**NOTAS DE AULA – INTRODUÇÃO AO GIT E AO GITHUB**

**Entendendo o que é Git e sua importância**

- Git é um sistema de controle de versões distribuído.

- Comandos básicos de navegação:

- Dir / cd / mkdir / del / rmdir

- tab: completa comando (facilitador)

- echo hello > hello.txt (redireciona o resultado do echo para um arquivo TXT)

**Tópicos fundamentais para entender o funcionamento do Git**

- SHA: secure hash algorithm: encriptação gera conjunto de caracteres identificador de 40 dígitos.

- Para um mesmo arquivo (conjunto de caracteres) que não foi modificado, será gerado o mesmo identificador de 40 dígitos. É assim que o GIT identifica quando um arquivo foi modificado ou não.

- Um arquivo contento os mesmos caracteres em seu conteúdo, vai gerar o mesmo código de identificação SHA1.

**Objetos internos do Git**

- Git guarda os arquivos fazendo um SHA deles, mas também acrescenta metadados.

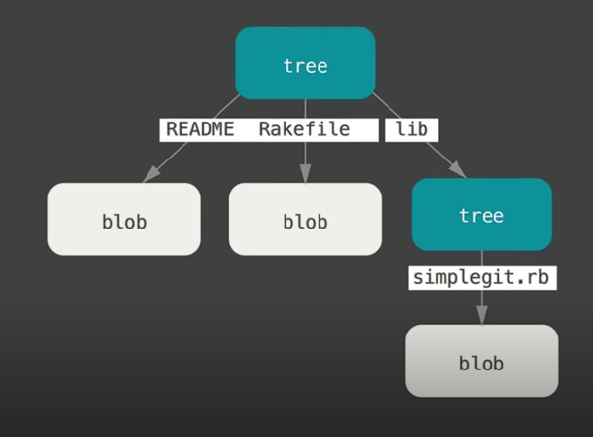
- Objetos internos do GIT responsáveis pelo versionamento de código:

**blob:** contém metadados do git dentro dele: tipo, tamanho, \0, conteúdo (*blob, tamanho 9 caracteres, \0, ola mundo*). **Não** armazenam o nome do arquivo. Tem o SHA1 dos metadados do arquivo.

Texto

Descrição gerada automaticamente

**trees:** armazenam ‘blobs’ como se fosse em uma estrutura ‘crescente’. As árvores também contêm metadados. Elas armazenam o nome do arquivo *(tree, tamanho, \0, blob, sha1 com os metadados da árvore, nome do arquivo)*.



**commits:** é o objeto que junta “tudo” e dá sentido à alteração que está sendo feita. Possui metadados *(commit, tamanho, SHA1 da tree, SHA1 do parente, autor, mensagem, timestamp)*. O parente é o apontamento para o último commit feito antes desse atual. Eles também possuem seu próprio SHA1 encriptando seus metadados, assim, uma alteração em um arquivo vai alterar o SHA1 do blob, que vai refletir na árvore, que vai refletir no commit.

Diagrama

Descrição gerada automaticamente

**Chave SSH e Token**

- Chave SSH é uma forma de estabelecer uma conexão segura e encriptada entre duas máquinas.

- Sempre há uma chave pública e uma chave privada.

- Coloca-se a nossa chave pública no repositório do Github e ele passa a reconhecer nossa máquina, nos deixando enviar código sem necessidade de nova autenticação.

- para gerar a chave acessar o git-bash e executar:

ssh-keygen -t ed25519 -C claudio.paiva01@fatec.sp.gov.br

- navegar até a pasta padrão (onde foram criadas as chaves pública e privada):

cat id\_ed25519.pub

- copiar a chave **pública** e adicionar no Github na web

- voltar no Git-bash e inicializar o ssh-agent, em seguida adicionar a chave **privada**:

eval $(ssh-agent -s)

ssh-add id\_ed25519

- a partir do momento que se tem uma chave SSH configurada na máquina não é possível simplesmente clonar um repositório copiando o link do repositório e usar o git-clone, mas deverá usar o link do caminho ssh (ao escolher como clonar o repositório no Github).

- para clonar: navegar até a pasta de destino e:

git clone ‘*caminho ssh copiado no repositório que deve ser clonado*’

**Token de acesso pessoal**

- É uma outra forma de autenticação.

- Gera-se um token no Github, guarda na sua máquina e sempre que for fazer um commit informa-se o usuário e no lugar da senha coloca-se o Token gerado.

- Ao gerar tokens no Github pode-se gerar uma data de expiração (se quiser).

- Em versões a partir do 2.30 go GitHub usa-se por padrão o Git Credential Manager Core

**Iniciando o Git e criando um commit**

- Cria-se a pasta do projeto

- Usando o Git-bash navega-se até esta pasta

- Iniciar/Criar um repositório vazio dentro da pasta atual:

**git init**

- A partir daqui o Git começa a ter possibilidade de gerenciar e versionar o nosso código

- Ao usar o Git pela primeira vez, serão solicitadas configurações iniciais de *name* e *e-mail* (usados como metadados nos objetos commit):

git config --global user.email "claudio.paiva01@fatec.sp.gov.br"

git config --global user.email Claudio

- Criar um novo arquivo na pasta do projeto e editar (digitar o código que for necessário) e, sem seguida, para adicionar o arquivo ao repositório:

**git add \***

**git commit -m "commit inicial"**

**Passo a passo no ciclo de vida**

Interface gráfica do usuário

Descrição gerada automaticamente

- **Untracked:** arquivos que o Git ainda não tem ciência deles.

- **Tracked:** arquivos já adicionados ao Git. Podem ser de 3 tipos:

- unModified: arquivo que já está **dentro do repositório do Git** e que ainda não sofreu alteração.

- modified: qualquer arquivo unModified que sofreu alteração.

- staged: todos os arquivos que estão prontos para serem enviados.

- Ao se criar um arquivo seu status é Untracked.

- Git add move arquivos untracked direto para staged e também arquivos Modified para staged.

- Quando o Git compara o SHA1 de arquivos e encontra diferença, então os arquivos mudam de unModified para Modified.

- Arquivos staged estão se preparando para fazerem parte de um commit.

- Git commit move os arquivos para unModified para reiniciar o ciclo.

- Git status permite monitorar os status dos arquivos.

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

- No ambiente de desenvolvimento local os arquivos ficam alternando entre Working Directory e Staging Area.

- Git commit -m “msg” faz com que os arquivos passem a integrar o nosso repositório local (que poderá ser empurrado para um repositório remoto).

- git add nome\_arquivo

- git add \*

- git add .

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

**Trabalhando com o GitHub**

- GitHub é uma espécie de rede social voltada a profissionais de TI cuja tecnologia que o sustenta é o GIT. É uma plataforma totalmente online onde você pode criar repositórios e hospedar neles seus projetos, colaborar com softwares open source, seguir outros(as) programadores(as) e interagir com códigos de terceiros.

git config --list

git config --global --unset name

git config --global user.name paivaclaudio

git config --global user.email “claudio.paiva01@fatec.sp.gov.br”

- commits realizados até o momento enquanto estavam sendo utilizados name ou e-mail diferentes daqueles usados para criar a conta no GitHub não podem ser alterados (alteraria o SHA do blob, tree, e toda a cadeia de objetos que estão amarrados uns aos outros).

- No GitHub cria-se um repositório (vazio) e depois vai no Git e aponta o repositório local para este repositório remoto que foi criado, cujo endereço no GitHub é uma URL (https://github.com/paivaclaudio/livro-receitas.git):

git remote add origin https://github.com/paivaclaudio/livro-receitas.git

- para ver as listas de repositórios remotos ao qual o repositório atual (caminho atual) está conectado:

git remote -v

“origin” é um alias

- Para levar o repositório local para o repositório remoto:

git push origin master

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

- Conflitos de merge podem ocorrer quando o mesmo arquivo foi modificado por mais de uma pessoa e precisa-se decidir quais alterações devem permanecer no GitHub

- Ao se empurrar (push) o repositório local e um conflito é encontrado, deve-se baixar o repositório remoto para sua máquina a fim de resolver o conflito e só então empurrar novamente.

git pull origin master

- Conflitos de merge precisam ser resolvidos manualmente, olhando-se e decidindo qual é o código correto que deve permanecer.

- Como baixar um repositório:

git clone https://github.com/python/cpython.git

- Clonar é diferente de simplesmente criar uma pasta, pois o processo de clone já cria um repositório de fato (terá uma pasta .git com todos os arquivos de versionamento).