**PRO GIT – Everything you need to know about Git**

(2ª edicação – Apress)

Scott Chacon

Bem Straub

1.CONTROLE DE VERSÃO

# Sobre Controle de Versão(VCS)

Controle de Versão é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo para que você possa lembrar versões específicas mais tarde. Ele permite que você reverta para estado anterior determinados arquivos ou um projeto inteiro, compare as mudanças ao longo do tempo, veja quem modificou pela última vez algo que pode estar causando um problema, quem introduziu um problema, quando, e muito mais. Usar um VCS também significa que se você estragar tudo ou perder arquivos, você pode facilmente recuperar.

# Sistemas Centralizados de Controle de Versão

Estes sistemas têm um único servidor que contém todos os arquivos de controle de versão, e um número de clientes que usam arquivos a partir desse lugar central. Esta configuração oferece algumas vantagens, como por exemplo, todo mundo sabe, até certo ponto o que todo mundo no projeto está fazendo e os adms têm controle sobre quem pode fazer o que. No entanto, esta configuração também tem algumas desvantagens graves. O mais óbvio é o ponto único de falha que o servidor centralizado representa é que qualquer coisa que acontecer com o Servidor impede o trabalho de continuar. Sempre que se tem a história do projeto em um único lugar, há o risco de perder tudo.

# Sistemas Distribuídos de Controle de Versão

-- GIT, Mercurial, Bazaar, Darcs

Neste sistema clientes não somente usam o estado mais recente dos arquivos: eles duplicam localmente o repositório completo. Assim, se qualquer servidor morrer, e esses sistemas estiverem colaborando por meio dele, qualquer um dos repositórios de clientes podem ser copiados de volta para o servidor para restaurá-lo. Cada clone é de fato um backup completo de todos os dados.

Além disso, muitos desses sistemas trabalham muito bem com vários repositórios remotos, tal que você possa colaborar em diferentes grupos de pessoas de maneiras diferentes ao mesmo tempo dentro do mesmo projeto.

A principal diferença entre o GIT e outros VCS é que esses outros sistemas armazenam informações como uma lista de mudanças nos arquivos; tratam a informação como um conjunto de arquivos e as mudanças feitas a cada arquivo ao longo do tempo.

Git trata seus dados mais como um conjunto de imagens de um sistema de arquivos. Toda vez que você fizer um commit, ou salvar o estado de seu projeto no Git, ele basicamente tira uma foto de todos os seus arquivos e armazena uma referência para esse conjunto de arquivos. Para ser eficiente, se os arquivos não foram alterados, Git não armazena o arquivo novamente, apenas o link para o arquivo idêntico anterior já armazenado. Git trata seus dados mais como um fluxo do estado dos arquivos.

Para pesquisar o histórico do projeto, o Git não precisa sair para o servidor para obter a história e exibi-lo para você, ele simplesmente faz a leitura diretamente do banco de dados local. Isto significa que você vê o histórico do projeto quase que instantaneamente. Se você quiser ver as alterações introduzidas entre a versão atual de um arquivo e o arquivo de um mês atrás, Git pode procurar o arquivo de um mês atrás e fazer um cálculo de diferença local, em vez de ter que pedir a um servidor remoto para fazê-lo ou puxar uma versão mais antiga do arquivo do servidor remoto para fazê-lo localmente.

Isto também significa que há muito pouco que você não pode fazer se estiver desconectado ou sem VPN. Se você estiver em um avião ou um trem e quiser trabalhar um pouco, você pode fazer commits até conseguir conexão de rede e enviar os arquivos.

Tudo no Git passa por uma soma de verificações (checksum) antes de ser armazenado e é referenciado por esse checksum. Isto significa que é impossível mudar o conteúdo de qualquer arquivo ou pasta sem que Git saiba. Você não perderá informação durante a transferência e não receberá um arquivo corrompido sem que o Git seja capaz de detectar.

O mecanismo que o Git utiliza para esta soma de verificação é um hash SHA-1. SHA-1 é uma sequência de 40 caracteres composta de caracteres hexadecimais (0-9 e-f) e é calculada com base no conteúdo de uma estrutura de arquivo ou diretório no Git. O Git armazena tudo em seu banco de dados não pelo nome do arquivo, mas pelo valor de hash do seu conteúdo.

# Três Estados no Git

O Git tem três estados principais que seus arquivos podem estar: **commited, modified e staged**. Commited significa que os dados estão armazenados de forma segura em seu banco de dados local. Modified significa que você alterou o arquivo, mas ainda não fez o commit no seu banco de dados. Staged significa que você marcou a versão atual de um arquivo modificado para fazer parte de seu próximo commit.

# Três seções principais de um projeto Git

Diretório Git, diretório de trabalho e área de preparo.

O diretório Git é onde o Git armazena os metadados e o banco de dados de objetos de seu projeto. Esta é a parte mais importante do Git, e é o que é copiado quando você clona um repositório de outro computador.

O diretório de trabalho é uma simples cópia de uma versão do projeto. Esses arquivos são pegos do banco de dados compactado no diretório Git e colocados no disco para você usar ou modificar.

A área de preparo é um arquivo, geralmente contido em seu diretório Git, que armazena informações sobre o que vai entrar em seu próximo commit. É por vezes referido como “índice”, mas também é comum referir-se a ele como área de preparo (staging area).

1—Você modifica arquivos no seu diretório de trabalho.

2—Você prepara os arquivos, adicionando imagens deles à sua área de preparo.

3—Você faz commit, o que leva os arquivos como eles estão na área de preparo e armazena essa imagem de forma permanente para o diretório do Git (repositório local).

Se uma versão específica de um arquivo está no diretório Git, é considerado commited. Se for modificado, mas foi adicionado à área de preparo, é considerado preparado. E se ele for alterado depois de ter sido carregado, mas não foi preparado, ele é considerado modificado.

# Comandos iniciais do Git - Configuração

git config - - system (para todos os usuários no sistema e todos os seus repositórios)

git config - - global (somente para o seu usuário)

$git config - - global user.name “nome”

$git config - - global user.email “email”

Você precisará fazer isso somente uma vez se tiver usado a opção - - global, porque então o Git usará esta informação para qualquer coisa que você fizer naquele sistema.

$git config - -global core.editor nomeEditor (escolher o editor de texto padrão)

$git config - - list (para listar todas as configurações do Git)

$git config <key> (para testar um chave específica como <user.name>)

2. FUNDAMENTOS DE GIT

# Inicializando um repositório em um diretório existente

Ir para o diretório do projeto:

Linux: $ cd/home/user/your\_repository

Windows: $ cd/c/user/your\_repository

$ git init (cria um novo subdiretório chamado .git que contém todos os arquivos necessários de seu repositório. Neste ponto, nada no projeto é monitorado ainda)

$ git clone url (para clonar um repositório existente basta usar este comando e adicionar a URL de onde se deve clonar – GitHub. O nome do repositório neste caso será o mesmo que o do clone)

$ git clone url NomeDiretirioDeDestino (ex: $git clone <https://github.com/libgit2/libgit2> mylibgit)



\*Arquivos rastreados são arquivos que foram incluídos no último snapshot; eles podem ser não modificados, modificados ou preparados (adicionados ao stage). Em resumo, arquivos rastreados são os arquivos que o Git conhece.

\*Quando você clona um repositório pela primeira vez, todos os seus arquivos serão rastreados e não modificados já que o Git acabou de obtê-los e você ainda não editou nada.

\*Assim que você edita alguns arquivos, Git os considerada modificados, porque você os editou desde o seu último commit. Você prepara os arquivos editados e então faz commit das suas alterações, e o ciclo se repete.

$ git status (para ver qual o status dos arquivos do seu repositório)

$ echo “texto para aparecer dentro do Readme” >> readme.md (criar um arquivo README a partir do terminal/.md arquivo Markdown)

$ echo ‘texto para aparecer dentro do readme’ > readme (sem definição do tipo de arquivo)

$ echo ‘teste’ >> readme.md

Obs: as três maneiras de adicionar o arquivo readme funcionaram do mesmo modo! Aspas simples e duplas e um ou dois colchetes angulares

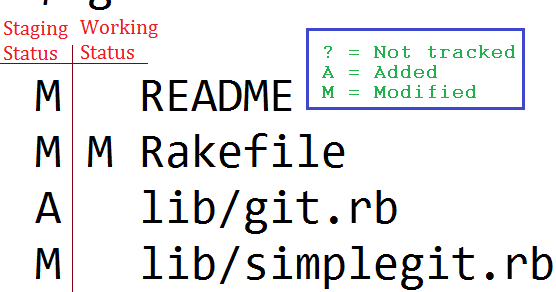
Ao identificar um arquivo não rastreado na pasta do repositório, com o git status, o Git irá avisar, contudo, não vai passar a incluir o arquivo nos seus commits a não ser que você o mande fazer isso explicitamente. O comportamento do Git permite que você só inclua arquivos que você realmente deseja comitar.

$ git add nomedoarquivo (para iniciar o rastreamento de um arquivo específico do repositório./ exemplo $ git add REAME.md)

O comando git add é um comando de múltiplos propósitos: serve para começar a rastrear arquivos e também para outras coisas, como marcar arquivos que estão em conflito de merge como resolvidos. Pode ser útil pensar nesse comando mais como “adicione este conteúdo ao próximo commit”

Supondo que criei um arquivo contributing.md no meu repositório e então adicionei ele, git add, significa que agora ele está preparado para ser comitado. Caso seja feita uma alteração neste arquivo antes de executar o commit propriamente dito, quando git status for executado novamente, o arquivo contributing.md terá dois estágios diferentes, um pronto para ser comitado e outro modificado mas não preparado para ser comitado. O arquivo contributing.md estará listado como preparado (staged) e também como não-preparado(unstaged). Acontece que o Git põe um arquivo no stage exatamente como ele está no momento em que você executa o comando git add. Se você executar git commit agora, a versão contributing.md que vai para o repositório é aquela de quando você executou git add, você tem que executar git add de novo para por sua versão mais recente no stage.

$git status –s / $ git status –short (a exibição de status será mais simples:



*?? = arquivos não rastreados*

*A = novos arquivos adicionados à área de stage*

*M = arquivos modificados*

No exemplo da figura acima, o arquivo README foi modificado e está na área de staged pronto para ser comitado. O arquivo Rakefile foi modificado, adicionado na área de stage, ou seja, está pronto para ser comitado, contudo, ele foi modificado novamente no repositório e não foi adicionado à área de stage (mesmo caso do exemplo do arquivo contributing.md acima). O arquivo lib/git.rb é um arquivo novo adicionado para a área de stage. O arquivo lib/simplegit.rb foi modificado e está na área de stage pronto para ser comitado, e não foi alterado na área de trabalho até este momento.

# Ignorando Arquivos

Frequentemente terá uma classe de arquivos que você não quer que sejam adicionados automaticamente pelo Git e nem mesmo que ele mostre como não-rastreados. Geralmente, esses arquivos são gerados automaticamente. Você pode criar um arquivo chamado .gitignore, contendo uma lista de padrões de nomes de arquivos que devem ser ignorados. Configurar um arquivo .gitignore, antes de você começar um repositório, geralmente é uma boa idéia para que você não inclua acidentalmente em seu repositório Git arquivos que você não quer.

# Visualizando Alterações Dentro e Fora do Stage

$git diff (Semelhante ao comenado git status, porém mais completo. Utilizado quando você quer responder exatamente o que foi alterado, não apenas quais arquivos foram alterados. Este comando responde as seguintes perguntas: O que você alterou mas ainda não mandou para o stage? O que está em stage, pronto para comitar? Apesar do comando git status também responder a esta pergunta de forma genérica, listando os nomes dos arquivos, o comando git diff exibe exatamente as linhas que foram adicionadas e removidas – o patch, como costuma se chamar.)

$git diff --staged (Mostra as alterações que você mandou para stage e que entrarão no próximo commit. Comparada as alterações que estão no stage com o último commit.)

É importante notar que o git diff sozinho (sem argumentos) não mostra todas as alterações feitas desde o seu último commit – apenas as alterações que ainda não estão no staged. Isso pode ser confuso porque, se você já tiver mandado todas as suas alterações para staged e utilizar o comando git diff ele retornará vazio. Também pode ser utilizado para mostrar as diferença entre um arquivos que está no stage, foi modificado e ainda não foi para stage novamente.