação)
  - Qualquer alteração no arquivo altera o sha1
  - Blob → Tree → Commit



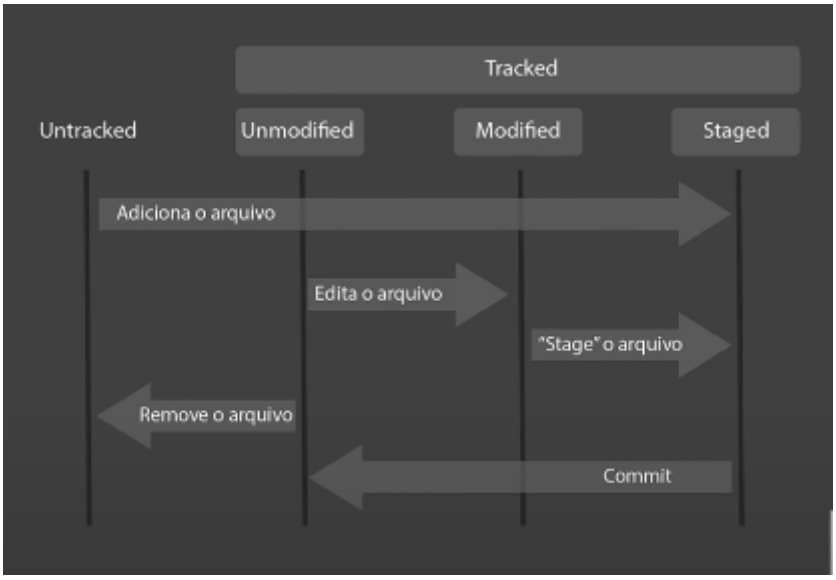
  - Git é um sistema distribuído seguro

Aula 06: Iniciando o Git e criando um commit

- Primeiros comandos com o Git:
    - Iniciar o Git: git init
    - Iniciar o versionamento: git add
    - Criar um commit: git commit
    - Criando um repositório:
        - git init (iniciar o git)
        - para ver pastas ocultas: ls -a
        - Para configurar o git pela primeira vez:
            - git config --global user.email **"**meuemail@gmail.com**"**
            - git config --global user.name **nome**
        - Adicionando um arquivo:
            - Usando markdown (forma mais humana de escrever o html)
                - # para o tamanho do texto
                - **texto** para negrito
                - _texto_ para itálico
                - - texto para lista não ordenada
                - * texto para lista ordenada
                - - [ ] para checkbox
                - ‘’’java ‘’’ para código
            - Adicionando o arquivo de fato:
                - git add * → pega tudo o que foi modificado
                - git commit -m “commit inicial”

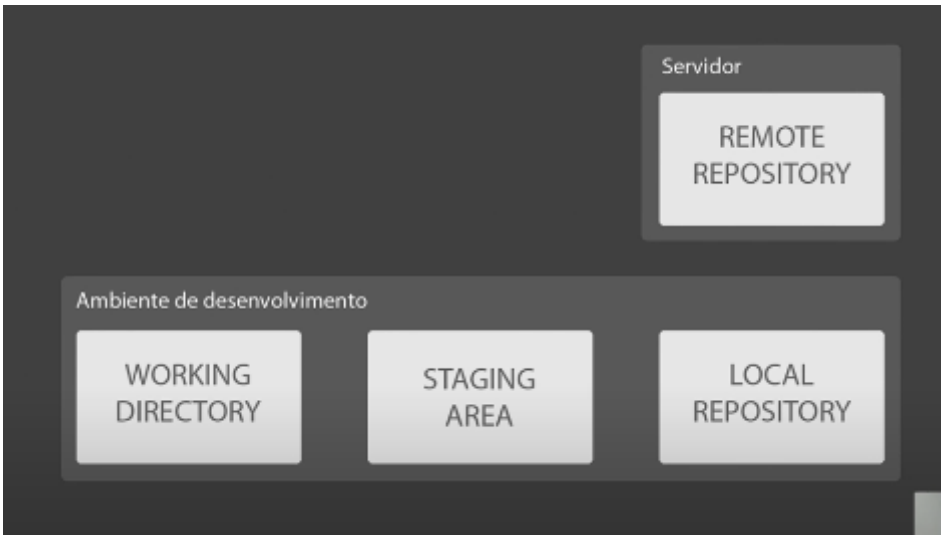
Aula 07: Passo a passo no ciclo de vida

- git init: cria um repositório no GitHub
- Tracked ou Untracked:

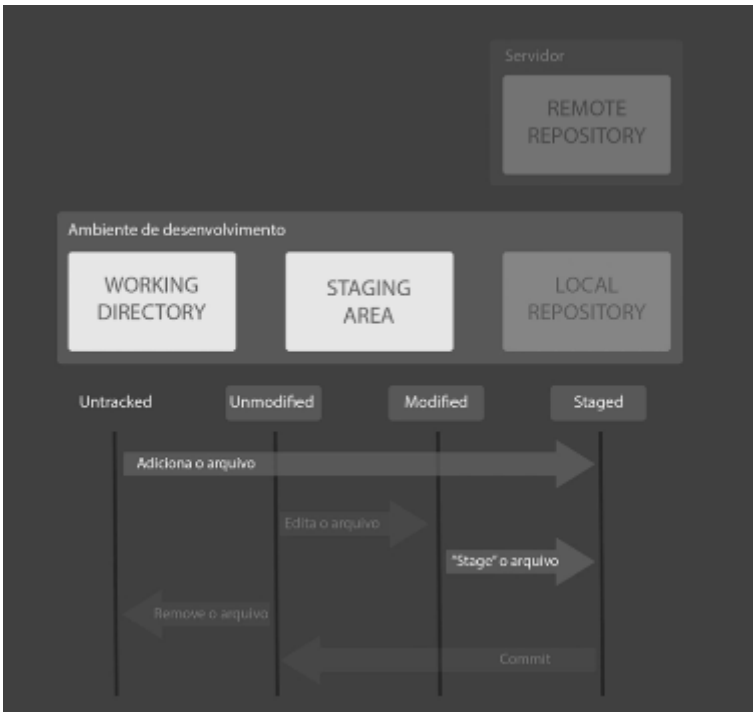


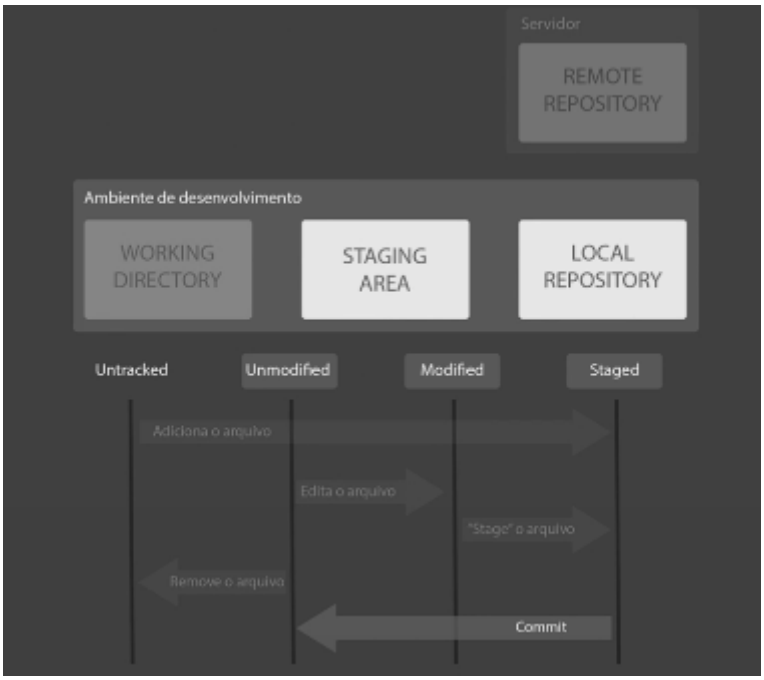
    - Tracked: arquivos que o git tem ciência deles
    - Untracked: arquivos que o git **não** tem ciência deles

- Unmodified: arquivo não modificado
- Modified: arquivo unmodified modificado
- Staged: onde ficam os arquivos que estão se preparando para fazer parte de outro tipo de agrupamento (commit)
- Processos:
    - todo arquivo quanto enviado pela primeira vez é untracked
    - git add: untracked → staged
    - unmodified p/ modified: git compara o sha1 dos arquivos
    - se rodarmos o git add de novo no arquivo modified ele vai para staged
    - arquivo unmodified p/ untracked: se removê-lo
    - staged p/ unmodified: commit
    - commit retorna todos os arquivos para unmodified pois acabou todas as alterações
- O que os repositórios significam:



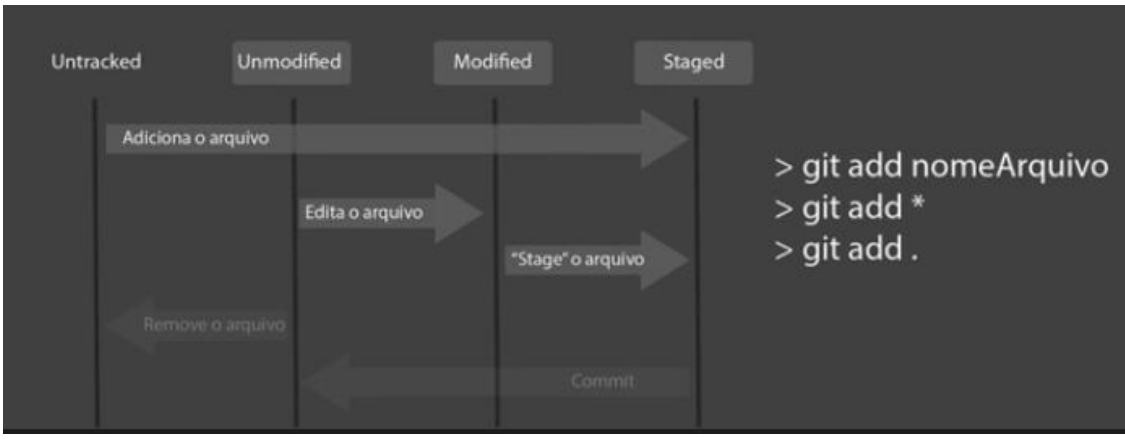
- Sistema é distribuído pois tem a versão local e do servidor
- Alteração local não repercute imediatamente no repositório remoto (para isso é necessária uma sequência de códigos específicos).
- No ambiente de desenvolvimento temos o diretório de trabalho (ex.: livro-receitas), o diretório de staging (imagem anterior à essa). Quando fazemos o commit os arquivos passam a fazer parte do repositório local e o repositório local pode ser empurrado para o repositório remoto



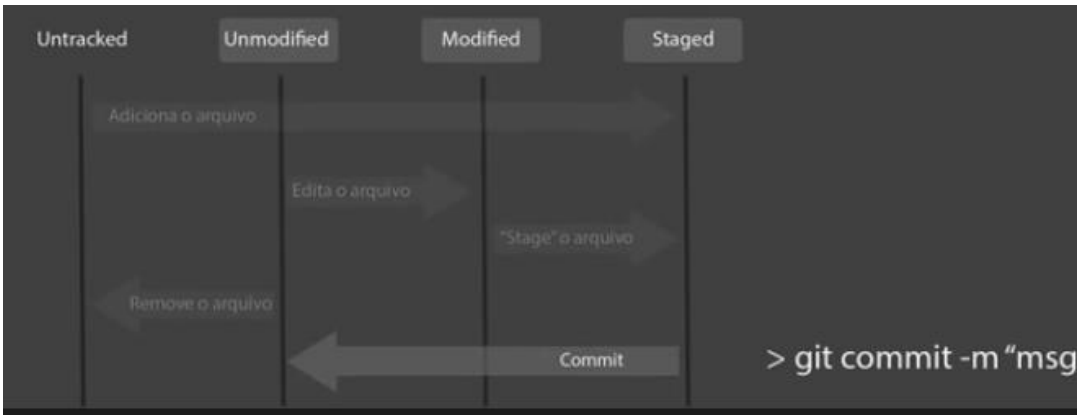


- Tudo no repositório local deve estar commitado, caso contrário não é possível enviar os arquivos do repositório local para o repositório remoto
- git status → para saber o status dos arquivos
- mv arquivo.txt ./pasta/ → para mover um arquivo para uma pasta (no git bash)
- git add arquivo.txt pasta/ → para adicionar o arquivo da pasta ao staged

- git rm arquivo.txt pasta/ → para mover o arquivo da pasta para untracked novamente



- git add . → adiciona todas as modificações do repositório local para Staging Area
- git add → move os arquivos para Staged

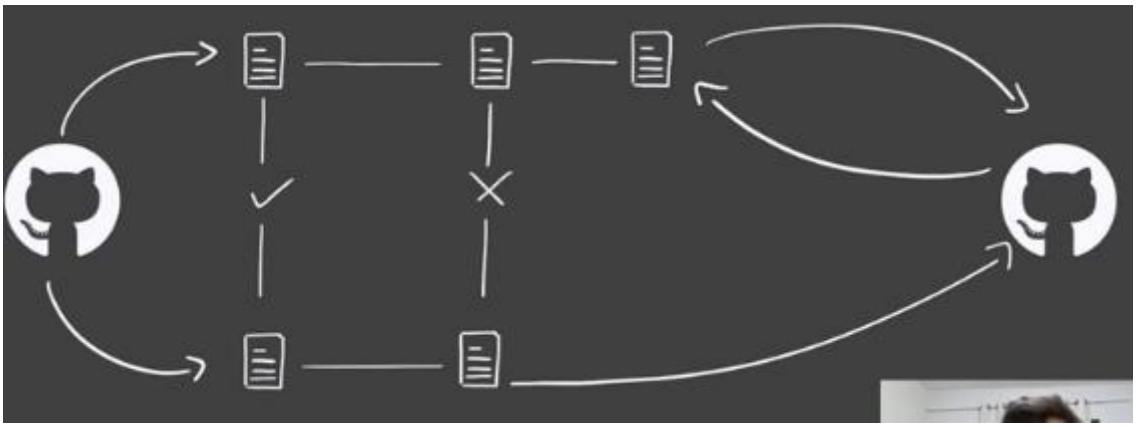


- git commit -m "msg" → move Staged p/ Unmodified

Aula 08: Trabalhando com o GitHub

- git config --list → para verificar as configurações do git
- git config --global --unset user.email → remove o email utilizado
- git config --global --unset user.name → remove o nome utilizado
- git config --global user.email "email@gmail.com" → adiciona o email ao git
- git config --global user.name "nome" → adiciona o nome ao git
- git remote add **origin** link_repositorio → adiciona a origem do envio (do repositório local para o remoto)
- **origin** é apenas um apelido
- git remote -v → lista a lista de repositórios remotos cadastrados
- git push origin **master** → para empurrar o código para o repositório remoto
- **master** é nome da branch

Aula 09: Como os conflitos acontecem no GitHub e como resolvê-los



- conflito de mergie: 2 pessoas alteram o mesmo código de maneira diferente e um sobe o arquivo antes do outro → GitHub avisa ao usuário que existe uma nova versão do código e apresenta essa versão para o usuário para que ele altere (para isso é usado o comando git pull **origin master**).
- para clonar um repositório → acessar o GitHub > clicar em Code > copiar link > **git clone link_do_repositorio**
- quando usamos o "git clone" a pasta baixada vem como um repositório (com arquivos de versionamento de código: a pasta .git)