# **GitHub para organização dos projetos**

**Introdução:**

GIT é uma ferramenta de gerenciamento de versões (Versionamento de código / Software de controle de versão [VCS])

Usado para gerenciar códigos grandes, vários usuários utilizando o mesmo código e versões paralelas (testes e produção)

**Tipos de Sistemas de Controle de Versão:**

* Software de controle de versão - Local / Linear

Exemplos: SOURCE CODE CONTROL SYSTEM (SCCS) E REVISION CONTROL SYSTEM (RCS) – OPEN SOURCE

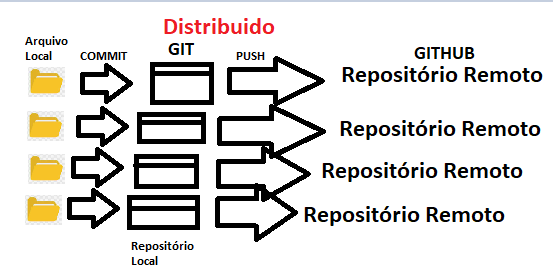
* Software de controle de versão - Centralizado / Linear (Cliente-Servidor)

Exemplos: CA SOFTWARE CHANGE MANAGER (CCC), **PERFORCE**, PANVALET, **CONCURRENT** **VERSION** **SYSTEM (CVS), APACHE SUBVERSION (SVN),** CLEAR CASE E VISUAL SOURCESAFE.

* Software de controle de versão Distribuído

Exemplos: **MERCURIAL** (DELTA), **BAZAAR** (SNAPSHOT), **DARCS** (DELTA), CODE CO-OP, GNVARCH, MONOTONE, FOSSIL, **BITKEEPER** (SNAPSHOT) E **GIT** (SNAPSHOT).





**Repositório Online/Remoto:** Pasta onde fica os códigos criados no GitHub. Site = <https://github.com>

* Login = FernandaGClemente
* Senha = Branca25102007

**REPOSITÓRIO LOCAL/OFFLINE:** Pasta onde fica os códigos criados no computador. Baixar = <https://desktop.github.com>

**PRINCIPAIS VANTAGENS DO GIT:**

* Controle de histórico
* Trabalho em equipe
* Ramificação do projeto
* Segurança
* Organização

**PRINCIPAIS VANTAGENS DO GITHUB:**

* Repositórios ilimitados
* Hospedagem de código-fonte
* Características de Rede Social
* Página GITHUB integrada
* Colaboração
* FORKS

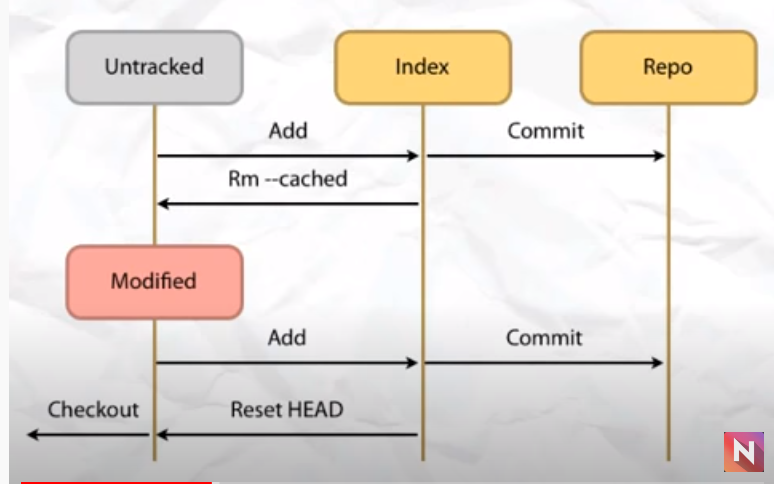
**OUTROS REPOSITÓRIOS PÚBLICOS:**

* GITLAB
* BITBUCKET
* PHABRICATOR
* GOGS
* KALLITHEA

**SISTEMA DE TRÊS ESTADOS:** Existem três estados nos quais um código pode se encontrar:

* **1° - Estado Modificado** – Indica que um arquivo sofreu alterações e que estão diferentes em relação a última versão que está armazenada no banco de dados. **SNAPSHOT ATUAL** no qual o trabalho é feito no momento. São arquivos adicionados, modificados ou removidos no diretório de trabalho (**WORKING AREA**). **(STATUS MODIFIED - Modificado/ UNTRACKED – Não rastreado)**
* **2° - Estado Preparado** – Os arquivos modificados são marcados em sua versão atual, pronta para ser armazenada no banco de dados. Os arquivos entram neste estado após uma instrução GIT ADD. Arquivos no **STAGING AREA** ou **INDEX** (Área de preparação antes de ser consolidado). (**STATUS STAGED) OBS:** Para retornar para o estado modificado, escreva: **GIT RM - - CACHED** < “Arquivo.extensão” >
* **3° - ESTADO Consolidado** – Dados são salvos no **REPOSITÓRIO** (Banco de Dados) onde o histórico é armazenado (Contendo o estado dos arquivos salvos), após a execução de uma instrução GIT COMMIT. Este estado é compartilhado. Arquivos no diretório do GIT. **(STATUS UNMODIFIED)**

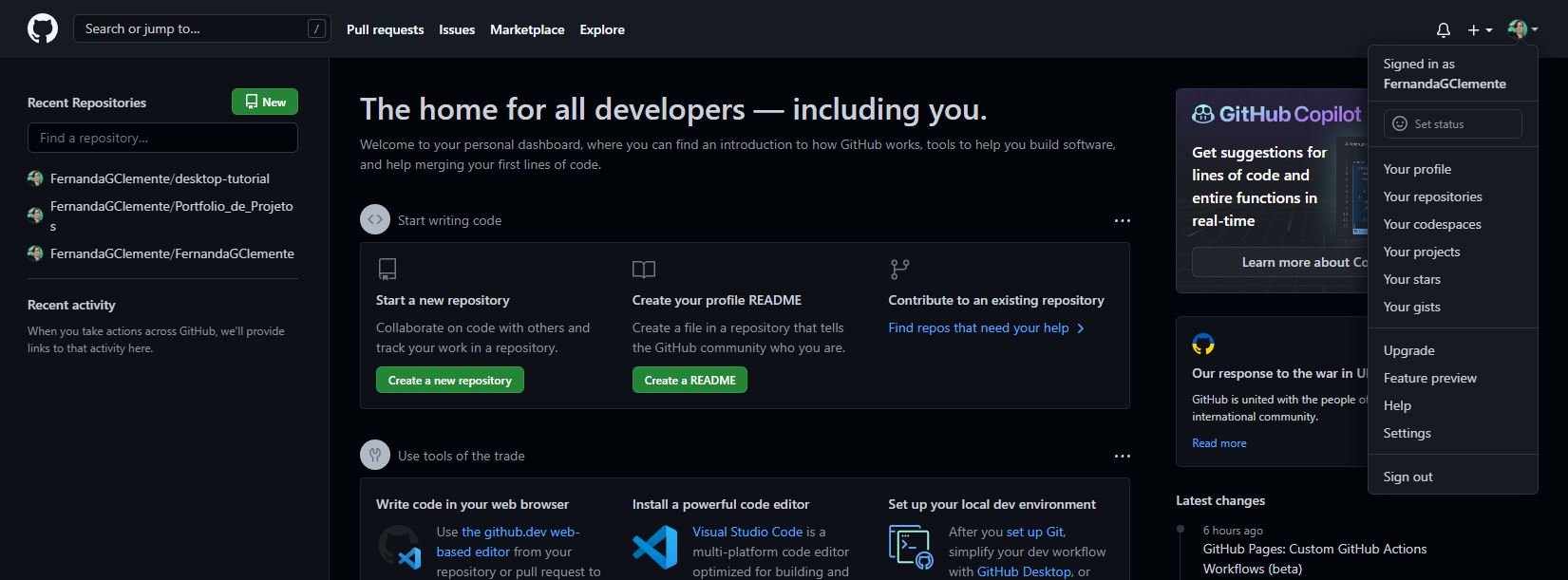
O comando **GIT STATUS + ENTER** mostra o estado no qual os arquivos se encontram no momento. (Aparecerá algumas informações, como: Quais arquivos já estão no GitHub, quais arquivos não estão e se algum arquivo foi atualizado)



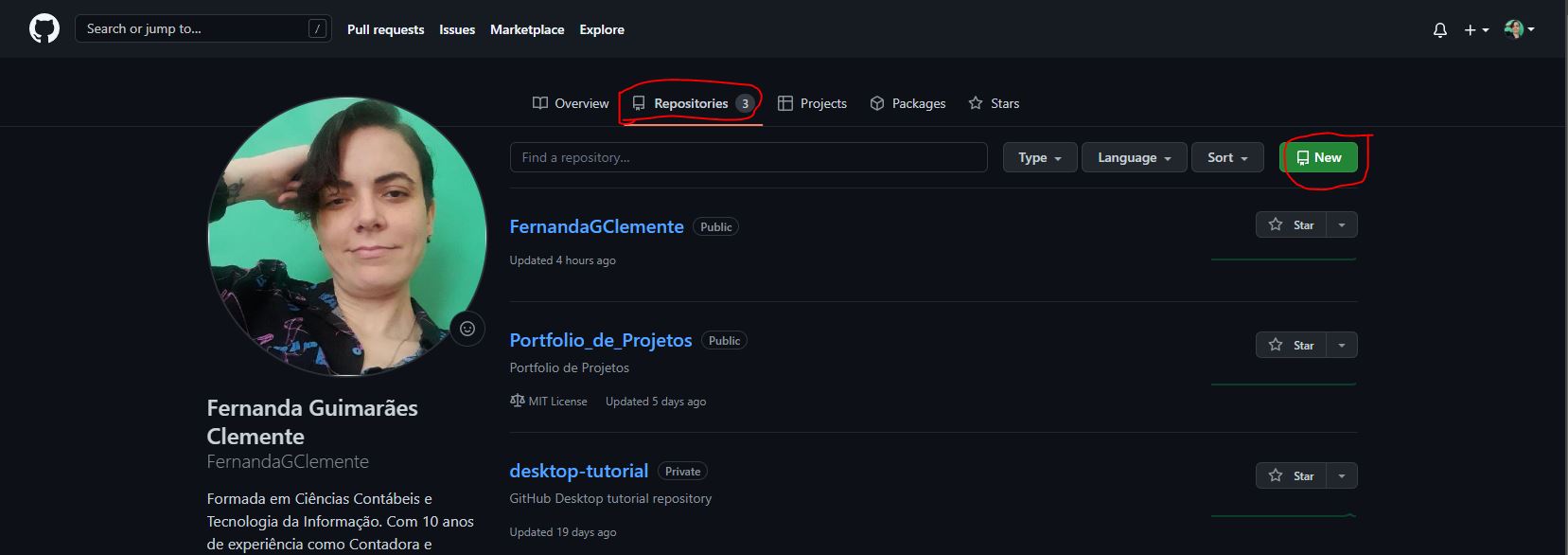
**FLUXO DE OPERAÇÃO DO GIT:**

1. Os arquivos são criados, modificados ou excluídos;
2. Arquivos que serão incluídos no SNAPSHOT são adicionados na área de STAGING;
3. Snapshot é criado;
4. Adicionar modificações ao banco de dados;
5. Um arquivo vai do estado modificado para STAGED e estão para COMMITED;

**CRIAÇÃO DO REPOSITÓRIO REMOTO NO GITHUB:**

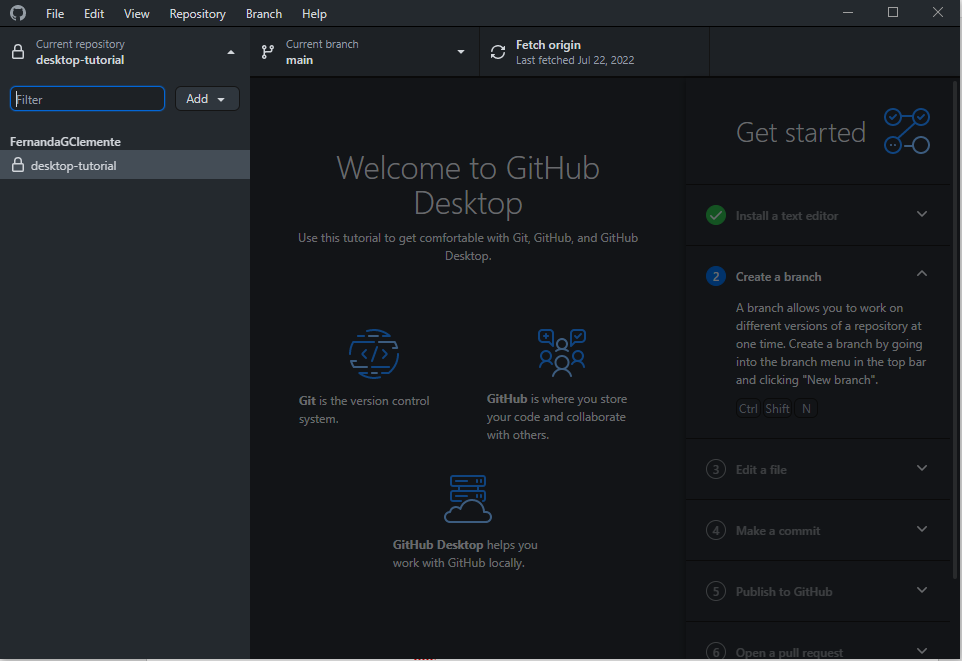


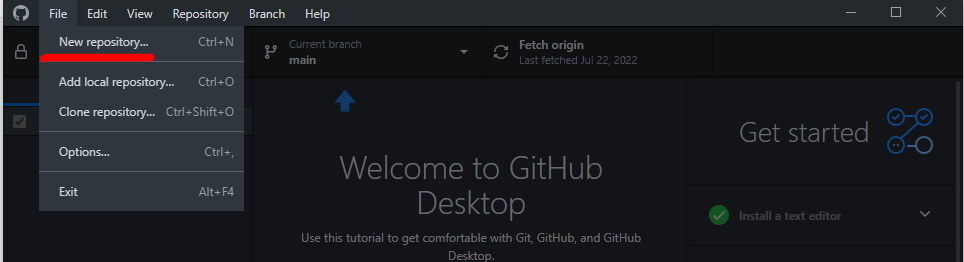
1. Entre no site do GitHub
2. Clique no PERFIL (Seta do lado da foto) 🡪 Clique em YOUR PROFILE
3. Clique em REPOSITORIES
4. Clique em NEW

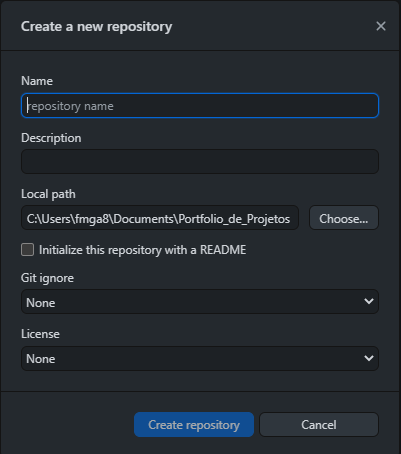


1. Informe o nome do repositório no campo “REPOSITOR NAME”, a descrição no campo “DESCRIPTION”, selecione se o repositório será Particular ou Público, marque “ADD A README FILE” e deixe como NONE os campos “ADD.GITIGNORE” e “CHOOSE A LICENSE”
2. Clique em CREATE REPOSITORY

**CRIAÇÃO DO REPOSITÓRIO LOCAL NO GIT**:







1. Clique em FILE e depois em NEW REPOSITORY. Informe o nome do repositório, uma descrição e o caminho da pasta onde os códigos estão (Clicando em CHOOSE), marque “INITIALIZE THIS REPOSITOR WITH A README” e deixe nos campos GIT IGNORE e LICENSE a opção NONE > CREATE REPOSITORY
2. Na aba HISTORY aparecerá a informação da criação do repositório local. Essa ação (processo) chamará de INITIAL COMMIT (Compromisso inicial) > Vinculou a pasta onde estão os códigos como repositório local > Será gerado dentro da pasta, 3 ou 4 arquivos:

* Uma pasta (.GIT) > Ela poderá estar oculta
* Documentos de texto (\*GITATTRIBUTES)
* Arquivo MD (Readme.md)
* Caso tenha informado alguma licença na criação do repositório (LICENSE)

1. Clique em PUBLISH REPOSITORY > Vinculou o repositório local (GIT) com o repositório remoto (GITHUB) > Desmarque “KEEP THIS CODE PRIVATE” > PUBLISH REPOSITORY

**CRIANDO UM ARQUIVO DE TEXTO QUALQUER DENTRO DO REPOSITORIO: Sintaxe:** **TOUCH** < “NomeDoArquivo.txt” >

**ESCREVENDO NO ARQUIVO.TXT CRIADO:**

**Sintaxe:** **START NOTEPAD** < “Arquivo.txt” > ou **START NOTEPAD++** < “Arquivo.txt” > ou **VIM** < “Arquivo.txt” >

**RENOMEAR UM ARQUIVO: Sintaxe: MV** < “NomeAntigoDoArquivo.extensão” > < “NomeNovoDoArquivo.extensao” >

**GIT IGNORE USANDO CÓDIGO:**

Será criado um arquivo de texto “\*GITIGNORE” e nele será listado os nomes dos arquivos que não serão adicionados no repositório online. **Sintaxe:** **TOUCH.GITIGNORE**

Para adicionar o arquivo IGNORE.txt na BRANCH, utilize o código **GIT ADD \*** e para fazer um COMMIT para gravar a adição do arquivo (**GIT COMMIT -M** “TituloDaVersão”). Para finalizar, faça um PUSH (**GIT PUSH**) para salvar no repositório online/remoto.

* Se for uma pasta, escreva dentro do arquivo de texto: NomeDaPasta/ pule uma linha e informe “Arquivo.extensão” e assim por diante.
* Editar o arquivo.txt (\*GITIGNORE), usando o editor VIM/VI \*GITIGNORE
* Caso queira adicionar um arquivo que está listado no GIT IGNORE, escreva: **GIT ADD** < “Pasta/Arquivo.extensão” > **-F**

**EDITAR O README.md USANDO CÓDIGO: Sintaxe: VIM** README.md **+ ENTER**

**Aperte a letra i** (Aparecerá INSERT na parte inferior) > Edite o arquivo > Para sair do modo edição, aperte **ESC : W Q + ENTER**

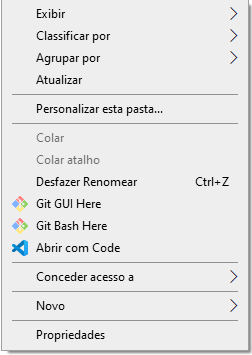
**COMO FAZER UM REPOSITÓRIO USANDO CÓDIGO:**

No diretório onde ficará o repositório dos códigos, clique com o botão direito do mouse. Abrirá um menu, clique em **GIT BASH HERE**.

Abrirá um prompt de comando e escreva o código abaixo para criar a pasta onde ficará os códigos do projeto:

**Sintaxe:** **MKDIR** < “NomeDaPasta” >

**OBS:** O comando **CD** < “NomeDaPasta” > é usado para ir até uma pasta específica.



**EXCLUIR A PASTA DO REPOSITÓRIO CRIADO: Sintaxe:** **RM -RF** < “NomeDoRepositório/” > ou < . git/ >

**CONFIGURAR O GIT NO COMPUTADOR:**

**Sintaxe:**

**GIT CONFIG...- -GLOBAL USER.NAME** < “NomeCompleto” > **+ ENTER**

**GIT CONFIG...- -GLOBAL USER.EMAIL** < “LoginDoGitHub” > **+ ENTER**

**GIT CONFIG...- -GLOBAL CORE.EDITOR** < “VIM, S ou EMACS” > **+ ENTER** (Por padrão é o VIM)

**GIT CONFIG...- -GLOBAL ALIAS.** “NOME\_ATALHO” < “Comando” > **+ ENTER**

**GIT CONFIG...- -GLOBAL HELP.AUTOCORRECT 1**

**GIT CONFIG...- -GLOBAL HELP.AUTOCORRECT TRUE**

**GIT CONFIG...- -GLOBAL INIT.DEFAULTBRACH** <“NomeNovoDaBranch”> (Mudar de Master para Main, por exemplo)

Para verificar qual é o nome da Branch > GIT CONFIG INIT.DEFAULTBRANCH

* **Acessar as configurações do GIT CONFIG, usando o USER.NAME ou USER.EMAIL ou - - LIST ou –SHOW-ORIGIN**
* **Acessar todas as configurações do GIT CONFIG, usando CAT ~/ . GITCONFIG**

**COMO INICIALIZAR UM REPOSITÓRIO:**

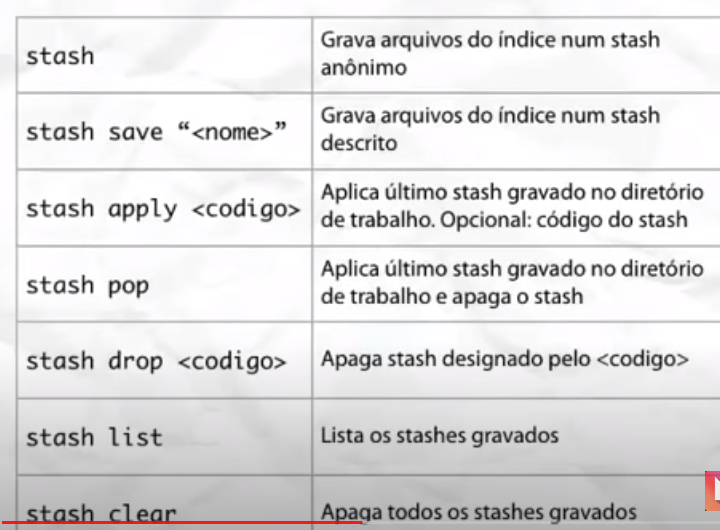
Abra o prompt e escreva: **Sintaxe:** **GIT INIT** **+ ENTER** ou **GIT INIT \*** **+ ENTER** (O ponto reforça a inicialização) Será criado uma pasta chamada “\*GIT” (Ela poderá estar oculta)

**Linguagem MARKDOW:** Usadanos arquivos README.md, ISSUES.md e PULL REQUEST.md

**Pausar as mudanças nos códigos e salvar numa área temporária,** **usando GIT STASH:**

Ainda não quer adicionar as mudanças no código e com isso “guarda” as alterações numa área temporária:

* **GIT STASH** > Guarda as mudanças no código (O arquivo modificado)
* **GIT STASH SAVE** <“NomeParaStash”>> Guarda as mudanças no código (O arquivo modificado), porém usa um identificador
* **GIT STASH APPLY** > Retorna com as mudanças no código (O arquivo modificado)
* **GIT STASH APPLY** <“N°doStash”> > Retorna com as mudanças no código (O arquivo modificado)
* **GIT STASH POP** > Retorna com a última mudança no código (O arquivo modificado) e apaga o STASH
* **GIT STASH DROP** <“N°doStash”> > Apara todos os arquivo modificados do STASH informado.
* **GIT STASH LIST** > Mostra a lista de todos os arquivos modificados e as mudanças nos códigos guardadas
* **GIT STASH CLEAR** > Limpa todos os arquivos modificados e as mudanças nos códigos guardadas



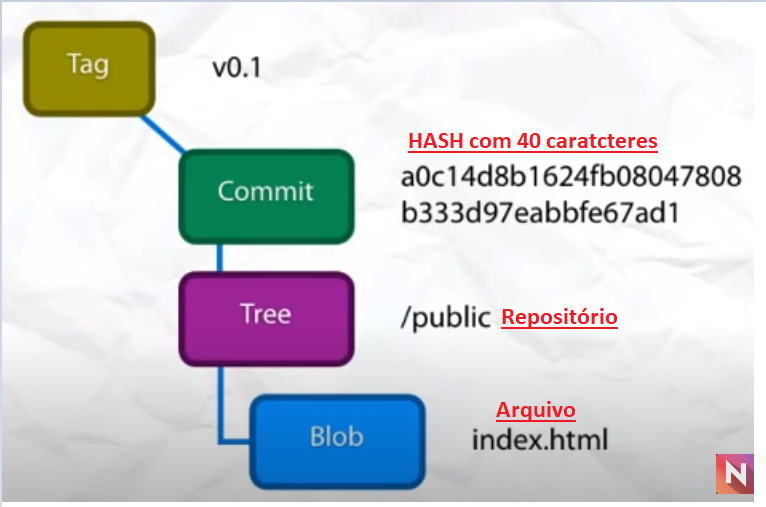
**PROTEGENDO O MASTER:**

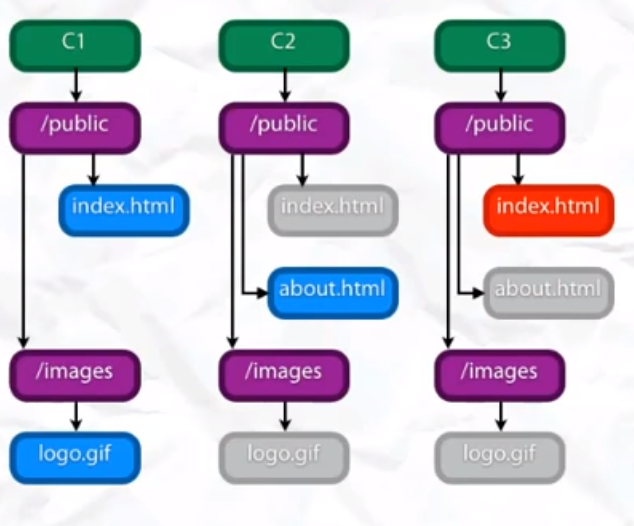
Não permitir que o usuário faça um PUSH direto no MASTER só consegue fazer o PUSH para MASTER com PULL REQUEST.

Dentro do repositório do GITHUB > SETTINGS > BRANCHES > ADD RULE > Marque as opções desejadas e informe um nome para a restrição (Regra)

**COMMIT:**

Snapshot do projeto em um determinado momento, com informações sobre o autor do conteúdo e quem o realizou. O estado anterior do projeto é denominado “PAI”. COMMITS são ligados entre si por conexões PAI-FILHO. Conjunto de COMMIT relacionados entre si por paternidade é chamado de BRANCH (Versões). Um COMMIT pode ter dois pais, se for criado pela união de dois BRANCHES (MERGES). Identificamos um COMMIT pelo seu nome, que é uma STRING de 40 caracteres obtidas pelo HASH (ALGORITMO SHA1 que dá ao COMMIT essa propriedade de único). **Cada COMMIT tem HASH exclusiva.**

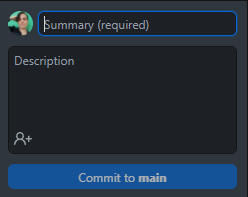




**ALTERAÇÃO OU ADIÇÃO NA BRANCH:**

Faça as alterações e as adições dos arquivos dentro do repositório:

1. No GIT Local, na aba CHANGES (Mudança) aparecerá as alterações ou as adições. Informe no SUMMARY (REQUIRED), um nome para a versão dos arquivos/códigos e na DESCRIPTION, informe detalhadamente o que foi adicionado ou alterado. Se for a inclusão do 1° arquivo, escreva “DEPLOY INICIAL” > Clique em COMMIT TO MAIN.
2. Clique em FETCH ORIGIN (Buscar na origem) > No painel aparecerá a informação que houve alterações no remoto > Clique em PUSH ORIGIN (Empurrar para origem) > “Empurra” os arquivos do repositório local (GIT) para o repositório remoto (GITHUB).



**COMO ADICIONAR ARQUIVOS AO CONTROLE DE VERSÃO:** **Sintaxe:**

(ou) **GIT ADD** < “NomeDoArquivo.extensão” > **+ ENTER** (Adicionar apenas um arquivo)

(ou) **GIT ADD \* + ENTER** (Adicionar vários arquivos)

**GIT COMMIT -M** “TituloDaVersão” **+ ENTER** (Grava a adição dos arquivos no repositório local)

**VINCULANDO O REPOSITÓRIO ONLINE COM O LOCAL:**

**Sintaxe: GIT REMOTE ADD** < “NomeReporitórioRemoto” > < “LINK\_URL\_REPOSITÓRIO” > **+ ENTER**

Exemplo: git remote add origin https://github.com/FernandaGClemente/NomeRespositorioCriado + ENTER

**Como enviar alterações PARA o repositório online:**

**Sintaxe:**

**(ou) GIT PUSH... - - SET -UPSTREAM** < “NomeReporitórioRemoto” > < “BRANCH” >

(ou) **GIT PUSH** < “NomeReporitórioRemoto” > < “BRANCH” >

Exemplo: git push – set -upstream origin master ou git push origin master

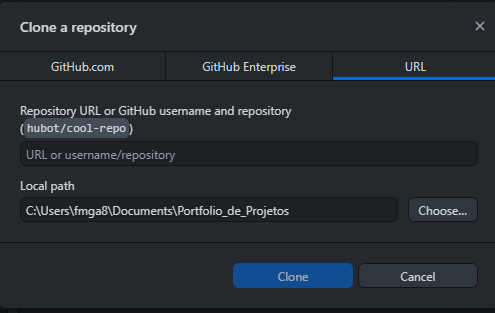
**OBS:** Faz uma busca no repositório local se tem algum arquivo ou alteração que não tenha no repositório remoto:

**Sintaxe:** **GIT FETCH** < “NomeReporitórioRemoto” > < “BRANCH”>

Exemplo: git fetch origin master

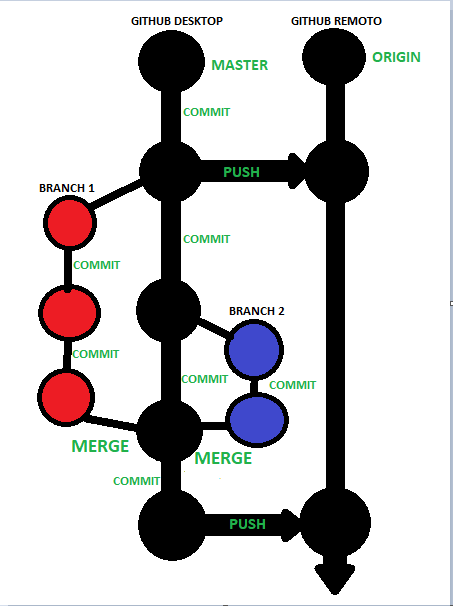
**CLONANDO OU IMPORTANDO UM REPOSITÓRIO:**

* Abra o GITHUB e escolha o repositório que quer importar
* Clique em CODE > Copie o link ou clique em “OPEN WITH GITHUB DESKTOP” para copiar o link
* No Desktop, clique em FILE > CLONE REPOSITORY > Cole o link em “URL OR USERNAME/REPOSITORY” e clique em CHOOSE para selecionar o local onde o repositório importado ficará salvo em “LOCAL PATH” > Clique em CLONE.



**CLONANDO COM CÓDIGO:**

* Abra o GITHUB e escolha o repositório que quer importar
* Clique em CODE > Copie o link ou clique em “OPEN WITH GITHUB DESKTOP”
* No prompt, escreva: **Sintaxe: GIT CLONE** < “LINK\_URL\_REPOSITÓRIO” > < “NomeDaPastaDoNovoRepositório” >



**cópia do próprio repositório:**

**Sintaxe: GIT CLONE** < “NomeDaPastaDoRepositorio/.git” > < “NomeDaPastaDaCopiaRepositório” >

**Visualizar as informações do repositório original no repositÓrio copiado:**

**Sintaxe: GIT REMOTE SHOW ORIGIN**

**ENVIANDO AS ALTERAÇÔES DE UMA BRANCH DE UMA CóPIA PARA O REPOSITóRIO REMOTO/ORIGINAL:**

**Sintaxe:**

**No repositório da cópia > GIT PUSH ORIGIN** < “NomeDaBranch” >

**No repositório remoto/original, dentro da BRANCH > GIT REMOTE ADD ORIGIN \*\*/ <** “NomePastaCópiaDoRepositório” >**/.git**

**No repositório remoto/original, dentro da BRANCH > GIT PULL ORIGIN** < “NomeDaBranch” >

**EXCLUIR A COPIA DO REPOSITÓRIO:**

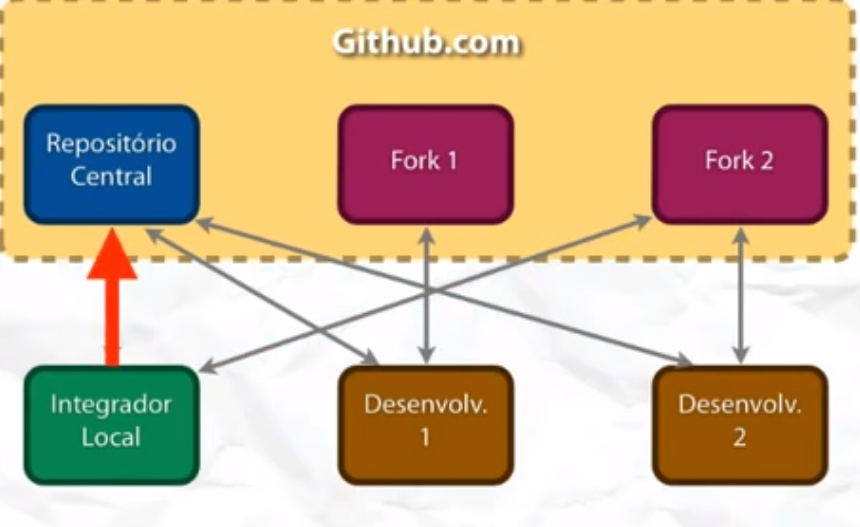
**Sintaxe:**

**GIT REMOTE RM ORIGIN** (Desconectando com o repositório remoto)

**RM -RF** < “NomeDaPastaDaCópiaRepositório” > (Excluindo a cópia do repositório)

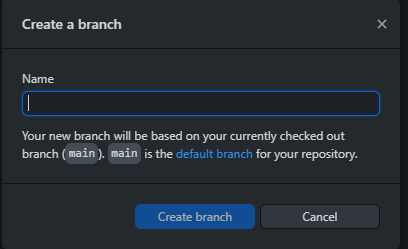
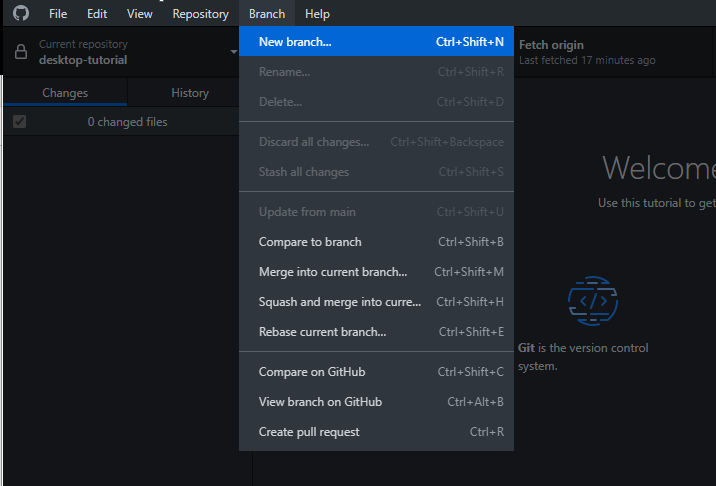
**FORK de um projeto:**

Copie um projeto, altere o código e depois envie PULL REQUEST no GitHub, dentro do repositório que tem o projeto, clique em FORK e depois clique em uma organização que queira que o projeto seja salvo.



**BRANCHS (Ramos/Galhos):** São versões em paralelas (Cópias). MASTER é o nome da BRANCH principal.

No GitHub Desktop, clique em BRANCH e depois em NEW BRANCH > Informe um nome da BRANCH e depois clique em CREATE BRANCH > PUBLISH BRANCH



**Selecionar**: Escolha qual a BRANCH vai alterar clicando em CURRENT BRANCH e selecionando-a com o mouse.

**COMO CRIAR VERSÕES DO CÓDIGO: Sintaxe: GIT BRANCH** < “NomeDaBranch” > **+ ENTER**

**Definir para qual versão será enviado os arquivos alterados QUANDO NÃO FOI FEITO O COMMIT:**

**Sintaxe: GIT CHECKOUT** < “NomeDaBrancheDefinida” > **+ ENTER**

Permite mudar e criar uma nova BRANCH com base em outra BRANCH, COMMIT ou TAG:

**GIT CHECKOUT -B** < “NomeDaBrancheOrigem” > < “NomeDaBrancheNova” > **+ ENTER**

**GIT CHECKOUT -B** < “NomeDaBrancheOrigem” > < “HashDoCommit” > **+ ENTER**

**GIT CHECKOUT -B** < “NomeDaBrancheOrigem” > < “NomeDaTag” > **+ ENTER**

**VERIFICAR QUAIS BRANCHES O REPOSITÓRIO TEM: Sintaxe: GIT BRANCH + ENTER**

OBS: A BRANCH que estiver sendo utilizada (ativa) estará com \* na frente do nome da BRANCH

**Como navegar entre versões do seu código DEPOIS QUE FOI FEITO COMMIT:**

Usar os RESET nas BRANCH e o REVERT no MASTER.

**Sintaxe:**

Exclui tudo até a versão informada → **GIT RESET/REVERT...- - HARD** < “N°daIDdaVersão” > **+ ENTER**

Exclui tudo até a versão informada → **GIT RESET/REVERT...- - HARD** < “N°daSHA” > **+ ENTER**

Exclui tudo até a versão informada → **GIT RESET/REVERT...- - HARD** < “TítuloDaVersão” > **+ ENTER**

Exclui apenas o COMMIT feito → **GIT RESET/REVERT...- - MIXED** < “N°daHashDoCommitAnterior” > **+ ENTER** (As alterações ficam em pausa)

Exclui apenas o COMMIT feito → **GIT RESET/REVERT...- - SOFT** < “N°daHashDoCommitAnterior” > **+ ENTER** (As alterações ficam em pausa)

Exclui apenas o COMMIT feito → **GIT RESET/REVERT...HEAD~1** **- - HARD** **+ ENTER**

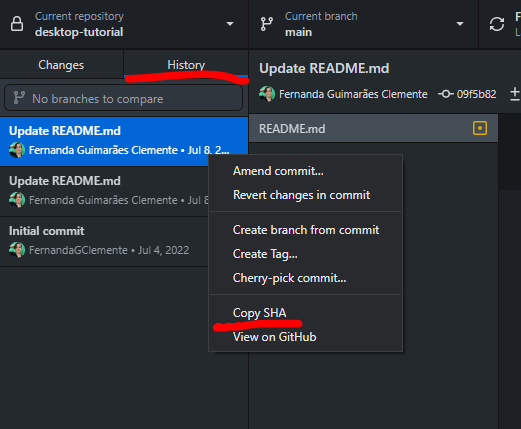
Exemplo:

|  |
| --- |
| N°doID (HEAD→<BRANCH>, < “NomeReporitórioRemoto” >/<BRANCH>) HEAD@ {0}: COMMIT: “TituloDaVersãoAtual” |
| N°doID HEAD@ {1}: COMMIT: “TituloDaVersão3” |
| N°doID HEAD@ {2}: COMMIT: “TituloDaVersão2” |
| N°doID HEAD@ {N}: COMMIT (inicial): “TituloDaVersãoInicial” |

**OBS:** Descobrir o número da SHA > No GitHub Desktop, na aba histórico, clique com o botão direito do mouse na versão que deseja voltar. Aparecerá o menu, clique em COPY SHA.

Para confirmar a “exclusão”, escreva: **GIT PUSH -F** < “NomeReporitórioRemoto” > <BRANCH> **(**Cria um novo COMMIT revertendo o COMMIT errado)

**OBS**: Reverte a alteração feita no COMMIT desejado. Apaga a alteração, mas não mexe no COMMIT todo (Não reseta)



**EXCLUIR UMA VERSÃO:**

**Sintaxe:**

(ou) **GIT BRANCH -d** < “NomeDaBranch” > **+ ENTER**

(ou) **GIT BRANCH -D** < “NomeDaBranch” > **+ ENTER** (Apaga de forma forçada)

(ou) **GIT PUSH** < “NomeReporitórioRemoto” >**:** < “NomeDaBranch” > **+ ENTER**

**PULL REQUEST (“Pedido para puxar” – Importar os arquivos para o repositório):** Processo para mesclar as duas versões trabalhadas em paralelas para ficarem iguais. > Só clicar na CREATE PULL REQUEST para as BRANCH serem mescladas no MASTER.

Depois de fazer o GIT PUSH da BRANCH temporário, vá no GitHub e clique no botão “COMPARE & PULL REQUEST” na etapa escrita “NomeBranchTemporario” HAD RECENT PUSHES LESS THAN A MINUTE AGO > Defina quais as BRANCHS serão unidas (Base: Principal / Compare: Temporária) > Escreva a descrição do que será alterado no código para ser aprovado > Clique CREATE PULL REQUEST > O usuário com autorização apertará em “MERGE PULL REQUEST” e depois “CONFIRM MERGE”.

1. CREATE A MERGE COMMIT (Todos os COMMITS serão ADD, através do MERGE no MASTER)
2. SHASH AND MERGE (Junta todos os COMMITS em um só e faz o MERGE para o MASTER)
3. REBASE AND MERGE (Todos os COMMITS serão ADD no topo do MASTER)

**EXECUTAR MERGE:** No GitHub Desktop, selecione a BRANCH MASTER e clique em BRANCH > MERGE INTO CURRENT BRANCH > Selecione a outra BRANCH que quer unir ao MASTER > MERGE “Nome da BRANCH” INTO MASTER > PUSH ORIGIN (O MASTER foi incrementado com as alterações da BRANCH) > Faça a junção com os outros BRANCH e realize o PUSH ORIGIN.

**COMO UNIR OS CÓDIGOS:**

Antes de fazer a união entre os códigos, “atualize” a BRANCH Principal com as versões do servidor. **Sintaxe:** **GIT PULL**

Entre na BRANCH Principal e execute o código, com isso, é necessário fazer um COMMIT extra (MERGE) para juntas as BRANCHS.

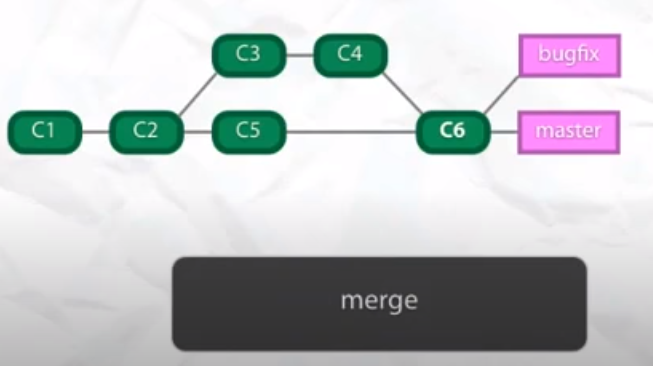
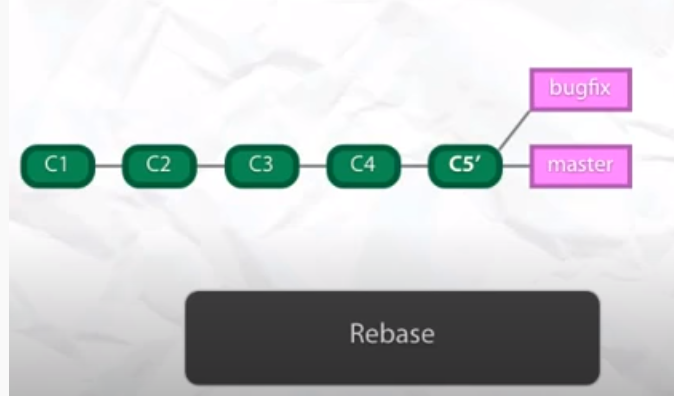
Normalmente utiliza MERGE para unir código quando estamos fazendo um FORK, pois é necessário visualizar a ordem cronológica e quando foi feita a colaboração (Modificação do código de outra pessoa)

**Sintaxe:**

**GIT MERGE** < “NomeDaBranchAdicionadanaBranchPrincipal” > **+ ENTER**

**GIT REBASE** < “NomeDaBranchAdicionadanaBranchPrincipal” > **+ ENTER**

**GIT REBASE -I HEAD** **~** < “N°daQtdeCommit” > **+ ENTER**



* **REBASE é unir duas BRANCH rebobinando (reaplicando) a ordem dos COMMITS do MASTER para incluir os COMMITS do BUGFIX e não faz um novo COMMIT (C6). Mantêm o histórico linear, porém modifica o último COMMIT (C5) para incluir as informações únicas do MASTER.**
* **MERGE é unir duas BRANCH (MASTER E BUGFIX) mesclando as duas versões num novo COMMIT (C6). Mantem a ordem cronologia visual.**

Escolha os comandos para os COMMITS:

* P, PINK = USE COMMIT
* R, REWORD = USE COMMIT, BUT EDIT THE COMMIT MESSAGE (mas edite a mensagem do COMMIT)
* E, EDIT = USE COMMIT, BUT STOP FOR AMENDING (mas parar para alterar)
* S, SQUASH = USE COMMIT, BUT MELD INTO PREVIOUS COMMIT (mas junte com COMMIT anterior)
* F, FIXUP = LIKE “SQUASH”, BUT DISCARD THIS COMMIT’S LOG MESSAGE (mas descarte a mensagem de LOG do COMMIT)
* X, EXE C = RUN COMMAND (THE REST OF THE LINE) USING SHELL (execute o COMMIT – o resto da linha - usando o SHELL)
* D, DROP = REMOVE COMMIT

Exemplo:

GIT REBASE -I HEAD ~2 (Unir 2 COMMITS)

|  |
| --- |
| COMMIT N → HEAD (ATUAL) |
| COMMIT 3 |
| COMMIT 2 |
| COMMIT 1 |

* PINK COMMIT N → HEAD (Usando o COMMIT ATUAL)
* SQUASH COMMIT 3 (Usando o COMMIT 3 e juntando com o COMMIT ANTERIOR, o COMMIT ATUAL)

**Conflito entre versões paralelas:** Quando alguma linha da estrutura possui informações diferentes em cada BRANCH.

No GitHub Desktop, sinalizará o conflito antes de realizar o MERGE da BRANCH > Será necessário escolher “Aceitar a mudança atual”, “Aceitar a mudança de entrada”, “Aceitar as duas alterações” e “Comparar as mudanças” > COMMIT MERGE > PUSH ORIGIN

**Sintaxe:**

**GIT PULL** < “NomeReporitórioRemoto” > <BRANCH> **- - REBASE/MERGE + ENTER** →

**GIT PUSH** < “NomeReporitórioRemoto” > <BRANCH> **+ ENTER** →

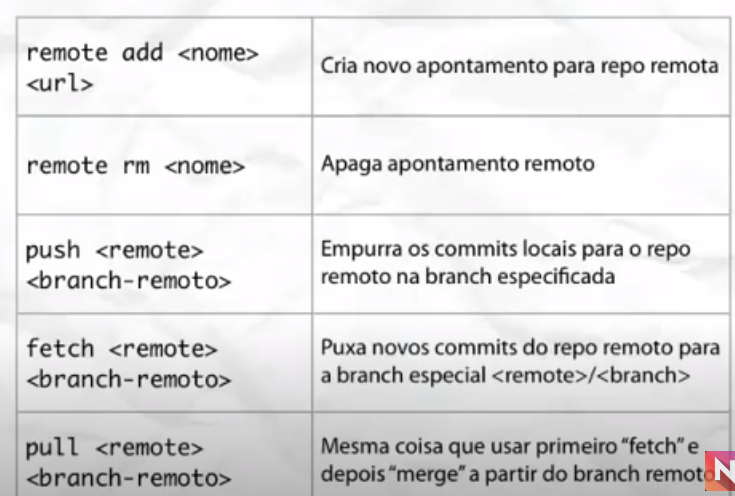
**GIT REBASE/MERGE - - CONTINUE**

Exemplo:

git pull origin master - - rebase/merge →

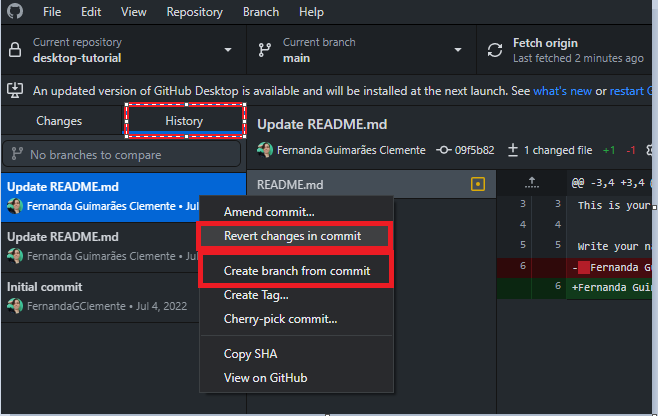
git push origin master

git rebase/merge - - continue



**HISTORY:** Recuperar uma versão antiga e atualizações forçadas.

Na aba HISTORY, identifique qual versão que quer voltar, clique com botão direito do mouse e clique em “REVERT CHANGES IN COMMIT” Reverta da versão atual até a versão que queira voltar. Realizando os PUSH ORIGIN em cada reversão. OU Faça uma BRHANCH da versão que deseja recuperar (“Voltar”), clicando com o botão direito do mouse e clique em “CREATE BRANCH FROM COMMIT” > Informe um nome para versão > Clique em CREATE BRANCH. Atualize os códigos da nova BRANCH e depois realize PUSH ORIGIN (Depois faça o mesmo processo para mesclar com o Master)



**COMO VERIFICAR HISTÓRICO DE ATUALIZAÇÕES (MOSTRA INFORMAÇÔES QUE O LOG NÃO MOSTRA):**

**Sintaxe:** **GIT REFLOG + ENTER** (A versão atual ficará no topo)

**OBS:** Mostra também os COMMITS que foram “excluídos”. Para restaura-los, escreva **GIT MERGE** <“N°doHASH”>

**GIT HELP (AJUDA):** No prompt, escreva o código abaixo para saber do que se trata os diversos comandos do GIT.

**Sintaxe:**

**GIT HELP** – Instruções mais gerais de cada comando

**GIT HELP -A** – Instruções mais detalhadas de cada comando

**GIT HELP** < “Comando” > - Instruções especificas do comando (Será aberto as informações no navegador)

Este arquivo em HTML está na pasta > **C:\Program Files\Git\mingw64\share\doc\git-doc**

**HOSPEDAGEM DE SITE:**

No GitHub remoto, entre no repositório em que os arquivos dos sites estão, clique em SETTINGS (Não é o SETTINGS do perfil). No meio da página, localize o campo GitHub PAGES > No campo SOURCE, selecione MASTER BRANCH, caso o site seja em HTML 5, CSS 3 e Java Script ou selecione MASTER BRANCH / DOCS FOLDER, caso o site não seja em HTML 5, CSS 3 e Java Script, pois hospedará o DOCS FOLDER.

**ISSUES (PROBLEMAS):**

* No GITHUB, dentro do repositório, clique em ISSUES e depois em NEW ISSUES.
* Escreva um título e a descrição do problema > Clique em SUBMIT NEW ISSUES.

**OBS:** Quem criou o repositório pode: LOCK CONVERSATION (Bloquear o “CHAT” referente ao ISSUES), INPIN ou PINNED ISSUES (Fixa ou desafixa uma ISSUE), TRANSFER ISSUE (Transferir uma ISSUE para ISSUE correta) e DELETE ISSUES (Excluir uma ISSUE)

**OBS:** No perfil do GITHUB, clique em SETTINGS > SAVED REPLIES > Informe um título e um texto de resposta. Inclua fotos, links e formatações > ADD SAVED REPLY.

**TEMPLATES de ISSUES e PULL REQUEST:** ISSUES.md e ISSUE.md

Repositório de modelo de TEMPLATES: <https://github.com/devspace/awesome-github-templates>

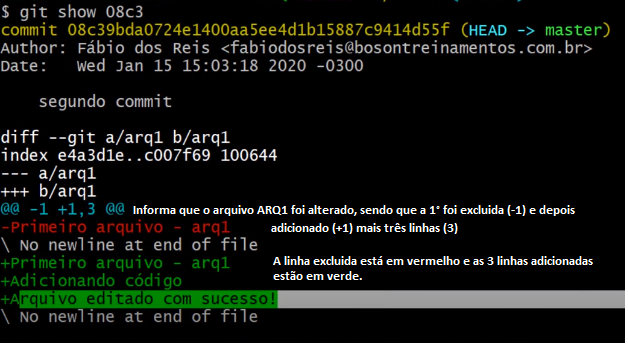
**Processo de revisão de código (code review):**

Ao receber um REVIEW faça as verificações e comentários e clique em FINISH YOUR REVIEW (Termine sua revisão). Aparecerá uma janela com as opções: “Fazer comentários” (COMMENT), “Aprovar” (APROVE) ou “Pedir mudanças” (REQUEST CHANGES) > SUBMIT REVIEW (Enviar revisão)

**OUTROS COMANDOS:**

**Sintaxe:**

* **GIT LOG** > Mostra a HASH do COMMIT, o autor, a data e o que foi feito.
* **GIT LOG... - - DECORATE** > Mostra mais informações do que “GIT LOG”.
* **GIT LOG... - - AUTHOR =** “Nome” > Mostra tudo o que foi feito pela aquela pessoa no repositório.
* **GIT LOG... - - GRAPH** > Mostra graficamente tudo o que foi no repositório.
* **GIT SHOW** > Mostra todas as informações relacionadas a HASH do último COMMIT.
* **GIT SHOW N°daHASH** > Mostra todas as informações relacionadas a HASH do COMMIT.



* **GIT DIFF** > Informa qual a diferença do arquivo antes ou depois da modificação
* **GIT DIFF...- - NAME-ONLY** > Informa qual o arquivo foi modificado.
* **GIT COMMIT...-AM...** “TítuloDaVersão” > Editar uma versão já existente.
* **GIT BLAME** > Informa qual foi o autor e o COMMIT da última alteração daquela linha do código
* **GIT LOG - - PRETTY = ONELINE** > Mostra todas as informações em linhas.
* **GIT LOG - - PRETTY = ONELINE - - GRAPH** > Mostra todas as informações em linhas e inclui gráficos.
* **GIT LOG - - PRETTY = ONELINE - - GRAPH - - ALL** > Mostra todas as informações em linhas e inclui gráficos da BRANCH e do MASTER.
* **GIT LOG - - SINCE =** ‘MM D AAAA’ > Mostra todas as informações desde a data MM D AAAA.
* **GIT LOG - - UNTIL =** ‘MM D AAAA’ > Mostra todas as informações até a data MM D AAAA.
* **GIT LOG - - SINCE =** ‘MM D AAAA’ ou N°.hours **- - UNTIL =** ‘MM D AAAA’ ou N°.hours > Mostra todas as informações entre as duas datas ou duas horas.
* **GIT LOG - - SHORTLOG** > Mostra as informações de forma simplificada (Quantidade de COMMIT, os nomes e o tipo por autor).
* **GIT LOG - - SHORTLOG -SN** > Mostra as informações de forma simplificada (Quantidade de COMMIT por autor).
* **GIT LOG - - N°** > Mostra as informações dos N° últimos COMMITS feitos.
* **GIT CLEAN -F** > Apaga arquivos do diretório de trabalho que não fazem parte do repositório online (Não foi aplicado o PUSH)
* **GITK - - ALL &** > Abri um programa gráfico para visualizar as versões existentes

**CRIANDO CHAVE SSH:**

1. Abra o GIT BASH > Escreva:

(ou) SSH-KEYGEN -T ED25519 -C < “E-mail” > + ENTER

(ou) SSH-KEYGEN.EXE -T RSA + ENTER

(ou) SSH-KEYGEN -T RSA -B 4096 -C < “E-mail” > + ENTER

1. Mostrará a mensagem “GENERATING PUBLIC/PRIVATE ALGORITHM KEY PAIR”;
2. Quando for solicitado a “ENTER FILE IN WHICH TO SAVE THE KEY” (Inserir um arquivo no qual deseja salvar a chave). Pressione ENTER (Deixe por padrão ID\_RSA)
3. Digite uma senha qualquer 2 vezes e depois pressione ENTER;
4. No prompt, escreva CD ~/...\*SSH/ > Escreva DIR > Mostrará ID\_RSA e ID\_RSA.PUB (NomeChave) > Escreva:

(ou) CAT ID\_RSA.PUB (NomeChave) + ENTER > Copie o código da chave

(ou) MORE ID\_RSA.PUB (NomeChave) + ENTER > Copie o código da chave

(ou) VI ID\_RSA.PUB (NomeChave) + ENTER > Copie o código da chave

1. No GitHub, entre nas configurações (SETTINGS) > Clique em SSH AND GPG KEYS > Clique em NEW SSH KEY > Escreva um nome para o título e cole o código da chave no campo KEY > ADD SSH KEY

**CRIANDO TAGS:** Para fazer uma versão nova de bibliotecas, de projetos, etc.

**Sintaxe:** **GIT TAG -A** < “N°daVersão” > **-M** “MensagemDaNovaVersão” **+ ENTER** →

(ou) **GIT PUSH** < “NomeReporitórioRemoto” > <BRANCH> **- - TAGS + ENTER** (Envia todas as TAGS)

(ou) **GIT PUSH** < “NomeReporitórioRemoto” > <BRANCH> **TAGS** < “N°daVersão” > **+ ENTER** (Envia a TAG específica)

Exemplo: git tag -a 1.0.0 -m “Mensagem” → git push origin master - - tags ou git push origin master tag 1.0.0

**MOSTRA TODAS AS TAGS FEITAS: Sintaxe:** **GIT TAG + ENTER**

**EXCLUIR UMA TAG NO GITHUB: Sintaxe:** **GIT PUSH** < “NomeReporitórioRemoto” > **:** < “N°daVersão” > **+ ENTER**

**EDITANDO COMMITS COM AMEND:** Usado quando o COMMIT ainda não foi enviado para o repositório online.

Foi feito um COMMIT, porém quer incluir nesse mesmo COMMIT algum arquivo, com isso, faça a adição desse arquivo com o GIT ADD < “NomeArquivo.extensão” > e depois escreva: **Sintaxe:** **GIT COMMIT -M “NomeDoCommit” - - AMEND + ENTER**

A única coisa que muda do COMMIT é a numeração da HASH, pois foi incluído um novo arquivo.

**PEGANDO COMMITS ESPECÍFICOS: Sintaxe: GIT CHERRY -PICK** < “N°daHash” > **+ ENTER**

**ADICIONANDO APENAS TRECHOS DE UM CÓDIGO: Sintaxe:** **GIT ADD -P**

Escolha um comando para ser executado no trecho do código (STAGE THIS HUNK?):

* Sim = Y
* Não = N
* Passe e vá para anterior = G
* Editar manualmente o que quer fazer = E
* Não e saia imediatamente do menu = Q
* Adiciona esse e todos os outros = A
* Não adiciona esse e nem os outros = D
* Passe e vá para o próximo = J
* Divide em partes menores o trecho = S

**CRIAÇÃO DE ZIP: Sintaxe: GIT ARCHIVE** <BRANCH> **- - FORMAT = ZIP - - OUTPUT =** “NomeDoZip.zip”

**WORKFLOWS DO GIT (FLUXO DE TRABALHO):**

É uma sequência de passos necessários para automatizar processos de acordo com um conjunto de regras definidas, permitindo que estes possam ser transmitidos de uma pessoa para outra.

**CENTRALIZADO WORKFLOWS:** BRANCH únicos (Master) e funciona melhor com equipes bem pequenas (4 pessoas).

1. **GIT ADD \***
2. **GIT COMMIT -M** “NomeDaVersão”
3. **GIT PULL ORIGIN MASTER - - REBASE**
4. **GIT PUSH ORIGIN MASTER**

**FEATURE BRANCH WORKFLOW:** Um BRANCH para cada FEATURE (com base no MASTER) e permite trabalhos paralelos com equipes maiores.

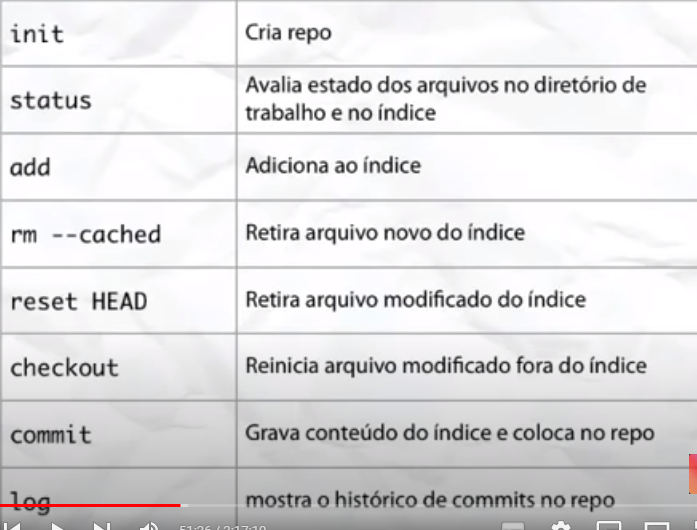
**GITFLOW WORKFLOW:**

1. Instalar a extensão GIT FLOW
2. **GIT FLOW INIT > MASTER > DEVELOP > FEATURE > RELEASE > HOTFIX > SUPPORT > V**
3. **GIT FLOW FEATURE START** “NomeDaFeature”
4. Faça as alterações no bloco de código
5. **GIT ADD\* ou GIT ADD** “NomeArquivoAletrado.extensão”
6. **GIT COMMIT -M** “NomeDaVersão”
7. **GIT FLOW FEATURE PUBLISH** “NomeDaFeature” **> PULL REQUEST > BASE: DEVELOP - COMPARE: FEATURE/** “NomeDaFeature”
8. **GIT CHECKOUT DEVELOP**
9. **GIT PUSH ORIGIN DEVELOP**
10. **CREATE PULL REQUEST** no GITHUB
11. **GIT CHECHOUT –**
12. **GIT FLOW FEATURE FINISH** “NomeDaFeature”
13. **GIT PUSH ORIGIN DEVELOP**
14. No GITHUB, clique em **DELETE BRANCH >** Terá apenas o MASTER e o DEVELOP
15. **GIT FLOW RELEASE START** “N°daVersãoDoProjeto”
16. **GIT FLOW RELEASE FINISH** “N°daVersãoDoProjeto”
17. **: W Q > I >** Exclui # para fazer a TAG com o “N°daVersãoDoProjeto” > **: W Q**
18. **GIT PUSH ORIGIN MASTER - - TAG**
19. **GIT CHECKOUT DEVELOP**
20. **GIT PUSH ORIGIN DEVELOP**
21. **GIT FLOW HOTFIX START** “NomeDaHotfix”
22. Faça as alterações no bloco de códigos
23. **GIT ADD \***
24. **GIT COMMIT -M** “NomeDaHotfix”
25. **GIT FLOW HOTFIX FINISH** “NomeDaHotfix”
26. **: W Q > I** > Informe uma mensagem da TAGS > **: W Q**
27. **GIT CHECKOUT MASTER**
28. **GIT PUSH ORIGIN MASTER - - TAGS**

**GIT CHECKOUT -B GH-PAGES**

**GIT PUSH ORIGIN GH-PAGES**

**Foi criado uma hospedagem do projeto > usuario.github.io/nome\_repositorio**



**ELEMENTOS:**

LS = Mostra os arquivos

-A = Mostra os arquivos, incluindo os ocultos

-LA = Mostra uma lista dos arquivos, incluindo os ocultos

-M = Permiti criar uma mensagem

**SEGURANÇA:** No GITHUB, no PERFIL, clique em SETTINGS > PASSWORD AND AUTHENTICATION > No campo TWO – FACTOR AUTHENTICATION, clique em ENABLE TWO-FACTOR AUTHENTICATION > No PASSWORD: Informe a senha do GitHub > Confirme a senha > Escolha SET UP USING AN APP > Continue > Use o aplicativo GOOGLE AUTHENTICATOR para escanear o QR CODE > Informe o número que aparecerá no APP e coloque no GitHub > ENABLE > Faça o Download dos códigos de recuperação > Salvei meus códigos de recuperação > DONE

32977-94baa

ad7c1-81b1d

93dc8-f4944

8ce6f-3182c

3a65b-96e8b

19e13-a7a6a

295d0-b58f9

75c3b-6dbd9

e390f-3699c

7ca09-9cbde

514e8-f2c41

5575f-99eac

b1654-2629c

4aa5f-5900a

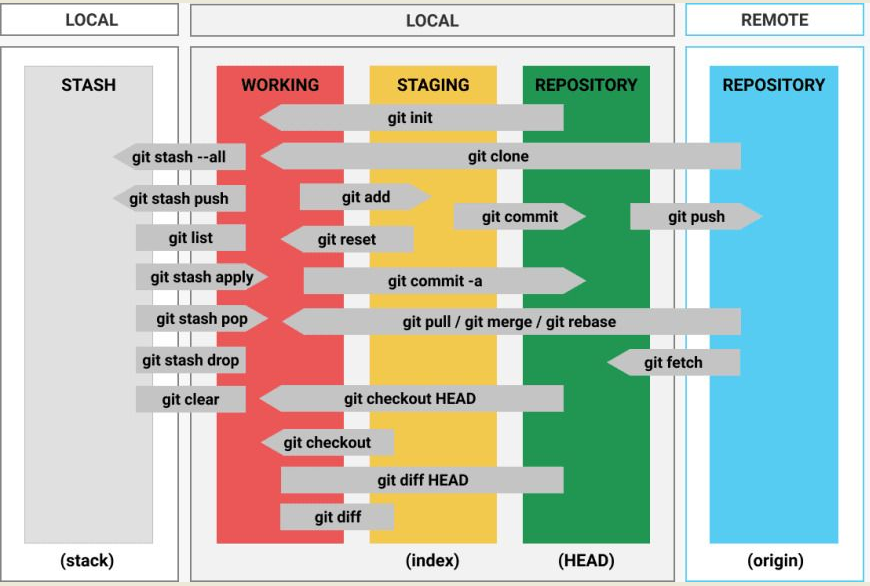
9fe50-ef178

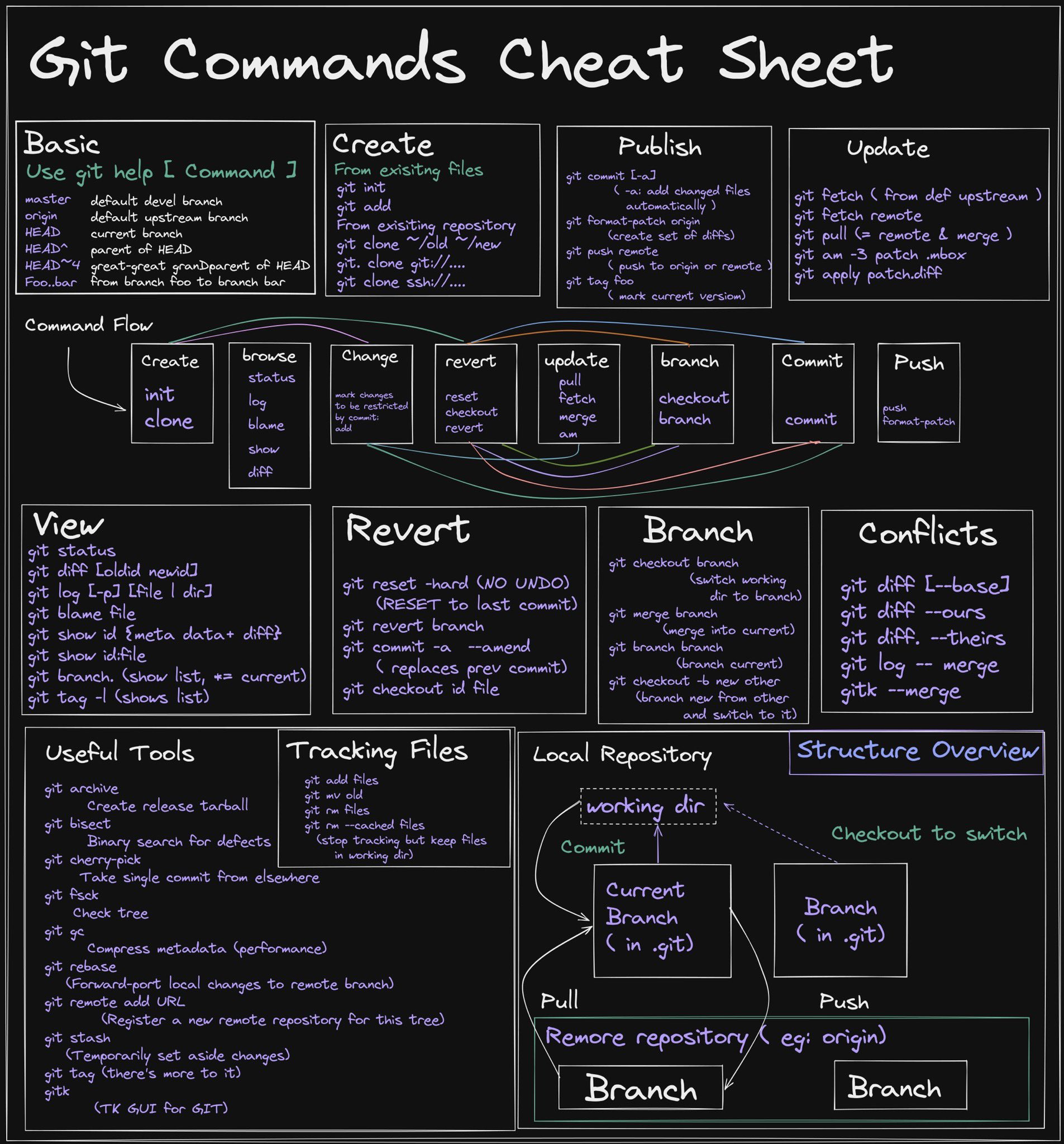
5edcc-0956c

**Descobri qual é a versão do GIT instalado: GIT -- VERSION**

# **Passo a passo**

1. Instale o GIT
2. Abra o GIT BASH
3. Gere a chave SSH
4. Faça as configurações iniciais do GIT:
   1. **GIT CONFIG...- - GLOBAL USER.NAME** < “NomeCompleto” >
   2. **GIT CONFIG...- - GLOBAL USER.EMAIL** < “LoginDoGitHub” >
   3. **GIT CONFIG...- - GLOBAL CORE.EDITOR** < “VIM, S ou EMACS” > **+ ENTER** (Por padrão o editor é o VIM)
   4. Verifique se as informações foram corretamente, usando **GIT CONFIG...- - LIST**
5. Faça a criação do diretório para armazenar o projeto:
   1. Use o comando CD/CD \*\* para ir até um local para criar a pasta onde ficará os códigos;
   2. Atalho CTRL + L ou escrever CLEAR = Limpa a tela e PWD mostra em qual diretório está;
   3. Escreva MKDIR < “NomeDoRepositório” > para criar a pasta ficará os códigos;
   4. Use o comando CD para entrar no repositório criado;
6. Inicializar o repositório com o comando **GIT INIT** ou **GIT INIT \***
7. Inclua os arquivos no repositório criado.
8. Faça as adições dos arquivos usando o comando **GIT ADD** < “NomeDoArquivo.extensão” > ou **GIT ADD \***
9. Faça a gravação das adições dos arquivos usando o comando **GIT COMMIT -M** “TituloDaVersão”.
10. Sempre que modificar, excluir ou adicionar arquivos no repositório, precisa adicionar as alterações (GIT ADD \*), gravar as alterações (GIT COMMIT) e empurrar as alterações para o repositório online/remoto (GIT PUSH)
11. **GIT FETCH** < “NomeReporitórioRemoto” > < “BRANCH”>
12. GIT PULL da BRANCH Principal do repositório original/Remoto;
13. Gerar uma nova BRANCH a partir da BRANCH Principal;
14. Trabalhar e adicionar novas funcionalidades na nova BRANCH que criou;
15. Finalizar o trabalho na BRANCH Temporária;
16. GIT CHECKOUT na BRANCH Principal;
17. GIT PULL da BRANCH Principal;
18. Mergiar (Unir) o código da BRANCH Temporário com a BRANCH Principal (Depois de testar);
19. GIT PUSH da BRANCH Principal;





# Hospedagem no GitHub de páginas em HTML, CSS e JavaScript

Hospedando site gratuitamente com GitHub Pages - @Curso em Vídeo HTML5 e CSS3

<https://www.youtube.com/watch?v=_jI3782DGDc&ab_channel=CursoemV%C3%ADdeo>