=======================================================

***O que é versionamento de Código?***

=======================================================

O versionamento é o processo de criar novas versões de um código toda vez que existir uma mudança significativa nele. De maneira geral, todo projeto de desenvolvimento é feito por etapas, sendo que as funcionalidades são incrementadas aos poucos, enquanto o código está sendo desenvolvido, o programador vai criando as versões conforme melhora, muda ou implementa algo. Depois, é possível voltar às versões anteriores caso seja necessário.

Repositório

Um repositório nada mais é que uma pasta onde estão todos os arquivos do nosso projeto, incluindo nosso versionamento. Normalmente trabalhamos com repositórios hospedados em algum lugar, como um servidor ou algum serviço de hospedagem, pois trabalhamos em grupos que podem ir enviando suas alterações para o projeto e cada outra pessoa pode baixar essa alteração em sua máquina.

Commits

Commits são como fotos da última versão do nosso código. Eles carregam tudo o que foi alterado em nosso projeto para que, quando precisarmos, possamos voltar ao commit onde possuíamos a versão que gostaríamos de utilizar.

Histórico

O histórico é a união dos nossos commits.

Branches

Branches são uma maneira de organizar o nosso trabalho com versionamento de código. Nós possuímos uma branch chamada master e nesta branch temos a versão oficial do nosso projeto. Quando vamos fazer uma alteração, podemos criar uma ramificação da master e trabalhar somente nela, sem afetar o código oficial. Ao finalizarmos nosso trabalho na nossa ramificação, então podemos juntar o nosso trabalho com a master.

Quando criamos uma nova branch, ela é, inicialmente, uma cópia de outra. Neste caso é uma cópia da master. Então ela carrega todos os commits da master e a partir daí teremos novos commits inseridos somente na nova branch. Ao juntarmos tudo, então temos a versão final do projeto. Cada branch carrega seu próprio histórico e nós só transformamos isso em uma história oficial quando juntamos com a branch oficial.

Inicializando um Repositório

Quando iniciamos um repositório, automaticamente temos uma branch master e estamos agora nessa branch. Trocar de branches é como trocar de pastas, você precisa rodar um comando para criar elas e outro para ir para essa branch.

Se fossemos trabalhar com um repositório remoto (um repositório que está hospedado em um servidor) seria necessário utilizar um comando para baixar este projeto ao invés de começar um do zero.

A cada nova alteração que fazemos, que seja uma versão do nosso software, podemos fechar um commit e guardar isso no histórico do nosso projeto.

Estados do nosso trabalho com Git

Working Directory

O working directory possui o estado atual do nosso código. Aquilo que estamos alterando neste exato momento.

Staging Area

A staging area é uma área de preparação antes de fechar um commit de fato. Nós podemos enviar alterações para o stage e continuar trabalhando em nosso código ao invés de fazer um commit por vez e precisar voltar commits. Podemos somente trabalhar de working para stage e de stage para working até que nosso trabalho esteja completamente pronto e assim fechamos um commit.

Git Directory

Este é de fato o nosso histórico. É onde o Git guarda todos os nossos commits e são eles que são enviados para o repositório remoto e compartilhado com outras pessoas. Nem o working directory, nem o stage são compartilhados, esses são estados locais. O Git directory é de fato o nosso banco de objetos do Git.

Criando o commit

Temos o nosso arquivo sendo acompanhado (tracked) pelo Git adicionado em stage. Sabemos que queremos sim guardar uma versão deste arquivo para enviar para outras pessoas ou mesmo para deixar guardado no histórico, podemos então rodar o comando git commit.

Lembre-se de colocar mensagens descritivas na legenda do seu commit, pois isso fará toda diferença se algum dia você precisar reverter algo que foi feito.

Ex: git commit -m "adiciona o arquivo exercicio-de-git.txt".

Lendo o histórico do Git

Assim que nosso commit foi realmente fechado, o Git moveu as alterações para o Git Directory e agora temos um comando que nos ajuda a ver o que tem lá dentro. Este comando é o git log.

Se houvesse mais commits, iria aparecer uma lista muito maior. Para sair do leitor de histórico do Git, basta pressionar a letra q no seu teclado.

Criando branches

Agora imagina que queremos criar uma área de testes antes de consolidar nosso trabalho na branch master. Vamos então utilizar o comando git checkout para criar e trocar de branch para que o histórico da master não fique “feio”. Lembre-se: o histórico é algo muito importante e funciona como uma documentação do nosso código, por isso não o deixe ficar ruim.

A sintaxe do comando git checkout, para criar uma branch e mudar para ela, é: git checkout -b nome\_da\_branch. Rodamos o comando git checkout com o parâmetro -b e o nome da branch que será criada. Se desejamos trocar de uma branch para a outra, rodamos somente o git checkout nome\_da\_branch.

O comando checkout além de trocar de branchs, poderia também reverter nosso código para uma versão anterior se utilizarmos, ao invés do nome de uma branch, o hash (id) de um commit antigo. Mas por enquanto foque em criar a branch, vamos aprender a reverter nosso trabalho mais abaixo.

Verificar que estamos agora em outra branch, não a master: git status

Juntando branches

Para juntar duas branches, precisamos rodar o comando git merge.

A sintaxe do comando é: git merge nome\_da\_outra\_branch.

Para que o git merge funcione, precisamos estar na branch que irá receber os commits da outra. No nosso exemplo, para receber o novo commit da branch arquivo-com-meu-nome na master, precisamos estar na master.

Para isso precisaríamos do comando git checkout master, mas nós já rodamos ele no passo anterior. Agora podemos executar diretamente o git merge.

Repare que, ao rodar este comando, ele nos diz que a nossa branch foi atualizada, recebendo um outro commit, com mais um arquivo com uma linha adicionada, assim como na parte em que aprendemos sobre criação de commits.

Deletando branches

temos o merge na master, podemos então deletar ela, pois ela não servirá para mais nada.

Para sabermos quantas branches temos em nosso repositório, podemos rodar o comando git branch

O asterísco ( \* ) informa em qual branch estamos neste momento.

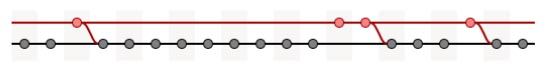
Para deletar uma branch, rodamos o comando git branch -d nome\_da\_branch. Bem parecido com o git checkout, quando passamos o parâmetro -d no git branch, ao invés de listar as branches, ele deleta a que informarmos.

Revertendo as coisas

Imagine agora que, as alterações que nós acabamos de fazer estragaram o nosso código. Juntar o arquivo que veio da branch arquivo-com-meu-nome fez com que o nosso software saísse do ar. Agora que vemos todo o poder do versionamento de código. Podemos reverter essas alterações com o comando revert.

Para reverter o que acabamos de fazer precisaremos rodar o comando git revert numero\_do\_commit.

No sistema centralizado, os programadores trabalham no mesmo ramo, quer este seja principal ou de manutenção. É um processo simples e fácil de compreender e até alguém com um conhecimento limitado do controle de versão, consegue entender-se com a restante equipa.



Já no caso do sistema distribuído, as coisas são um pouco mais complexas. A estrutura peer-to-peer, os ramos privados e as fusões que aparentam estar desorganizadas podem incutir ao projeto uma ideia de confusão inicial.



Contrariamente àquilo que ocorre na versão centralizada, nos DVCS não se trata apenas de fazer commit e update. Todos os programadores envolvidos têm que ter um conhecimento mais vasto das ferramentas em uso e, idealmente, também de um modelo de trabalho com padrões de fluxos a seguir.

