GIT

Sistema Distribuido Seguro

Sistema de versionamento de código distribuido

Comandos Git

Windows	Unix	
cd	cd	mudar de diretório
cd ..	cd ..	voltar um nível de diretório
dir	ls	listar o que tem dentro do diretório
mkdir	mkdir	criação de diretório
del / rmdir	rm -rf	deletar diretório
cls	clear	limpar tela
tab	tab	auto completar
exit	exit	finalizar terminal
up arrow	up arrow	navegar por comandos já utilizados

Windows	Unix	
echo	echo	devolve o que escreveu no terminal

$ echo hello > hello.txt;

o “ > “ é um redirecionador de fluxo, neste caso ele vai pegar o hello e jogar dentro do arquivo hello.txt

No windows o comando **del**, não deleta o diretório de forma recursiva, só apaga o conteúdo que possui dentro.

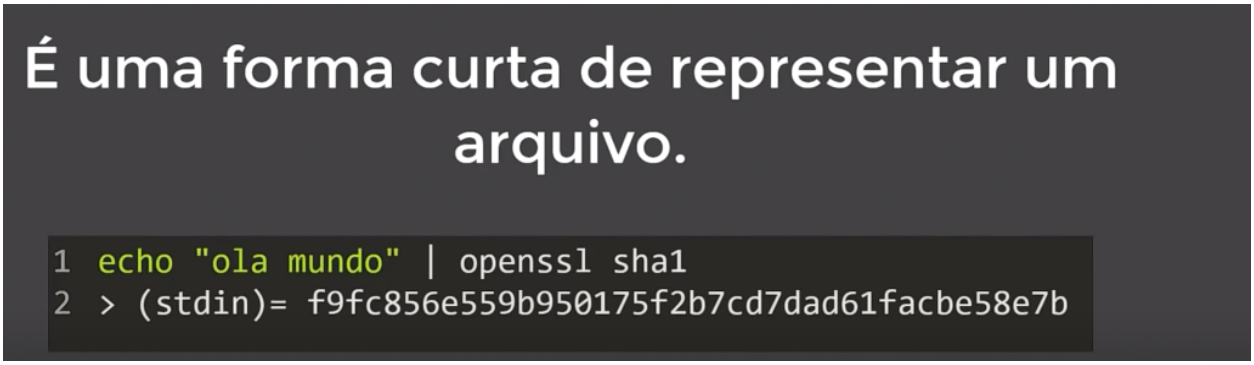
Windows: rmdir workspack /S /Q = o “s” Exclui arquivos especificados do diretório atual e de todos os subdireários. Exibe os nomes dos arquivos à medida que eles estão sendo excluídos e o “q” Especifica o modo silencioso. Não é solicitado que você exclua a confirmação.

Unix: rm -rf workspace = o “ r “ é de recursivo e o “ f “ é de forçar

Strings
são sequências de caracteres alfanuméricos (letras, números e/ou símbolos) amplamente usadas em programação.

Git por baixo dos panos

SHA1 (secure hash algorithm), conjunto de funções hash criptográficas projetadas pela NSA; A encriptação gera conjunto de caracteres identificador de 40 dígitos



$ openssl sha1 gitgit.txt ⇒ SHA1(gitgit.txt)= da39a3ee5e6b4b0d3255bfef95601890afd80709

Após modificar o conteúdo dentro do arquivo gitgit.txt

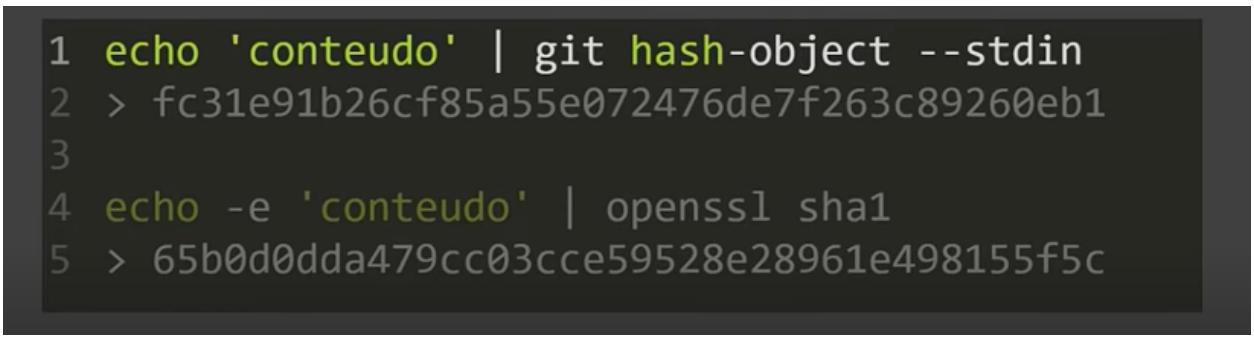
$ openssl sha1 gitgit.txt ⇒ SHA1(gitgit.txt)= f61e90f629c3539a2b1050ab63dcb8f2b2ce6504

> Toda vez que você altera algum conteúdo dentro do arquivo citado, ele gera um novo identificador único de 40 dígitos.

Caso necessite desfazer a alteração e você rode novamente o comando $ openssl sha1 {nome do arquivo}, ele mostrará novamente o mesmo identificador antes da alteração. Tornando-o assim fácil a identificação de alterações.

Objetos internos do Git

Blob = bolha



$ git hash-object = função do git

$ —stdin = flag que espera receber um arquivo.

Neste exemplo ao rodar o comando $ echo -e ‘conteudo” | openssl sha1, ele gera um novo identificador de 40 dígitos, mas porque isso acontece?

Os arquivos ficam guardado dentro do objeto “ Blob “ que contém metadados, onde vai ter o tipo do objeto, tamanho da string ou arquivo, o \0 e o conteúdo do arquivo, onde ele guarda apenas o sha1 que é o identificar deste blob.

Dessa vez ao utilizar os metadados na string no comando $ echo -e ‘blob 9\0conteudo’ | openssl sha1. o identificador vai ser o mesmo.

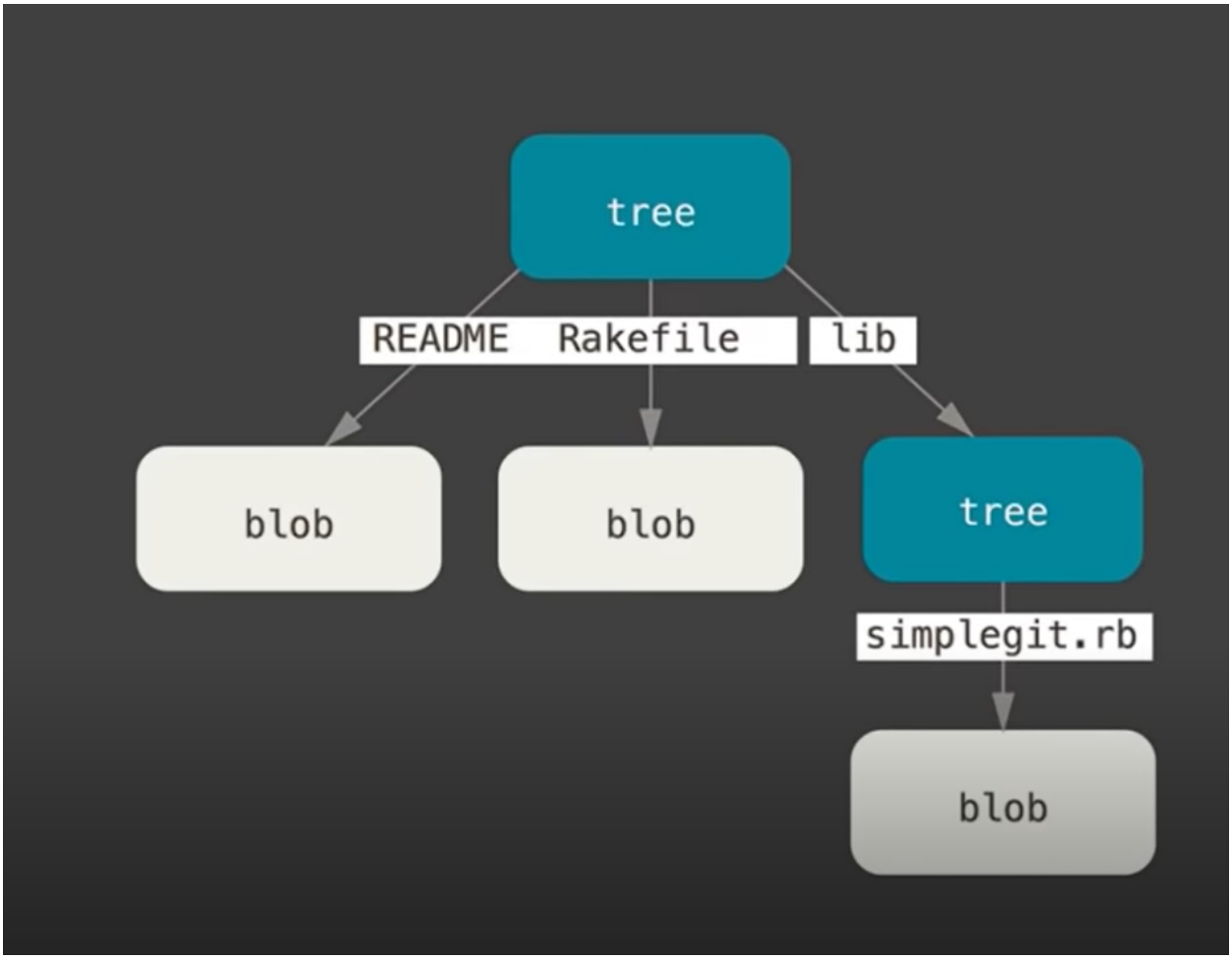
```
1 echo 'conteudo' | git hash-object --stdin
2 > fc31e91b26cf85a55e072476de7f263c89260eb1
3
4 echo -e 'blob 9\0conteudo' | openssl sha1
5 > fc31e91b26cf85a55e072476de7f263c89260eb1
```

Tree = Árvores

Contém metadados e aponta para os blob’s, armazena o nome do arquivo e sha1.

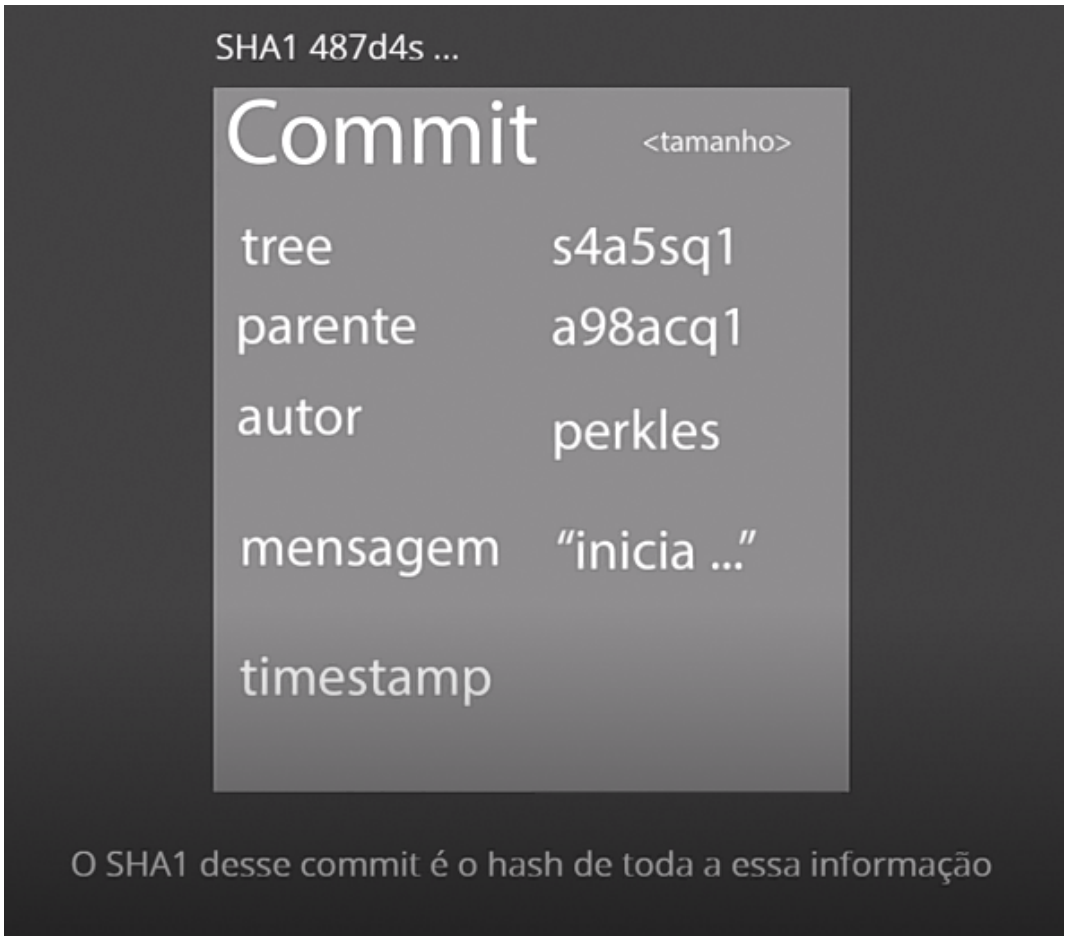
Ela é responsável por montar a estrutura de onde os arquivos estão localizados, ela aponta para blob’s ou outras Tree’s, e que também possuem sha1 desses metadados;

Obs: blob’s tem o sha1 do arquivo, a Tree apontam para essas blobs e possuem sha1 com os metadados das Tree.

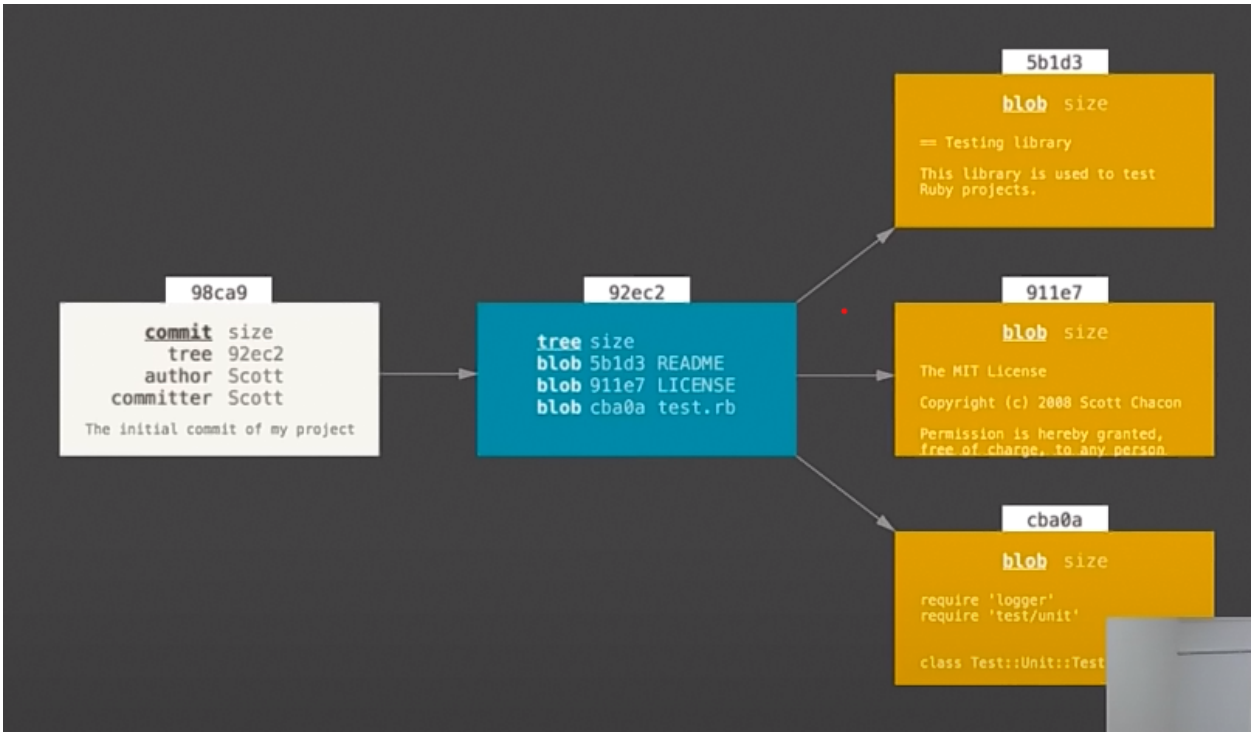


Commit

O commit significa alterações que apontam para uma tree, para parente (o ultimo commit realizado), para um autor e mensagem. Também possuem timestamp, onde leva data e hora da criação.



Possuem criptação identificador de 40 dígitos, onde alterando uma blob ele gera um sha1 da blob que possua vez altera os metadados das tree, qualquer modificação reflete nas tree e no commit por isso se torna confiável.

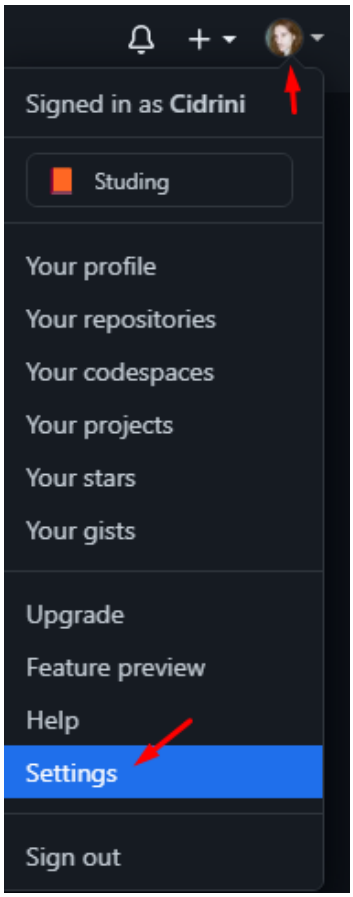


Chaves SSH

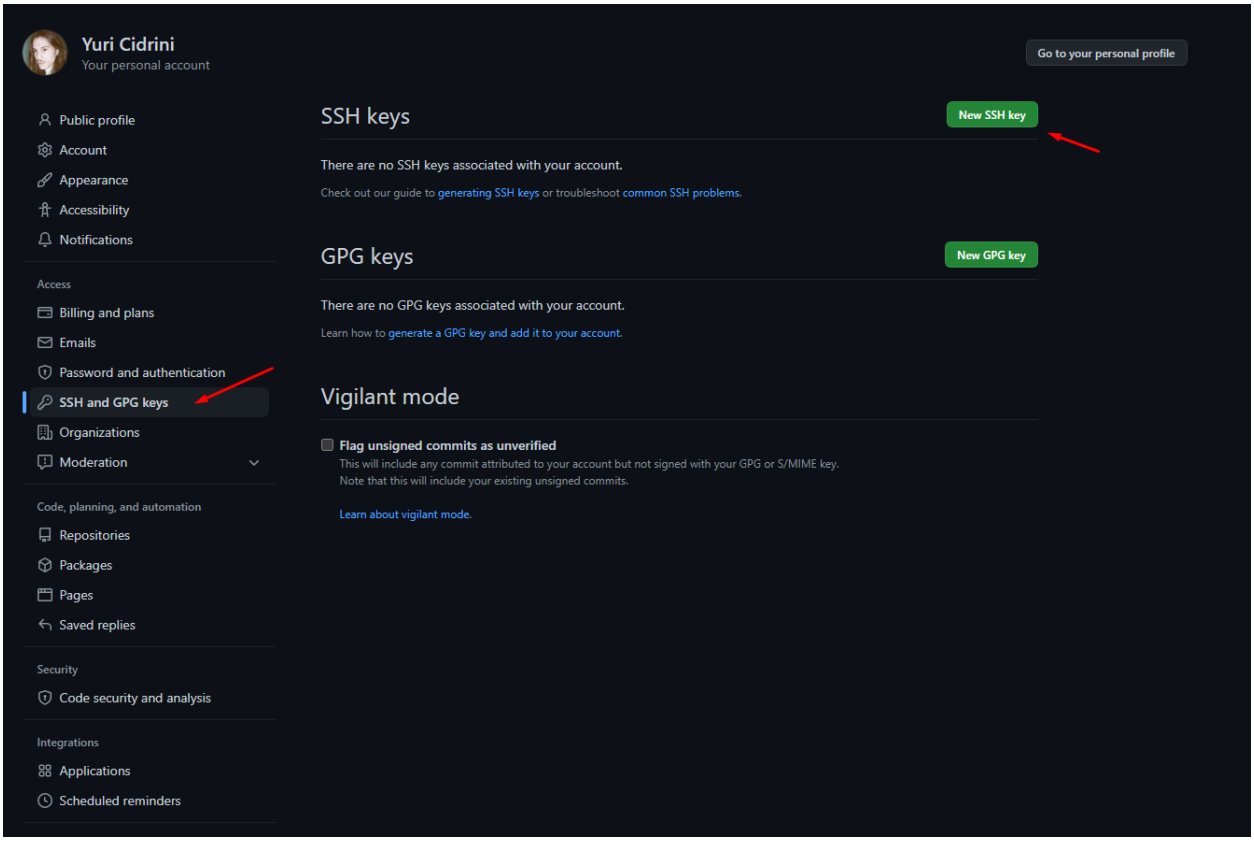
Chave SSH é uma forma segura encriptada para estabelecer uma conexão entre duas máquinas Github oferece 2 chaves, umas pública (direto pro Github) e uma privada (máquina pessoal), com elas fará o Github reconhecer nossa máquina como uma máquina confiável, assim reconhecendo todos repositorios que estiverem na nossa máquina por esse processo SSH. Então seremos capazes de enviar códigos sem precisar de senhas e autenticações.

Diretório das chaves dentro do Github:

Depois de logado no Github, clique em sua foto de perfil no canto superior esquerdo e em seguinte se encaminhe até “Settings”



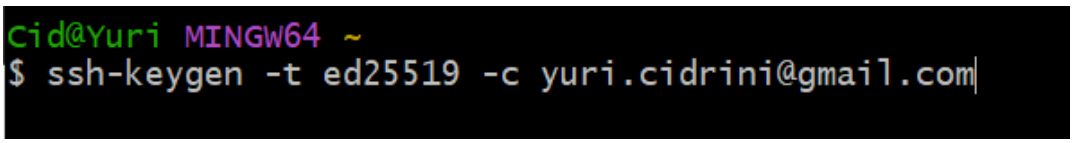
Após isso procure por SSH and GPG Key nas opções do lado esquerdo e clique em “New SSH Key”



▼ Comandos do Git Bash para gerar uma chave SSH para GitHub

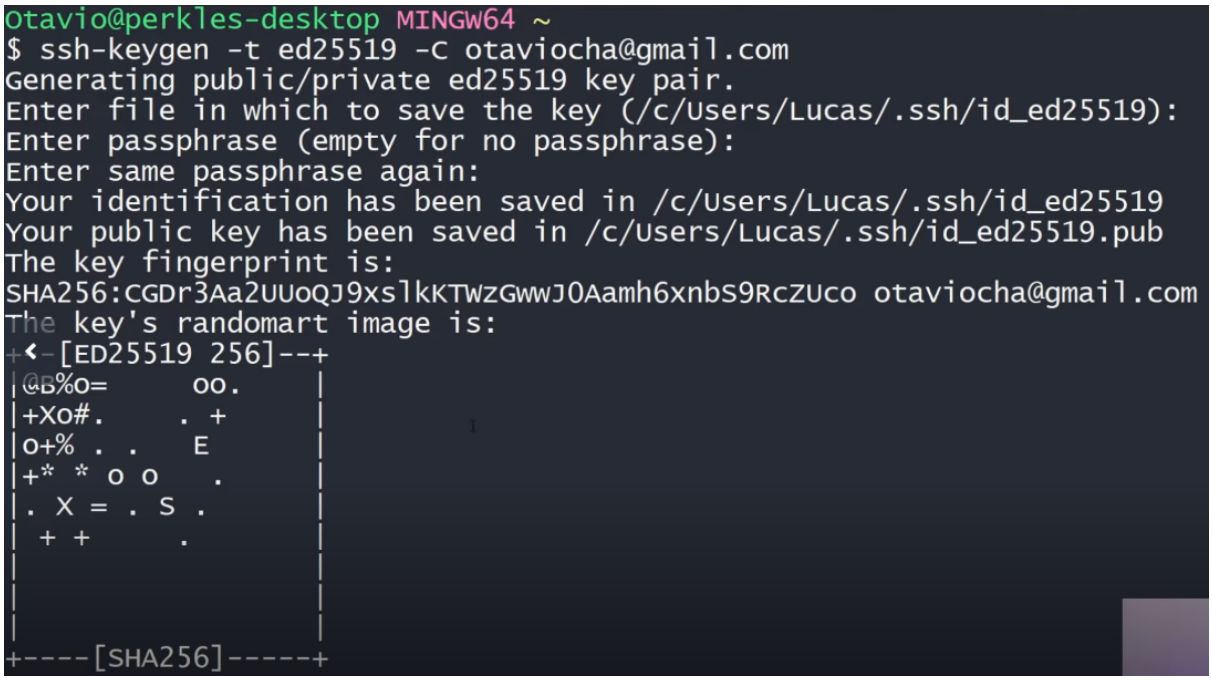
Dentro do terminal GitBash:

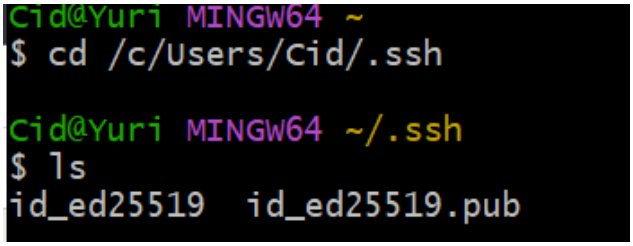
```
$ ssh-keygen -t ed25519 -C seuemail (GitHub)
ed255 = tipo de encriptografia da chave
-C precisa ser maiúsculo
```



Após presionar enter ele te mostrar onde essa Chave SSH será salva em seu PC, precione enter e ele criarar 2 chaves uma privada e uma .pub (pública)

Mostrará seu fingerprint (indentidade da chave) tipo de encriptografia, local que foi salva, nome da encriptografia, etc.

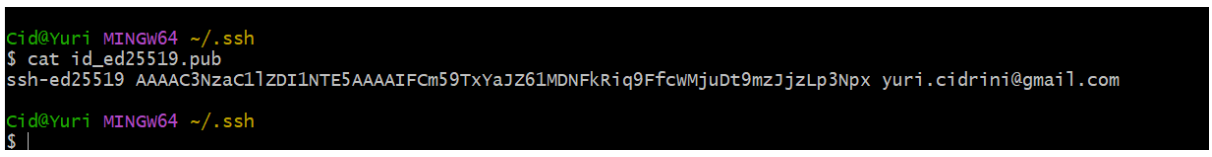




Usando o comando a gente poderá ter a acesso ao conteúdo, a chave pública que será usada no GitHub

```
$ cat id_ed25519.pub
```

id = identificação da chave e após ele o nome da chave



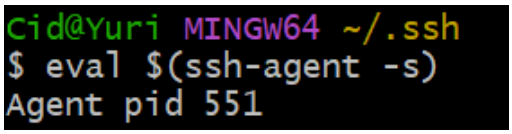
Podemos copiar nosso SSH que será usado no GitHub

Ps: Apenas enviar para o GitHub a sua chave pública (.pub)

Após colocar chave no GitHub precisamos inicializar o SSH Agent para poder ligar a maquina com as chaves do GitHub

```
$eval $(ssh-agent -s)
```

Ele te retornará seu Agent Pid que será o número do processo que estará rodando na sua maquina



Agora podemos entregar nossa chave para o Agent Pid utilizando o comando

```
$ssh-add id_ed25519
```

Lembrando que temos que passar a nossa chave privada dessa vez, já que o Agent ficará encarregado de desencriptografar a mensagem.

▼ Token de Acesso Pessoal

Terminado de criar navegaremos ate a chave usando o comando

```
$cd /c/Users/Cid/.ssh (depois do c/ deverá coloca o diretório de onde foi salva sua chave)
```

Ele listará as duas

Iniciando o git e criando um Commit

```
$ git init - inicia o git dentro do repositório criando uma pasta “ .git “ oculta dentro do repositorio
```

```
$ ls -a - mostra arquivos ocultos
```

Caso seja a primeira vez que esteja rodando o git, requer algumas configurações.

```
$ git config global user.email user.email “leonardocesar-pe2010@hotmail.com”

$ git config --global user.name user.name Leonardo
```

Commitando

```
$ git add *
$ git commit -m “mensagem”
```

```
$ git status - mostra o status do git no momento atual
```

```
Leonardo Silva@Leonardo-PC MINGW64 /c/workspace/livro-receitas (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    strogonoff.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        receitas/

no changes added to commit (use "git add" and/or "git commit -a")

Leonardo Silva@Leonardo-PC MINGW64 /c/workspace/livro-receitas (master)
$ git add strogonoff.md receitas/

Leonardo Silva@Leonardo-PC MINGW64 /c/workspace/livro-receitas (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    strogonoff.md -> receitas/strogonoff.md


Leonardo Silva@Leonardo-PC MINGW64 /c/workspace/livro-receitas (master)
$ git commit -m "criando pasta receitas, moveu arquivo para receitars"
[master 46bf025] criando pasta receitas, moveu arquivo para receitars
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename strogonoff.md => receitas/strogonoff.md (100%)

Leonardo Silva@Leonardo-PC MINGW64 /c/workspace/livro-receitas (master)
$ git status
On branch master
nothing to commit, working tree clean
```

GitHub

```
git config --global --unser user.name - remove no exemplo a configuração de nome do git
```

Adicionando origem para enviar os arquivos

```
$ git remote add origin git@github.com:LeonardoCSR/livro-receitas.git
```

Listar a lista de repositórios remoto

```
$ git remote -v
```

Comando para enviar o repositorio local para o remoto

```
$ git push origin master
```