**GIT E GITHUB**

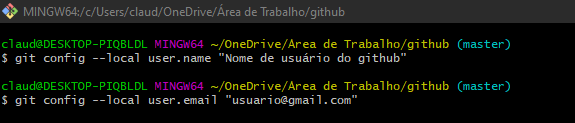
**GITHUB** é a aplicação na Web [www.github.com](http://www.github.com) aonde é possível criar sua conta e visualizar as alterações feitas no GIT.

**Git ou GitBash** é um (CMD)controlador de versionamento, serve para salvar diversas versões de arquivos em um servidor e esse arquivo pode ser acessado pelo usuário aonde vai localizar todas as versões de arquivos salvas por ele, com isso o usuário pode voltar a primeira, antepenúltima, penúltima ou “n” versão utilizada se desejar. Outra opção é que o arquivo pode ser usado em conjunto com outras pessoas, aonde os usuários que alterarem esse mesmo arquivo vão receber alertas de mudanças ao tentar salvar o arquivo no local “servidor” aonde salva todas as versões desses arquivos.

**Comandos básicos com GIT:**

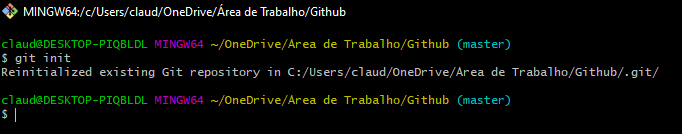
**CONFIGURANDO USUÁRIO E EMAIL LOCAL OU GLOBAL**

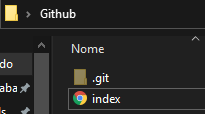
Configurar e atrelar o seu usuário e e-mail ao git instalado localmente, assim ele passará a carregar as modificações de arquivos para a sua conta informada. Para utilizar global basta trocar –local por –global



**INICIANDO O GIT EM UM REPOSITÓRIO**

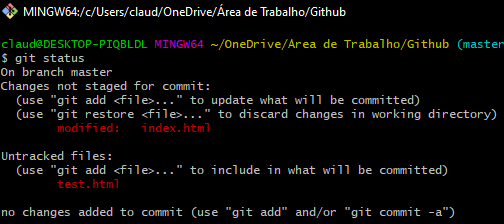
Vai inicializar o seu repositório e vai criar um subdiretório (.git) dentro do seu repositório, o processo ainda não é monitorado, mas o GIT já entende que aquele repositório pertence a ele e que vai ser utilizado.

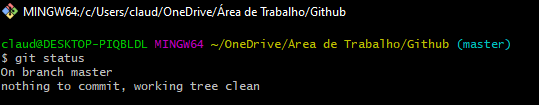




**CONSULTANDO E IDENTIFICANDO STATUS (HISTÓRICOS) DOS REPOSITÓRIOS NO GIT**

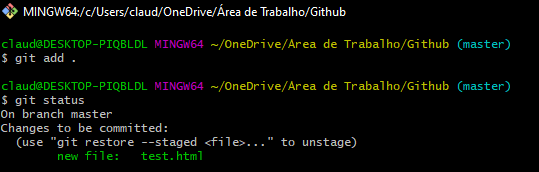
Dentro do seu diretório existem arquivos que nunca foram modificados (não-rastreados **ex:test.html**) e estão disponíveis para serem adicionados ao stage (arquivos que o GIT “vai conhecer”) pela primeira vez. E existem também arquivos que já foram adicionados ao stage alguma vez e já foram modificados (**ex:index.html**). A segunda imagem identifica que não existe nenhum arquivo para ser enviado ao stage, porque provavelmente todos os arquivos já foram adicionados.





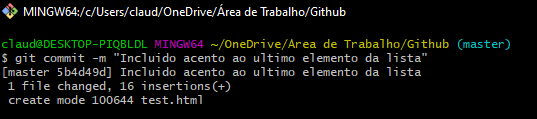
**RASTREANDO ARQUIVOS**

Utilizando o “git add .” você passa a rastrear todos os arquivos dentro do seu repositório, para rastrear um diretório específico que possa conter outros subdiretórios utilizar o “git add ./Nomedodiretório” ou arquivo “git add index.html”. Quando o arquivo passa a ser rastreado ele fica com a mensagem de “New File” no comando git status.



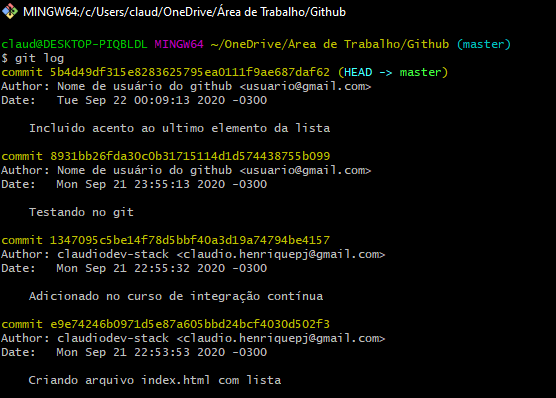
**COMMITANDO ARQUIVOS**

Após o arquivo ser rastreado ele está preparado para ser salvo.   
O “-m” é para enviar uma mensagem no arquivo que você está fazendo, com isso vai informar que o arquivo foi trocado e criado com sucesso.



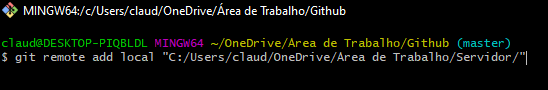
**LOG DE ALTERAÇÕES DE ARQUIVOS**

Visualiza todas as alterações feitas dizendo o Usuário, Date e cada commit é gerado um HASH code e os comentários de cada commit. Existem também comandos como “git log -p” que detalha as mudanças internas do arquivo e “git log –oneline” que detalha as descrições inseridas nos arquivos. Existem também consultas de log avançadas no link [https://devhints.io/git-log /](https://devhints.io/git-log%20/) git log --graph



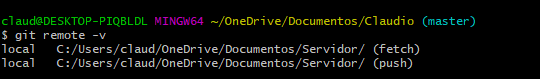
**TRABALHANDO DE FORMA REMOTA COM OS REPOSITÓRIOS**

Para adicionar repositórios remotos como por exemplo (O repositório do usuário2 ou servidor) você deve adicionar e setar o caminho desse repositório, pode ser por pasta, url, geralmente url github. PS: É necessário utilizar as “” quando o comando existe algum espaço como por exemplo /Area de Trabalho/ se o caminho não tiver espaços não é obrigatório.



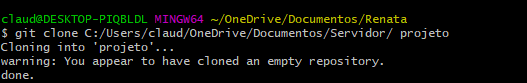
**IDENTIFICANDO REPOSITÓRIOS REMOTOS VINCULADOS**

Para identificar qual repositório remote foi adicionado a sua... digamos “conta” basta usar o comando abaixo, veja que está pronto para “buscar” e “empurrar” no repositório Servidor.



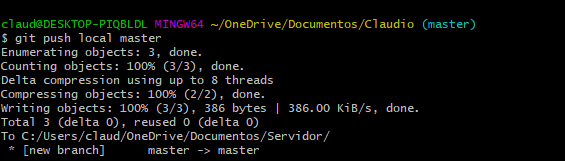
**CLONANDO UM REPOSITÓRIO REMOTO**

Quando temos um repositório já criado por outro usuário e sendo utilizado, basta clonar do caminho correto do repositório remoto dele para o seu diretório, assim todas as mudanças feitas serão clonadas. PS: Se o outro usuário não subiu as mudanças para o servidor por exemplo, talvez ele retorne com os diretórios vazios.



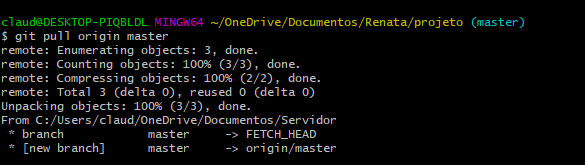
**DANDO PUSH DE UM REPOSITÓRIO**

No exemplo abaixo, estou empurrando ou enviando as mudanças feitas para o repositório remote “servidor”. Com isso é possível um outro usuário pegar as mudanças desse “servidor”, sem necessitar ser diretamente do meu remote local.



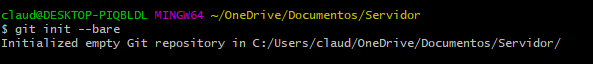
**DANDO PULL DE UM REPOSITÓRIO**

O comando abaixo busca as mudanças ou atualizações feitas nos arquivos e carrega como sendo o atual. É necessário identificar o remote que você está dando pull



**INICIANDO COM --BARE**

Com o comando git init --bare você está criando um repositório que é *pushable*. Geralmente os repositórios bare são criados no servidor e são considerados repositórios para armazenamento, em contraste aos repositórios que vão nas máquinas dos desenvolvedores que seriam os repositórios de desenvolvimento, criados com o comando git init (sem o --bare).  
Apesar do GIT ser um sistema de controle de versionamento distribuído, é muito comum que exista um repositório central que facilite a troca de informações entre os desenvolvedores, evitando a necessidade que os computadores dos desenvolvedores se comuniquem diretamente entre si.

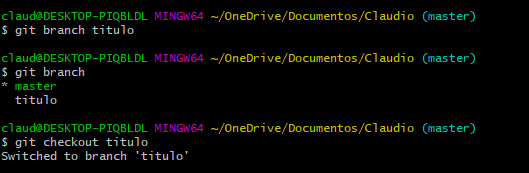
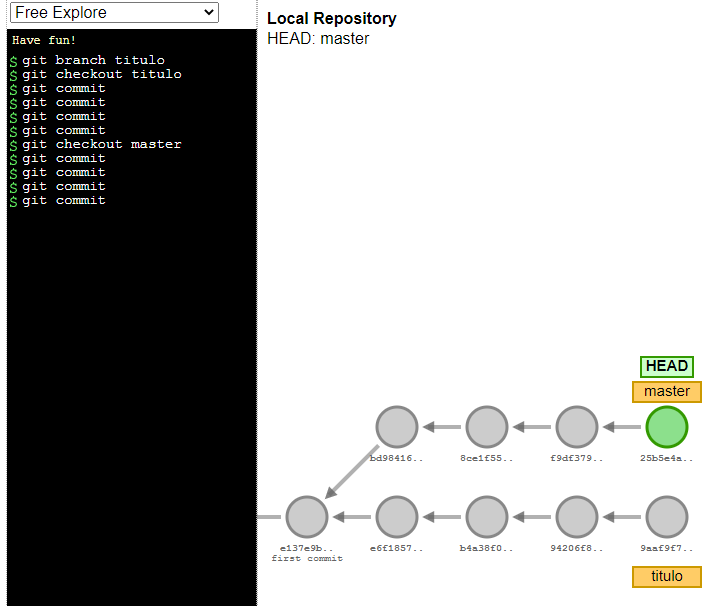


**BRANCHES**

<http://git-school.github.io/visualizing-git/>

I ideia das branches em resumo, são criar diretrizes diferentes para os usuários não utilizarem as mesmas branches, porque com isso as alterações feitas por ambos não hora de buscar as atualizações não criem conflitos umas com as outras. O link acima explica a utilização de branches de forma visual.

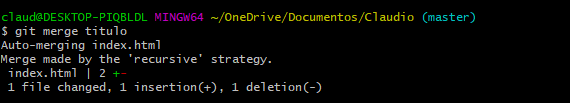
Para criar uma nova branch basta utilizar o primeiro comando, para saber quais branches existem criadas utilizar o segundo comando e para definir essa branche como principal basta utilizar o terceiro comando.

**UNINDO BRANCHES**

Após realizar as mudanças na branch (título) identifiquei uma outra pequena falha na minha branch (master), portanto eu corrigi ela e após isso com as duas branches corretas, eu fiz um merge ou seja a junção dos conteúdos e trocas nas duas branches para que tivesse um conteúdo único e final.

<https://www.treinaweb.com.br/blog/git-merge-e-git-rebase-quando-usa-los/>  
Link sobre funciona o Merge e o Rebase



**ATUALIZANDO BRANCHES**

O rebase faz é produzir um histórico linear, mais limpo e mais fácil de ser lido, pois os branches são literalmente fundidos. Pela fusão, também não é gerado aquele commit adicional “estranho” que acontece no merge.

