**Quando devemos commitar um código?**

Não existe uma regra que informa a melhor hora para efetuar um commit, mas existe uma recomendação que diz que é melhor commitar um código que funciona, nunca devemos commitar um código que não seja funcional. Ou seja, a cada alteração funcional, ou significativa no código devo gerar um novo commit, não seria interessante gerar o commit apenas no final da minha feature (funcionalidade).

Alguns comandos para acompanhamento dos commits que são bastante úteis são, o git log, o git log -p (esta opção vai me dar as informações mais detalhadas de cada commit).

Para fazer o merge de um projeto no git, eu devo estar na branch que eu quero fazer merge, por exemplo tenho duas branches uma “Master”, e outra “Titulo”, e quero fazer o merge dessas duas branches, para isso eu devo estar na branch Master, e executar o comando git merge titulo, assim o git vai criar um novo commit desse merge e me dará a opção de editar a mensagem desse novo commit, ao editar essa mensagem caso eu queira mudar a mensagem, eu pressiono :x e enter.

O comando para que possamos trazer os commits de outra branch para a branch principal (master), basta executar o mesmo processo do git merge, lembrando que para isso eu devo estar na branch master, e executar o comando git rebase titulo. Assim o git vai entender que o último commit feito na branch master será meu commit atual, ou seja que eu estou trabalhando nessa branch.

Para reverter alterações no git eu tenho três opções, que são a de cancelar as alterações antes de adicionar o arquivo para commit, executando o comando git checkout -- <nome do arquivo>, ou caso eu já o tenho adicionado para fazer o commit, posso executar o comando git reset HEAD <nome do arquivo>, e caso eu já tenha adicionado o arquivo, e feito o commit dele, posso executar o comando git revert <hash do commit>. Para que eu possa obter o hash do commit eu posso executar o comando git log, e ver o hash logo à frente do nome “commit”. Essas funcionalidades são bastente úteis caso eu precise desfazer alguma mudança.

Temos ainda a opção de trabalhar em um código posteriormente, com o comando git stash, esse comando “salva” o estado atual do meu arquivo, permitindo que eu trabalhe em outras funções. O comando git stash list, exibe para mim tudo o que está salvo para que seja trabalhado depois, e sua exibição é como de um array, começando em 0. Para que eu possa trabalhar com um item específico do meu stash eu posso executar o comando git stash apply <posição do meu trabalho pendente>, mas depois eu preciso removê-lo com o comando git stash drop <posição do meu trabalho pendente>, mas caso eu tenha apenas um item no meu stash eu posso executar diretamente o comando git stash pop, esse comando aplica o meu trabalho pendente, e remove este item do meu stash

No git é possível que eu volte meu projeto para um estado determinado apenas com o número do commit (hash), executando o comando git checkout <numero do hash>, posso voltar meu projeto para o ponto desse commit, mas é importante lembrar que estarei “desanexado” do projeto como um todo, assim o git cria “temporariamente” uma branch para que eu trabalhe no projeto no ponto em que eu voltei, mas todo e qualquer commit que for feito nessa branch será descartado, assim se tornando necessário que eu crie uma nova branch para que o git entenda que eu preciso que todas as mudanças feitas a partir daquele ponto precisam ser salvas. Dessa forma terei as branches que já estarão criadas no decorrer do trabalho, mais a branch que foi criada a partir dessa volta nos commits.

O git me permite ver as diferenças entre os commits, com o comando git diff <numero do primeiro hash>..<numero do segundo hash> eu posso ver as alterações dos commits, realizados nesse intervalo. Caso eu precise pausar o meu trabalho mas não o adicionei ainda para fazer o commit, eu posso executar diretamente o comando git diff, sem a necessidade dos hashs dos commits anteriores, mas a partir do momento que eu adiciono o arquivo para commitar, este comando não mais me exibe essas diferenças.

Para criar novas versões no git, eu executo o comando git tag -a <nome da versão>, posso utilizar qualquer nome como versão. Posso ainda adicionar uma mensagem para minha tag, acrescentando o parâmetro -m, e entre parêntese escrever minha mensagem. Para que eu possa enviar a minha tag para o github ou um repositório local eu executo o comando git push origin <nome da versão>

O arquivo .gitignore, serve para que o git não mais “rastreie”, os arquivos que eu determinar dentro desse arquivo. Sua estrutura é bastante simples, bastando apenas eu criar o arquivo, exatamente com este nome **.gitignore** e ao abrir este arquivo eu devo especificar para este arquivo (para ele especificar para o git), quais arquivos ele deve ignorar, lembrando que para isso, eu devo informar o nome do arquivo com a extensão dele.

**Diferença entre git pull e git fetch**

***Git pull***, executa o comando para atualizar nosso repositório local a partir de um repositório remoto. Este comando ao ser executado, faz automaticamente a atualização da branch fazendo um merge.

***Git fetch,*** executa o comando para atualizar o repositório local a partir de um repositório remoto, mas, ao ser executado faz a atualização da branch mas **não** faz o merge de possíveis atualizações que foram feitas no repositório remoto.