MANUAL DO INICIANTE

**BÁSICOS GIT E GIT FLOW  
BY JAILTON DPAULA**

Início Março de 2022

Sumário

[*PRIMEIROS PASSOS GIT* 3](#_Toc100425893)

[GIT CONFIG 3](#_Toc100425894)

[*NAVEGANDO COM GIT* 3](#_Toc100425895)

[*GERENCIANDO ARQUIVOS* 3](#_Toc100425896)

[*VIM* 4](#_Toc100425897)

[*REPOSITÓRIO GIT* 4](#_Toc100425898)

[Iniciando repositório git 4](#_Toc100425899)

[Áreas do GIT 4](#_Toc100425900)

[UNTRACKED (Não Rastreado) 4](#_Toc100425901)

[UNMODIFIED (Não Modificado) 4](#_Toc100425902)

[MODIFIED (Modificado) 5](#_Toc100425903)

[STAGED 5](#_Toc100425904)

[GIT STATUS 5](#_Toc100425905)

[*UNTRACKED E MODIFY* 5](#_Toc100425906)

[GIT ADD 5](#_Toc100425907)

[GIT RESTORE (Voltar arquivo) 6](#_Toc100425908)

[GIT DIFF 7](#_Toc100425909)

[GIT DIFF --NAME-LOG 7](#_Toc100425910)

[*STAGED* 8](#_Toc100425911)

[GIT DIFF --STAGED 8](#_Toc100425912)

[GIT RESET HEAD 8](#_Toc100425913)

[*COMMIT* 9](#_Toc100425914)

[GIT RESET 9](#_Toc100425915)

[GITK 9](#_Toc100425916)

[GIT COMMIT 9](#_Toc100425917)

[GIT LOG 11](#_Toc100425918)

[*GIT IGNORE* 12](#_Toc100425919)

[*BRANCH* 13](#_Toc100425920)

[BRANCH 13](#_Toc100425921)

[*GIT STASH* 15](#_Toc100425922)

# ***PRIMEIROS PASSOS GIT***

Para começa a utilizar o GIT é necessário fazer as configurações iniciais, as duas primeiras configurações obrigatórias é E-mail e Usuário.

### GIT CONFIG

**🖮 git config --global user.name “JAILTON DPAULA”**

Definir o nome do usuário do git.

**🖮 git config --global user.name**

Consulta o usuário configurado para o git.

**🖮 git config --global user.name “jailton.viana2@hotmail.com”**

Definir o e-mail do utilizador do git.

**🖮 git config --global user.name**

Consulta o e-mail configurado do utilizador do git.

**🖮 git config --list**

Lista todas configurações.

# ***NAVEGANDO COM GIT***

**🖮 cd pasta** ou **cd pasta/pasta**

Acessar uma pasta via prompt.

**🖮 cd ..** ou **cd ../..**

Volta a pasta.

**🖮 dir**

Detalha o conteúdo dessa pasta (diretório e arquivos).

**🖮 mkdir “nome pasta”**

Cria uma pasta (diretório).

# ***GERENCIANDO ARQUIVOS***

O git por padrão utiliza o editor vim, porém pode ser alterado para que o utilize o editor de sua preferência, assim quando for necessário abri algo com editor padrão abri o da preferência.

**🖮 git config --global core.editor “code --wait”**

Definir o VS Code como editor padrão do git.

**🖮 code “nome arquivo.formato”**

Para abri ou criar um arquivo é necessário o comando do editor mais o nome do arquivo. O comando a cima abri com o VS Code.

**🖮 vim “nome arquivo.formato”** ou **vm “nome arquivo.formato”**

Abri com o editor vim.

# ***VIM***

**“ ~ ”:** Apenas marcador de linha.  
**“ I ”:** Iniciar o insert.  
**“ esc ”:** Sair do modo de insert.  
**“ ; ”:** Abri campo para editar comando.  
**“ w ”:** Salvar arquivo.  
**“ q ”:** Sair do vim.  
**“ wq ”:** Salvar e sair do vim.

# ***REPOSITÓRIO GIT***

### Iniciando repositório git

Um repositório tratasse de uma pasta onde todo conteúdo dela é monitorado pelo git, rastreado assim exclusões, adições e modificações.

**🖮 git init**

Inicia um repositório

Este comando é usando via prompt de comando dentro da pasta (diretório) no qual contém seu projeto, ele inicia um repositório git. Dentro da pasta é criada uma pasta invisível com nome. git dentro dessa pasta tem todo conteúdo do versionamento. Todos os itens existentes ou adicionado após o início do repositório na pasta serão enviados para área UNTRACKED, por se tratar de arquivos novos dentro do repositório.

O git realiza versão do projeto em commits com hash (códigos) únicos, para chegar até o commit a modificação tem que passar por algumas áreas.

### Áreas do GIT



### UNTRACKED (Não Rastreado)

Está área contém os arquivos que foram recém adicionados ou os arquivos que já existia no projeto e foram excluídos, ou seja, arquivos que estavam no commit anterior área UNMODIFIED.

### UNMODIFIED (Não Modificado)

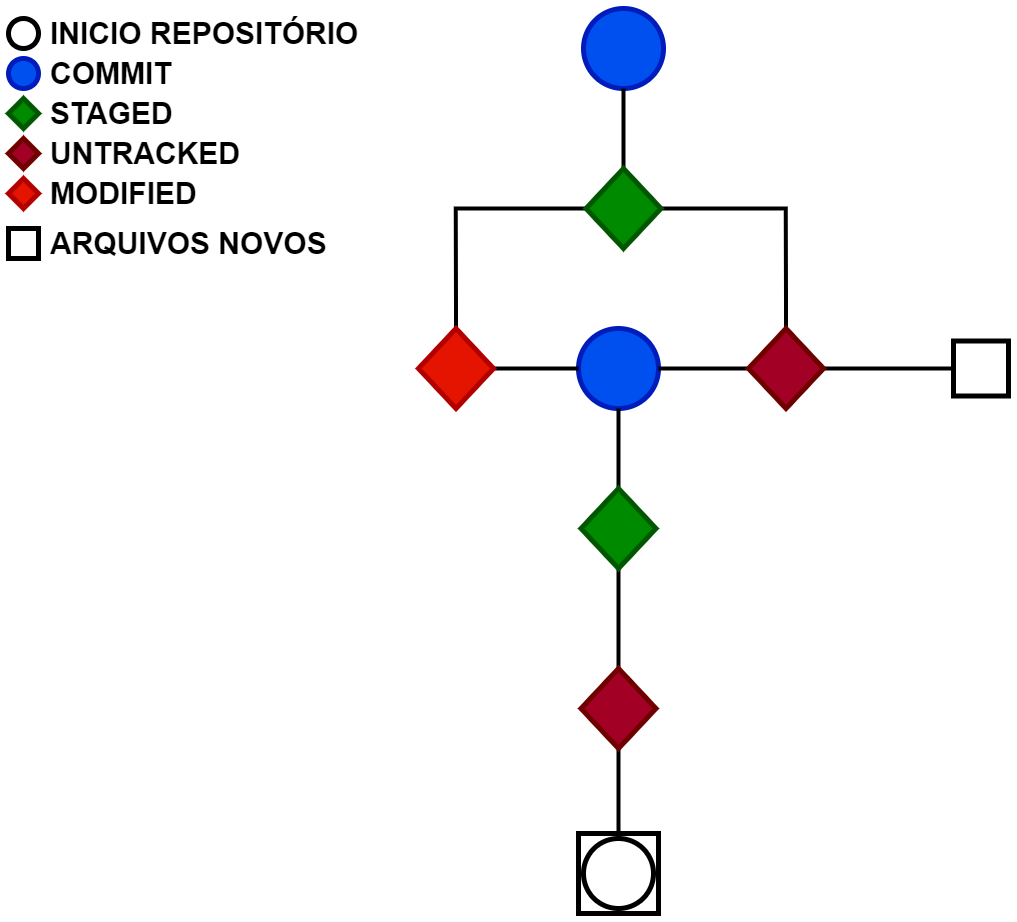
Está área é representada pelo commit, ela é um ponto salvo na linha do tempo do projeto e tudo que é modificado pós salvo na UNMODIFIED vai para área MODIFIED ou UNTRACKED do próximo UNMODIFIED.

### MODIFIED (Modificado)

Está área contém os arquivos que existi no commit anterior, porém foram modificados de alguma maneira.

### STAGED

Esta é a área onde concentra todas modificações que vão ser passada para o próximo commit, quando o registro de modificação sai das áreas UNTRAKED ou UNMODIFIED para STAGED essa modificação fica suspensa e todas alterações realizadas nos arquivos a parti desse momento serão apenas aplicadas nas áreas UNTRAKED ou na UNMODIFIED do commit em construção, as modificações são vão para área de STAGED se partirem das áreas UNTRAKED ou na UNMODIFIED.



### GIT STATUS

**🖮 git status**

O comando git status mostra os arquivos que contém nas áreas UNTRACKED, MODIFIED e STAGED.

# ***UNTRACKED E MODIFY***

### GIT ADD

**🖮 git add nome-arquivo**

O comando git add serve para enviar alterações da UNTRACKED e MODIFED para área STAGER.

**🖮 git add .**

🠶 Regra “**nome-arquivo**”: o nome do arquivo pode ser substituído por o caractere coringa ponto “.”, dessa forma enviando todos os arquivos contidos na MODIFIED e UNTRACKED para área STAGED.

**🖮 git add \*.png**

🠶 Regra “**nome-arquivo**”: Podemos utilizar também o caractere especial coringa asterisco “**\***”, onde quando substituído pelo nome, parte do nome ou formato significa que aceita qualquer sequência de caractere.

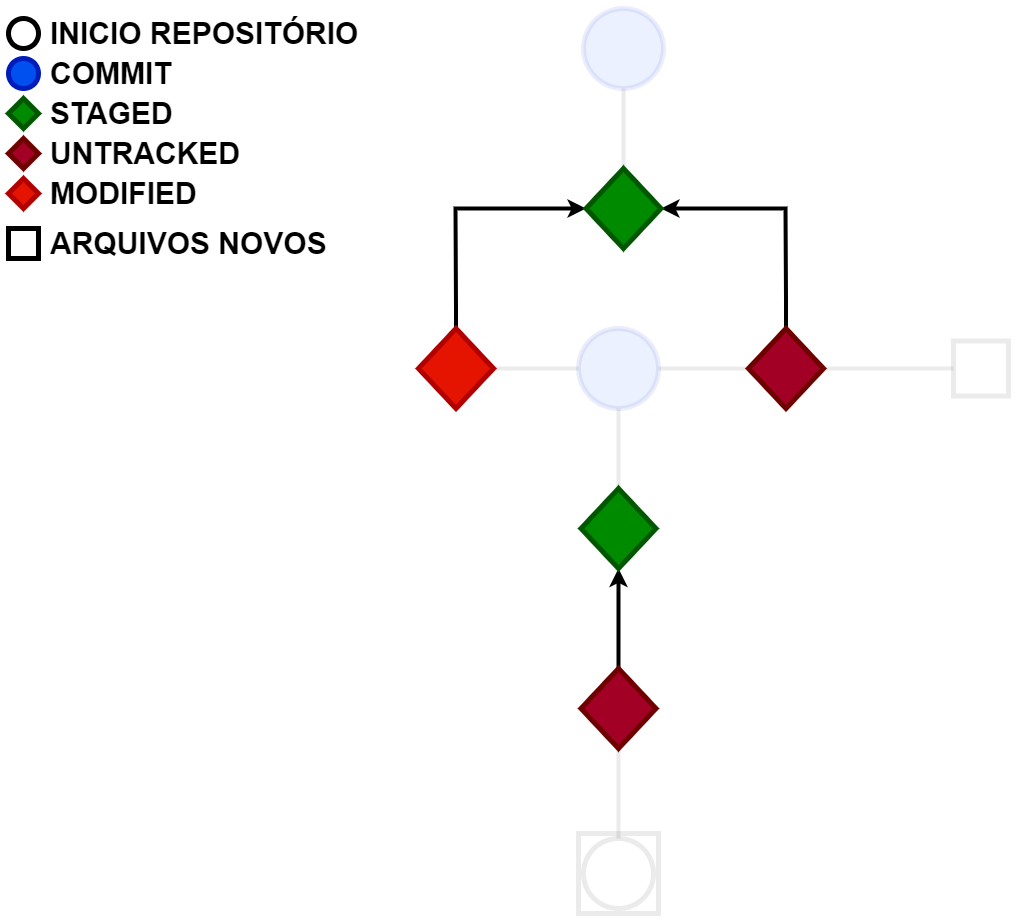
Exemplos de Arquivo Selecionado:   
*documento.png  
arquivo.png  
pasta/arquivo2.png*

O uso com arquivo git add seleciona qualquer arquivo que o formato seja PNG e esteja na área MODIFIED e UNTRACKED e envia para área STAGED.

**🖮 git add documentos-\***

🠶 Regra “**nome-arquivo**”: Seleciona qualquer arquivo que tenha no começo do nome a sequência de string “**documentos-**” na área MODIFIED e UNTRACKED e envia para área STAGED.

*documento-janiero.pdf  
documento-importante.jpeg*



### GIT RESTORE (Voltar arquivo)

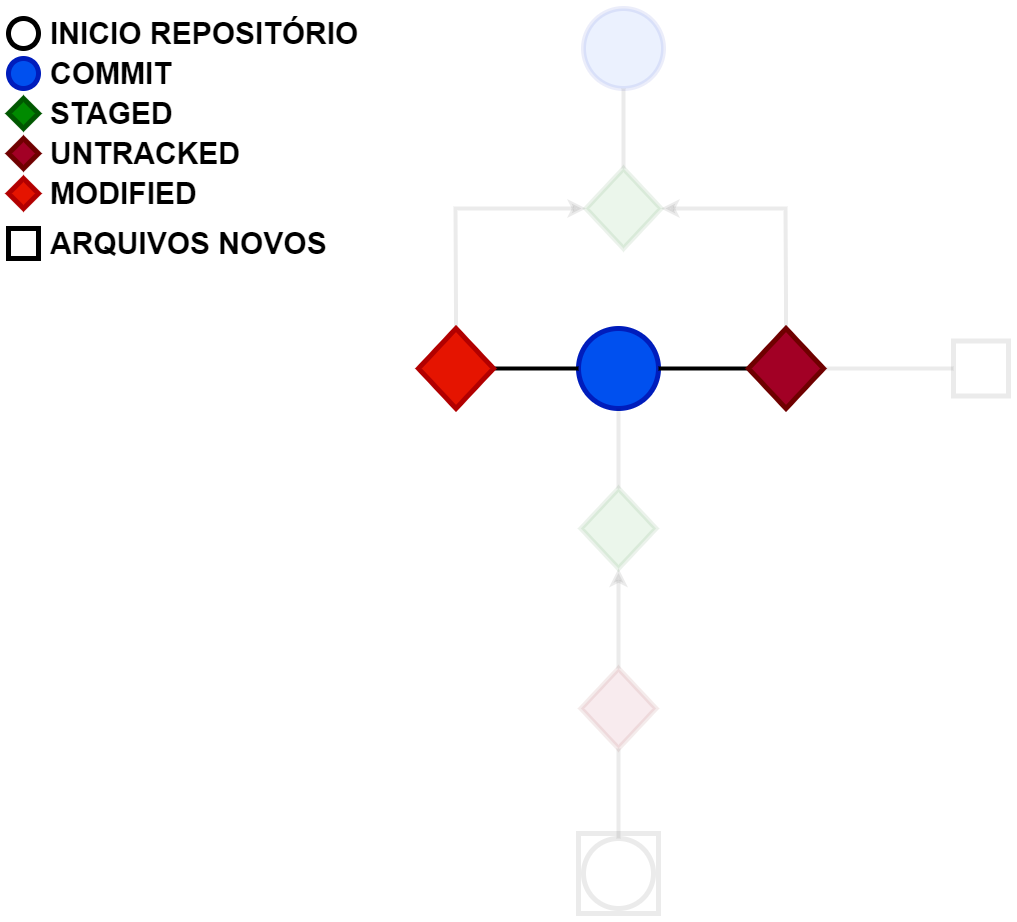
**🖮 git restore**

Este comando serve para reverter alterações feitas no arquivo.  
🠶 *Regra nome do arquivo se aplica.*

### GIT DIFF

**🖮 git diff**

Este comando mostra a diferença do último commite para a alterações contidas nas áreas *UNTRACKED e MODIFIED.   
regra nome do arquivo se aplica.*



### GIT DIFF --NAME-LOG

**🖮 git diff --name-only**

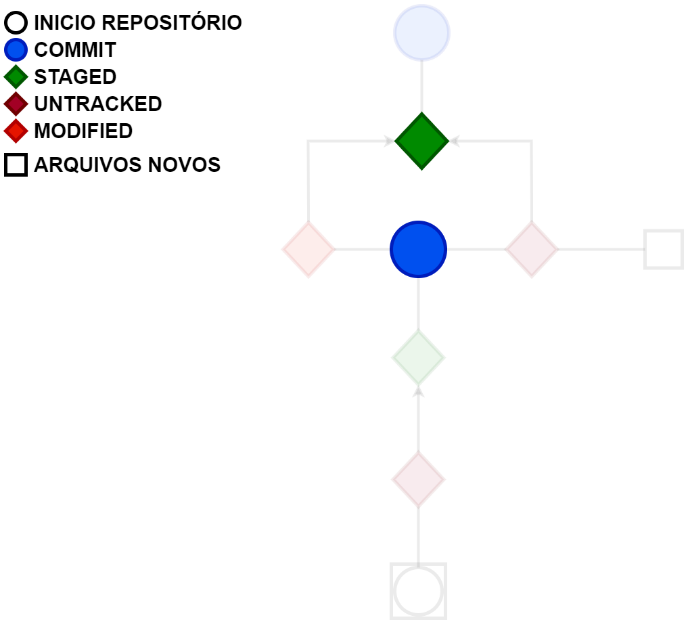
Comando similar ao comando *git dif,* porém não mostra a diferença do código apenas os arquivos alterados*.   
regra nome do arquivo se aplica.*

# ***STAGED***

### GIT DIFF --STAGED

**🖮 git diff --staged**

Este comando mostra a diferença do código contido na área STAGED para o ultimo commit*.   
regra nome do arquivo se aplica.*

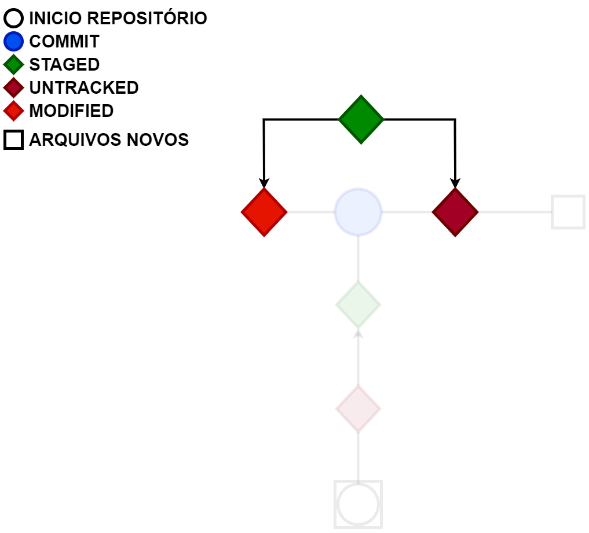


### GIT RESET HEAD

**🖮 git reset HEAD nome-arquivo**

*A regra nome do arquivo se aplica, porém, nome do arquivo é opcional, quando não informado pega todos os arquivos contidos na SATGED.*

Este comando serve para voltar as alterações da área *STAGED* para *MODIFIED.*

****

# ***COMMIT***

### GIT RESET

O comando *git reset* serve para reverter um commit (*UNMODIFIED), podendo voltar para STAGED, MODIFIED / UNTRACKED ou apagando o commit (UNMODIFIED).*

**🖮 git reset --soft**

Volta o commit para a área *STAGED*.

**🖮 git reset --mixed**

Volta o commit para área MODIFY ou UNTRACKED.

**🖮 git reset --hard**

Apaga o commit por completo.

### GITK

**🖮 gitk**

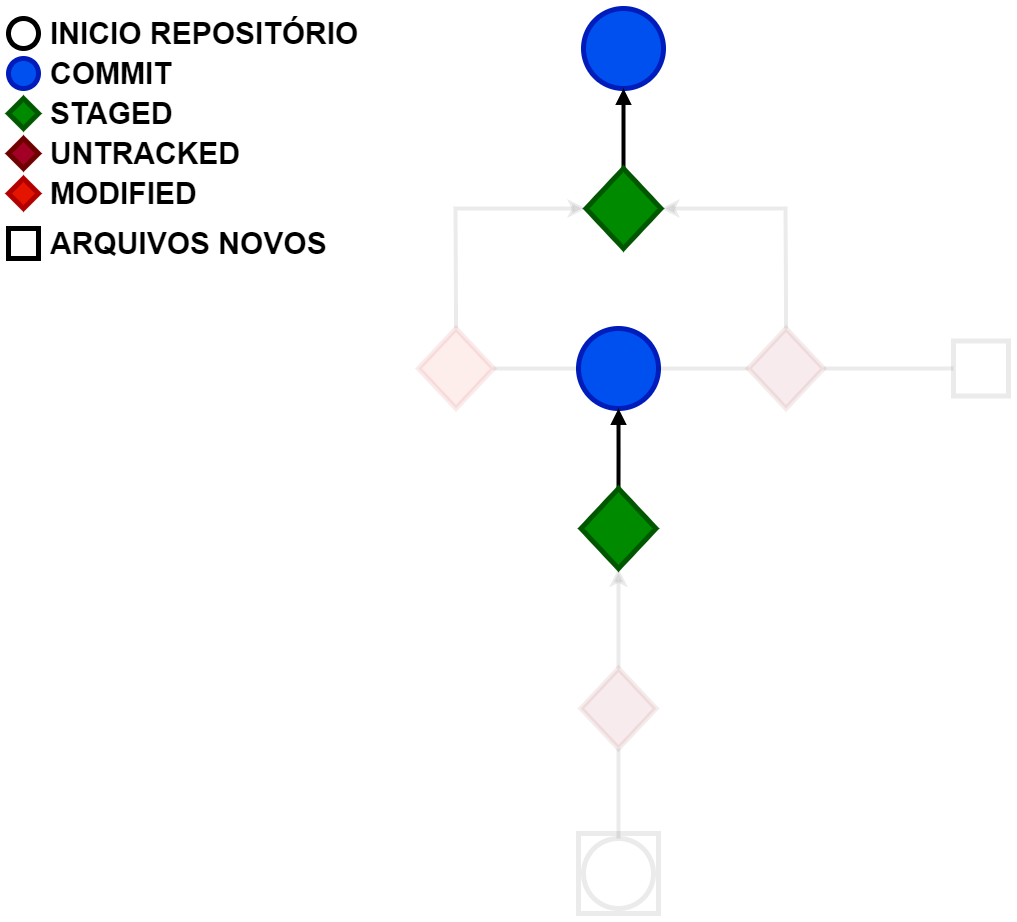
Este comando abri a interface gráfica do git*.*

### GIT COMMIT

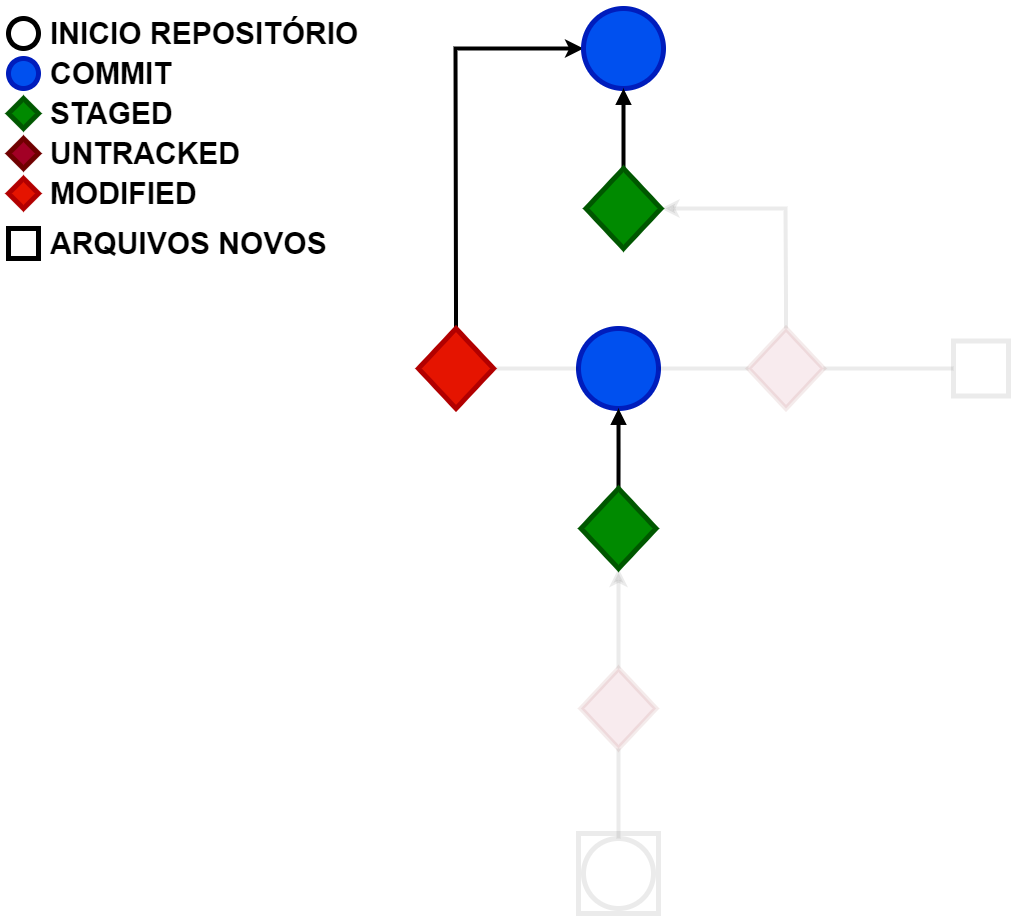
O comando *git commit* serve para enviar as alterações contidas na área *STAGED* para uma nova *UNMODIFIED (commit)*ou sobrescrever a *UNMODIFIED (commit)* anterior.

**🖮 git commit nome-arquivo –m“mensagem”**  *regra nome do arquivo se aplica, porém nome do arquivo é opcional, quando não informado pega todos contidos na SATGED.*

*O comando –m* serve para escrever uma mensagem no commit, caso não informado abri o editor padrão para escrever um mensagem.

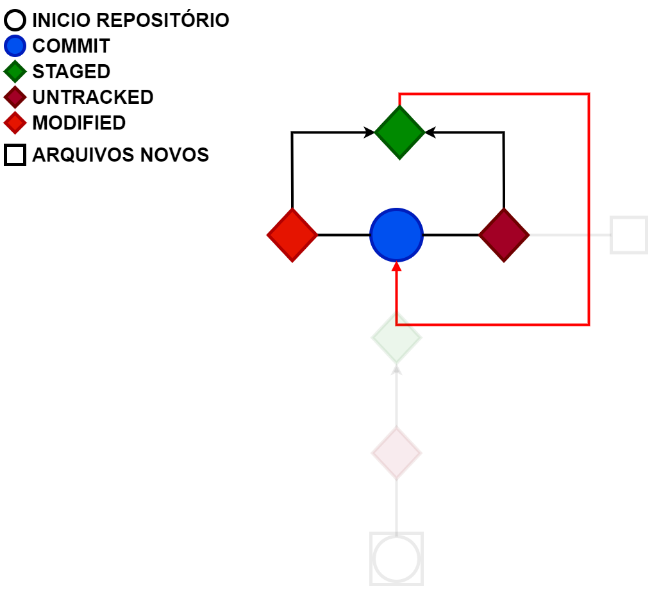


**🖮 git commit -a –m“mensagem”  
🖮 git commit -am“mensagem”**o comando *–a* ou *–am* serve para enviar as alterações contidas na *MODIFIED* e *STAGED* para o novo commit (*UNOMDIFIED).*



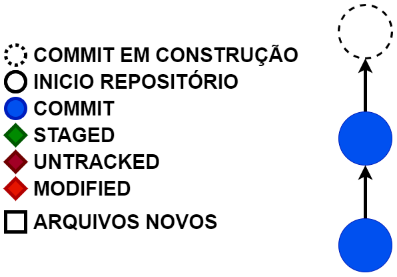
**🖮 git commit --amend -m“mensagem”**

Este comando atualiza o último commit (*UNMODIFIED)* realizado ao invés de criar um novo commit (*UNMODIFIED*)*.*

**

**Commit**

Um commit são pontos de restaurações criado dentro do projeto, este ponto pode ser consultado ou restaurado.

Nesse exemplo cada commit (*UNMODIFIED)* é um ponto azul o ponto branco tracejado é o novo commit que está sendo formado. As novas *MODIFIED*, *UNTRACKED e* STAGED.

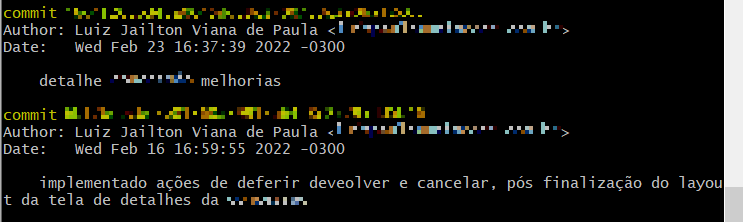
Cada commit gera um hash único.

Os commits podem ser consulta através da forma gráfica com o comando *gitk* ou outra interface gráfica, porém existi diversos comandos para consulta via prompt de comando.

### GIT LOG

**🖮 git log**

Este comando lista em ordem cronológica e decrescente os commit existente no projeto com suas informações como hash, autor, data, mensagem e etc.



**🖮 git log --decorate**

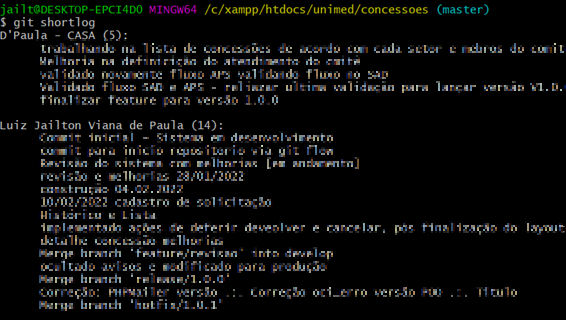
???

**🖮 git log --author=“autor”**

Filtra os commits pelo autor determinado no parametros.

**🖮 git shortlog**

Agrupa os commits por autor trazendo a quantidade de comitê do lado do nome e a baixo trazendo os commits realizados apenas com suas observações.



**🖮 git shortlog -sn**

Traz apenas o nome dos autores e a quantidade de commit realizado no projeto.

**🖮 git log --graph**

Mesmo resultado do *git log* porém com um gráfico ao lado esquerdo, não recomendado, melhor utilizar *gitk*.

**🖮 git show hash**

Mostra a diferença do commit informado para o commit anterior, podendo passar dois hash para comparação.

# ***GIT IGNORE***

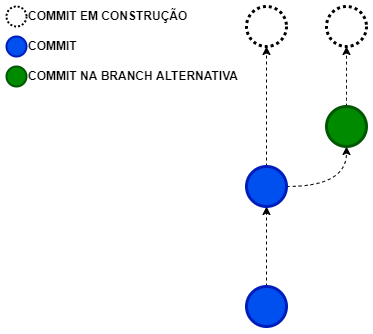
O arquivo *.gitignore* serve para adicionar formatos de arquivos, nomes ou partes de arquivos e pasta que o git não vai rastrear dentro do repositório, ou seja, não vai mapear mudanças, exclusão ou adição.

*regra nome do arquivo se aplica.*

Exemplo:  
\*.formato  
arquivo.formato  
\*/pasta/

# ***BRANCH***

Branchs são ramificações da linha do tempo principal. No padrão básico do git é utilizado sempre duas branch a principal de dados de produção e a de desenvolvimento onde é realizado todas alterações.



### BRANCH

**🖮 git branch**

Este comando lista todas as branchs ativas no projeto.

**🖮 git branch nomeBranch**

Cria uma branch com o nome informado.

**🖮 git checkout nomeBranch**

Mudar da branch atual para a branch mencionada.

**🖮 git checkout –b nomeBranch**

Junção dois dois últimos comandos, cria uma nova branch e já muda para a nova branch criada.

**🖮 git checkout –d nomeBranch**

Apaga a branch mencionada.

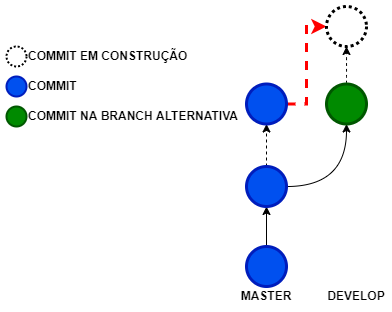
**🖮 git checkout –D nomeBranch**

Apaga a branch mencionada independente das restrições.

**🖮 git merge nomeBranch**

O comando *merge* trás as alterações contidas na branch mencionada para branch atual.

No exemplo estaríamos na branch *develop* e executaria o comando ***git merge máster*** trazendo assim as modificações contidas



# ***GIT STASH***

GIT STASH