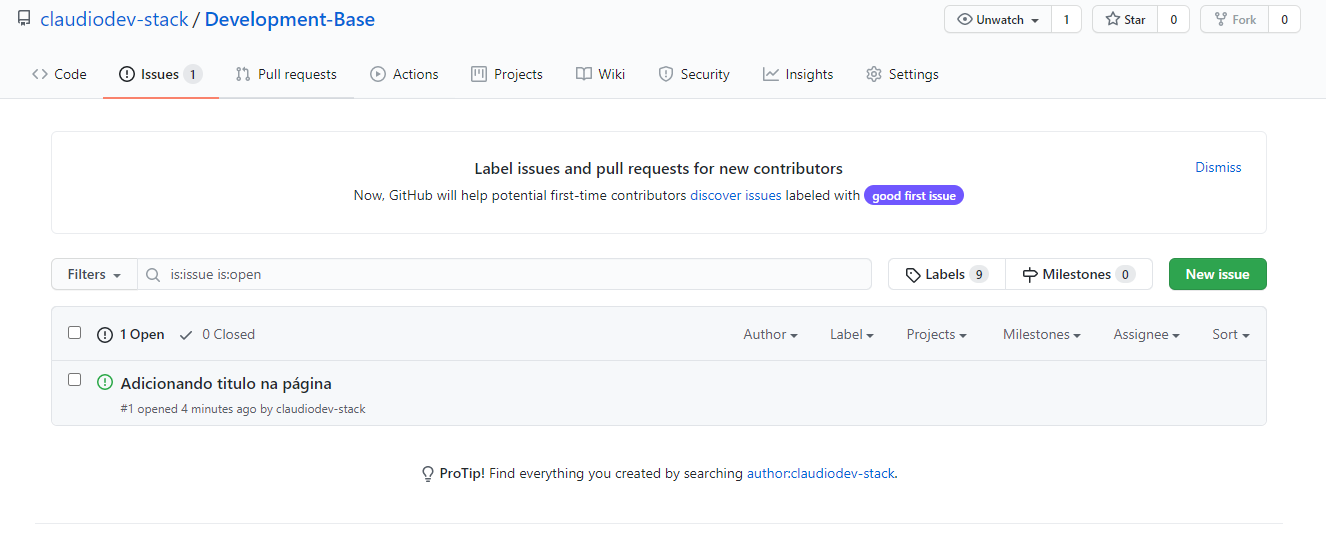
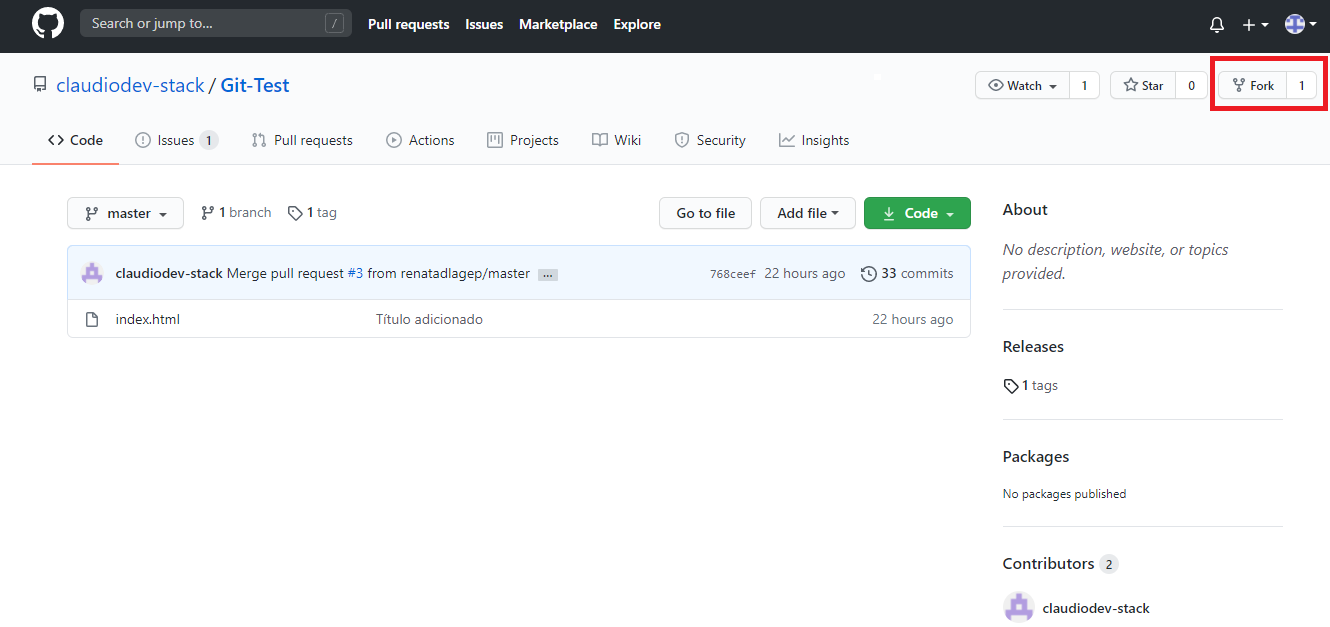
**CRIANDO ISSUES (PROBLEMAS) PARA DEBATES E SOLUÇÕES EM OPEN SOURCE**

Na comunidade de desenvolvedores o código open source pode ser visto, melhorado e compartilhado entre si, com isso o campo de issues de diversos githubs de usuários compõe issues para serem resolvidos, você também pode criar novas issues para debate e ajuste, melhoras de ser código e também ajudar a comunidade.



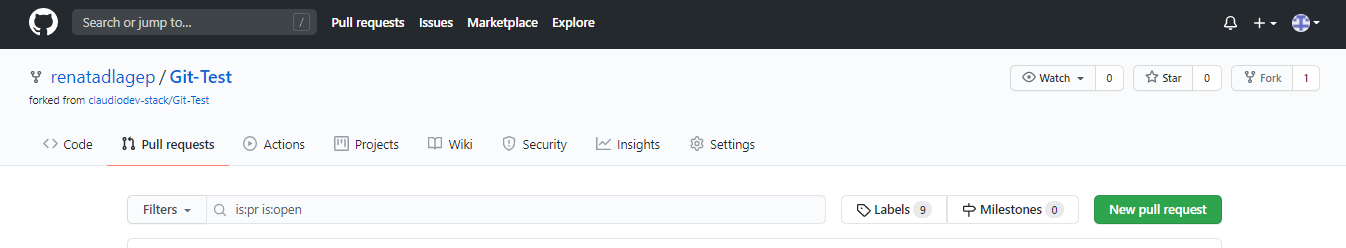
**FORK**

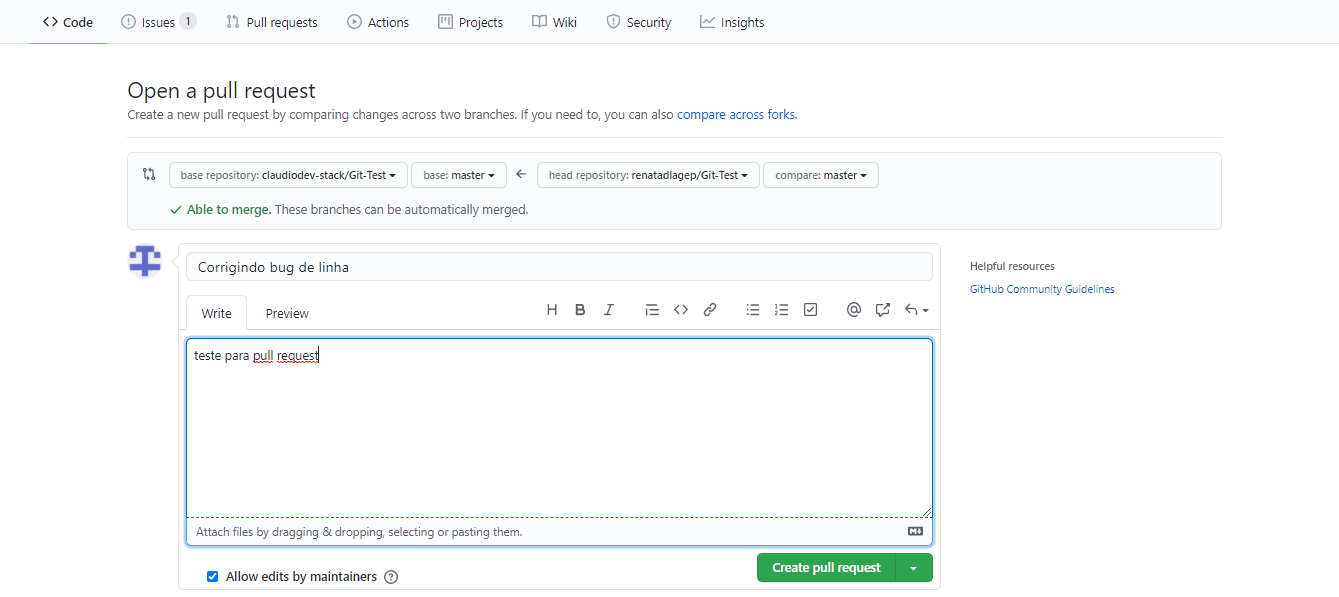
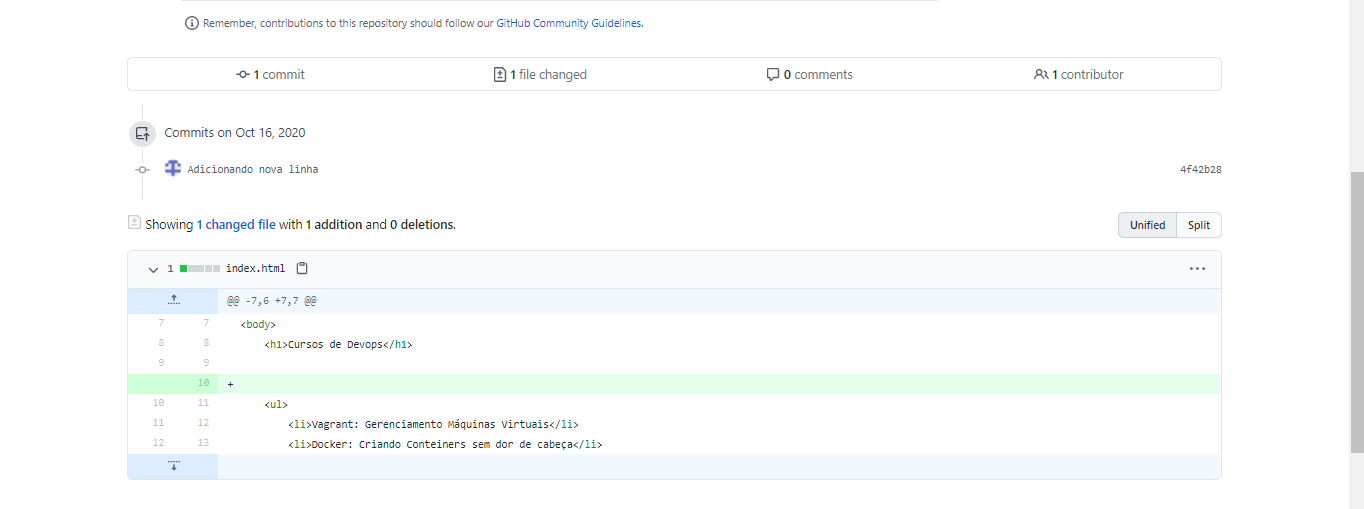
Uma fork é uma cópia de um repositório que deseja trabalhar e atualizar dentro da sua conta github. Basta clicar no canto direito do github em fork e vai construir a fork, com isso basta adicionar ao remoto, trabalhar dentro dela e depois dar um push para o remoto novamente.



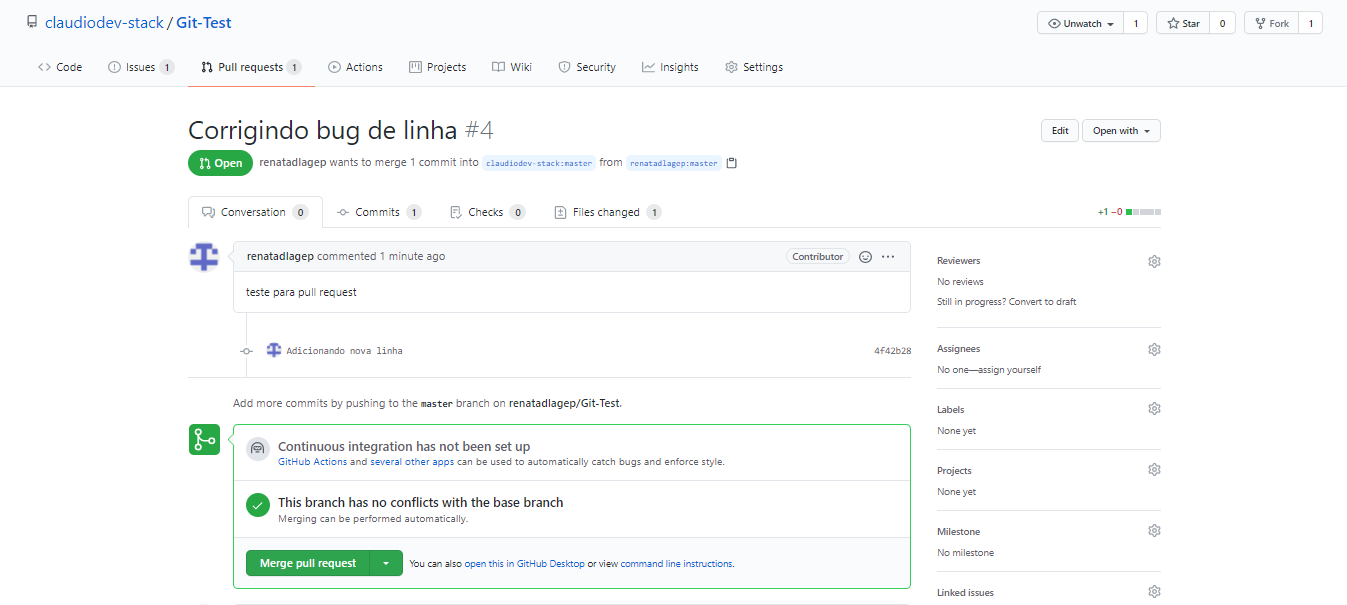
**PULL REQUEST**

Para enviar as alterações feitas dentro da fork para o repositório original, ou seja, o dono do repositório que você utilizou para criar ou reparar o código é necessário criar um pull request (pedido de envio) na sua conta github. Feito isso, basta clicar no botão create pull request, será enviado para o repositório original uma autorização de merge do documento.



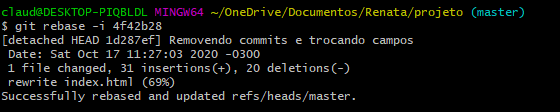
 

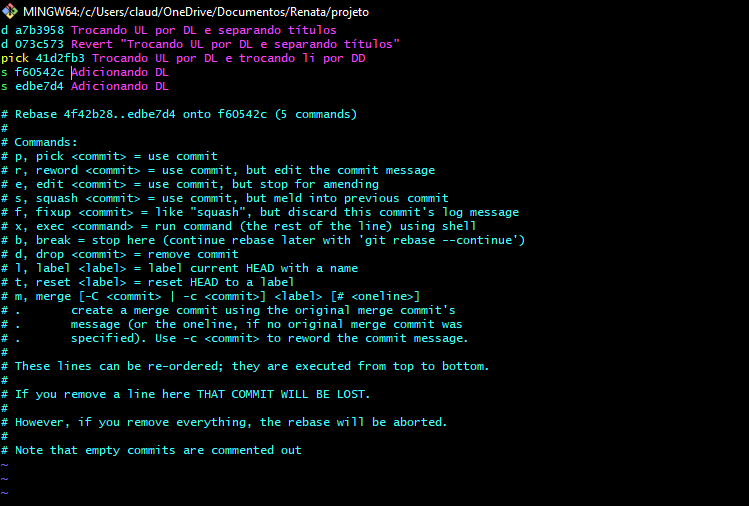
O dono do repositório original vai entrar em sua conta e com isso vai em “Pull Request”, vai exibir se vai exisir algum pedido de envio para autorizar, clicando em “Merge Pull Request” o seu código vai ser ajustado para o novo código criado ou reparado.

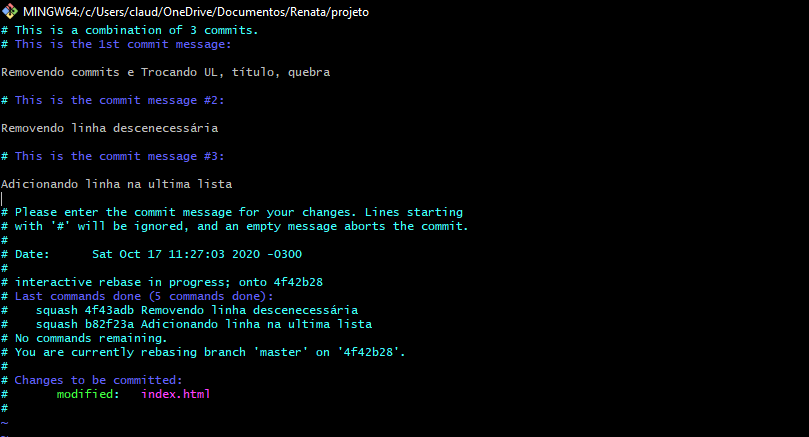


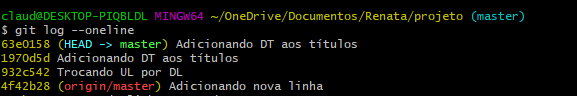
**UNINDO COMMITS**

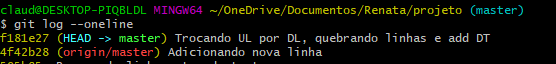
Quando utilizamos commits desnecessários como quebra de linhas ou outros exemplos, é possível unir esses commits para que em um caso de pull request o usuário que vai validar o pull request não perca tempo com diversos commits que podem ser resumidos em um único commit.   
Para isso é necessário utilizar o rebase sendo que você pode setar o HEAD dele e colocar a posição do commit ex: “git rebase HEAD~3” ou então apontar no Hash Code abaixo do que você vai modificar. Ele vai levar para o exemplo (imagem:2), lá você pode configrar na frente dos commits o comando que deseja fazer com cada commit, por exemplo “s” na frente significa squash que seria juntar os commits “pick” ou “p” é o commit que ficará como o principal. Feito isso “:”, “x” e na próxima tela se quiser editar os comentários e depois “:”, “x” ele vai retornar para a primeira imagem indicando que o rebase foi feito, quando verificar no git log os commits criados vão virar um único commit.





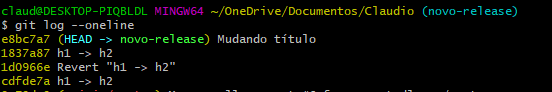


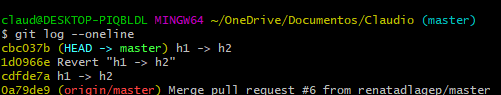


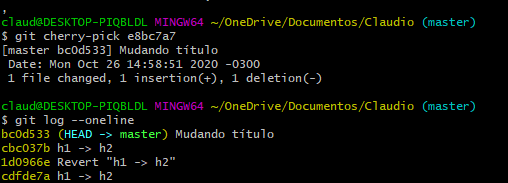


**TRABALHANDO COM COMMIT’S ESPECÍFICOS**

Quando é necessário localizar um commit específico que eu desejo continuar a partir daquele ponto é necessário utilizar o comando cherry-pick, só preciso localizar a branch (ex:novo-release) e o commit (e8bc7a7) e depois chamar da minha branch (master) para que seja atualizada para aquele ponto. Ex: Bug’s em commits futuros, continuar projeto daquele commit, etc.



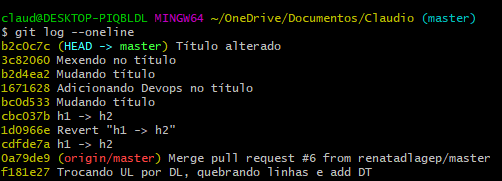


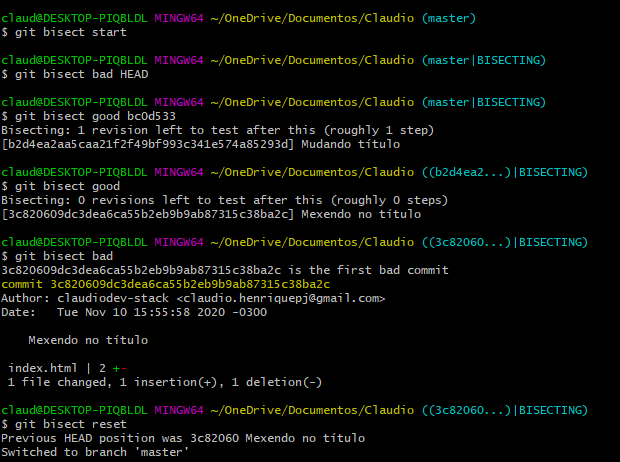


**IDENTIFICANDO BUG’S NOS ARQUIVOS**

Com o git bisect é possível localizar mudanças feitas no código e encontrar possíveis bugs, para isso é necessário startar o bisect e depois localizar qual o momento do commits que começou a ocorrer as falhas “bad” nos arquivos, no exemplo em imagem iniciei a partir do HEAD. Com isso colocando o bisect como “good” apontando para o ultimo commit que de fato “imaginei” que estava correto e afirmando com o good as revisões, e caso pesquisamos novamente o bad ele vai informar o ultimo commit que foram feitas as alterações dentro desses commits, utilizando o reset você sai do modo Bisect e o código volta para a ultima alteração feita.   
Caso necessário poderá realizar um revert no HASH indicado no “bad” e voltar para o commit que deseja.

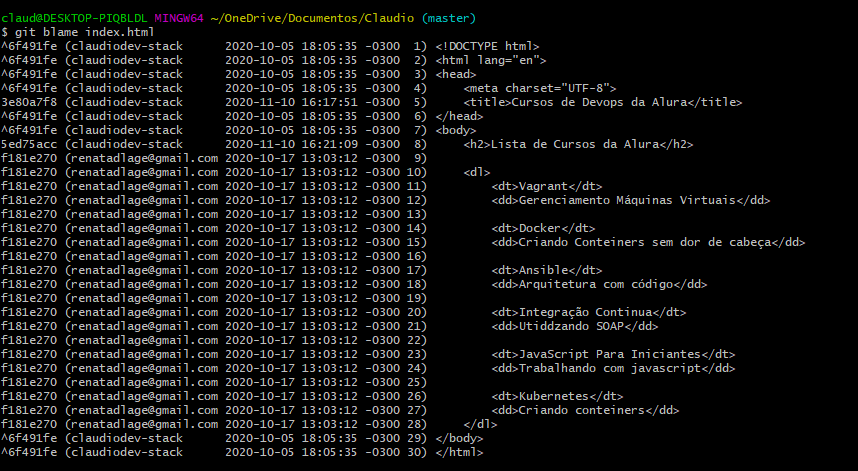






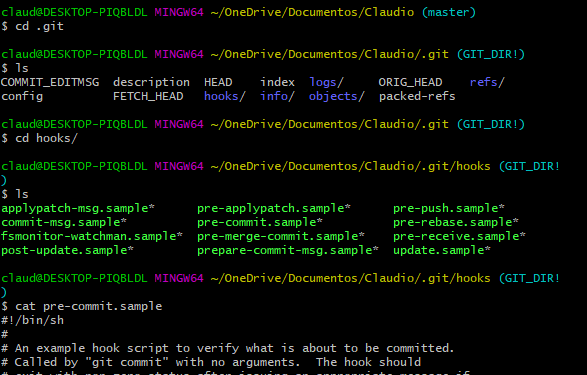
**IDENTIFICANDO BUG’S POR USUÁRIOS**

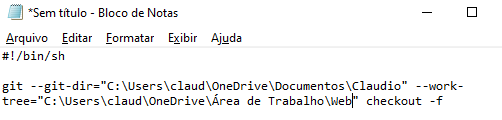
O comando git blame consegue identificar os usuários já configurados com o git config que fizeram alterações dentro dos arquivos e o período que foi feito. Observação: O comando não é para localizar um “culpado”, mas sim para ajudar um companheiro a melhorar o código ou até aprender com mudanças feitas por usuários mais experientes.



**HOOKS E DEPLOYS COM GIT**

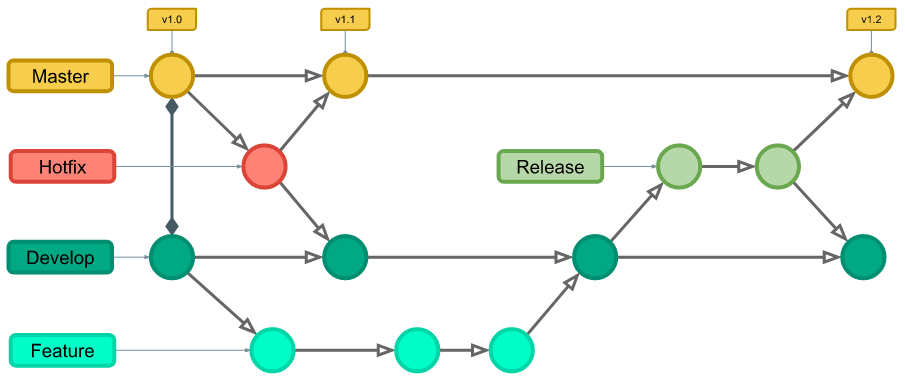
Dentro da pasta do .git existe a pasta hooks e dentro dela existem exemplos de comandos com shell script, você pode criar o seu evento shell script automatizado pelo git. Por exemplo, criar um deploy do seu arquivo da pasta LOCAL para a pasta WEB “Servidor”, sempre que você comitar o arquivo ele vai criar uma cópia na pasta WEB “Servidor” e com isso você consegue visualizar diretamente o arquivo com as mudanças feitas.

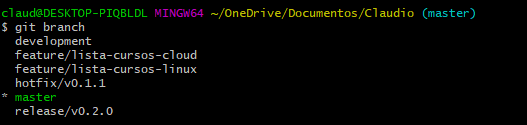




**BOAS PRÁTICAS COM BRANCHES (GIT FLOW)**

Nessa parte não existe nenhum comando novo, mas sim uma forma prática e correta muito usada pela comunidade de desenvolvedores. Geralmente é comum utilizar somente a branch master para se trabalhar em projetos pessoais, mas o correto em grupos são criar branchs de “produção”, “teste”, “desenvolvimento”, etc. Com isso utilizar essas branches para fazer as devidas mudanças nos arquivos e assim passar para a escala de branches (ex:img1), vale lembrar de manter a branch de desenvolvimento atualizada. Caso seja necessário fazer alguma mudança na branch de produção, nunca fazer nela! Crie uma nova branch de teste.





**FERRAMENTAS VISUAIS COM GIT**

No próprio site do git você encontra exemplos de ferramentas visuais melhores para observar o fluxo de atividades de uma forma melhor. Os três mais conhecidos e usados são: GIT COLA, GIT DESKTOP, GITKRAKEN.

Git cola: Entre os 3 foi um dos primeiros criados, talvez não tenha um design atual, etc. <https://git-cola.github.io/downloads.html>   
Git Desktop: É uma ferramenta boa, porém o que falta nele é um design com uso de GIT FLOW  
<https://desktop.github.com/>   
GitKraken: Essa ferramenta já é am ais completa dentre as 3. <https://www.gitkraken.com/download>   
  
Todos os 3 são bastante utilizados pela comunidade, mas existem outros e o gosto fica a critério de cada um.

Exemplo: GITKRAKEN

