Controle de versão utilizando Git – Software Engineering Institute

Georg Buske (georg@s-e-i.biz)
João Carlos Fachini (jfachini@s-e-i.biz)
João Rodolfo Brambilla (jbrambilla@s-e-i.biz)

Sumário

1 – História

Tudo começou como uma necessidade do projeto Linux. Projeto este, que utilizava o BitKeeper para controle de versão.
Mesmo servindo aos propósitos do projeto por um tempo, o BitKeeper possuia como principal empecilho ser um sistema proprietário. Tendo em vista a ideologia de código aberto seguida pelo projeto Linux, se tornou impossível continuar mantendo dependência de um software fechado em seu projeto.
Como alternativa a esse problema, alguns membros da equipe começaram a fazer um processo de engenharia reversa no BitKeeper o que acabou culminando no desenvolvimento do Git.

2 – Configurando o Git

Antes da criação do repositório e utilização do git, devemos configurá-lo. É necessário configurar a conta do usuário git definindo o seu nome de usuário e e-mail :

git config --global user.name “Seu Nome”
git config --global user.email “seu@email.com”

Opcionalmente, você também pode configurar o ambiente de linha de comando:

git config --global color.status auto
git config --global color.diff auto
git config --global color.branch auto

Para não precisar ficar pondo senha a cada commit, faça:

git config --global credential.helper cache

Padrão de 15 minutos de cache, para aumentar faça:

git config --global credential.helper 'cache --timeout=3600'

3 – Criação repositório

Nessa parte vamos simular a criação do lado “servidor”.
Vamos começar criando um repositório git (arquivo.git) na pasta: *branchMasterExemplo*
Abra o terminal e vá até essa pasta. Estando nela execute o comando:

git init

O próximo passo é adicionar os arquivos no respositório iniciado, para verificar como seu repositório está, execute:

git status

Ao usar git status, você perceberá que os arquivos estão na seção “untracked”, isso significa que eles ainda não estão no repositório git

Para resolver isso utilizaremos o comando:

git add .

Onde “.” significa tudo.

Agora ao utilizar o git status, você verificará que os arquivos foram para a seção “Changes to be commited”

O próximo passo será a realização do primeiro commit.

Para isso execute:

git commit -a -m”First Commit”

Tendo feito isso, você possui agora o seu repositório git pronto.

Você não realizará alterações nessa pasta para preservar sua integridade.

Vamos criar um diretório de trabalho que representará o nosso lado “cliente”.

Vá para o diretório onde você pretende adicionar seu diretório de trabalho.

Estando nela, execute o seguinte comando:

git clone pathTo..branchMasterExemplo/.git

Tendo feito isso você verá a seguinte mensagem:

“Checking connectivity... done”

4 – Master/ Branches

Dentro do seu diretório de trabalho você pode executar o comando git branch. Você visualizará a branch master que está no seu repositório git.

Agora vamos criar nossa primeira branch, para isso execute o comando:

git checkout -b branchA

A partir de agora se você executar o comando git branch, você perceberá que será mostrada a master e a branchA, além disso, a branchA estará selecionada com um caracter “*”.

Essa seleção significa que você está dentro da branchA, neste momento qualquer alteração feita, será realizada somente nessa branch, sem alterar a master!

Agora vamos testar a funcionalidade de deleção de uma branch.

Para isso utilizaremos o comando:

git branch -D branchA

Ao executarmos o comando git branch, podemos verificar que só restou a master.

Lembrando que essa deleção é apenas do lado “cliente”, para deletar de verdade tem que estar na pasta do servidor.

Depois da deleção devemos criar novamente a branch e alterar alguns arquivos nela. Isso é necessário para o nosso próximo passo.

5 – Commit

Faça alterações nos arquivos e volte para a master:

git checkout master

Agora estando na master, abra algum arquivo que havia sofrido alteração e veja que ele continua alterado. Isso ocorreu porque você não deu commit.

Toda vez que você quiser concluir as alterações em uma branch, você deve usar o commit.

Para corrigir o problema criado no exemplo vamos voltar para a branchA:

git checkout branchA

E estando nela vamos dar o commit:

git commit -a -m”salvando alterações na branchA”

Agora sim! Podemos ir para a master e verificar os arquivos alterados.

Eles estarão como eram antes das alterações, já que elas foram feitas na branchA e não na master.

De volta na branchA, podemos ver que o conteúdo dos arquivos foram alterados.

Agora ainda na branchA, vamos criar um arquivo novo. Criado o arquivo vamos dar commit.

Receberemos como saída a seguinte mensagem:

“nothing added to commit but untracked files present (use "git add" to track)”

Essa mensagem ocorre porque não adicionamos o arquivo no repositório git, apenas criamos ele enquanto arquivo.

Para adicioná-lo no git vamos usar:

git add newFile

Agora sim podemos usar o commit! Tendo feito o commit vamos retornar para o master.

No master podemos verificar que o novo arquivo não estará presente.

Voltando para o branchA, verificamos que nosso novo arquivo estará lá!

Agora vamos simular uma alteração indesejada no nosso repositório. Estando no branchA vamos alterar algum arquivo. Vamos imaginar que nos arrependemos da alteração e queremos deixar nosso branch como era antes. Para isso utilizamos:

git checkout nomeDoArquivoAlterado

Isso fará com que as alterações sejam descartadas.

Caso tivéssemos alterações indesejadas em mais de um arquivo poderíamos utilizar:

git checkout .

Agora vamos enviar nossa branch para o “servidor”. Para isso faremos:

git push origin branchA

Para conferir se o resultado está ok, vamos para nosso diretório branchMasterExemplo, onde iniciamos nosso repositório.

No diretório, utilizamos:

git branch

6 – Push

Agora vamos enviar nossa branch para o “servidor”. Para isso faremos:

git push origin branchA

Para conferir se o resultado está ok, vamos para nosso diretório branchMasterExemplo, onde iniciamos nosso repositório.

No diretório, utilizamos:

git branch

Se tudo estiver ok, teremos:

master

branchA

Podemos ainda ir para o branchA e verificar os arquivos que tinhamos criados/ alterados no lado cliente.

7 - Fetch

Ainda no diretório branchMasterExemplo, na branchA vamos alterar o arquivo Hello.php, vamos adicionar linhas a ele. Feito isso commit as alterações.

Agora vamos para o diretório de trabalho, nele vamos usar o comando:

git fetch origin branchA

8 – Merge

Entrando no branchA podemos ver que o arquivo Hello.php está sem alteração alguma.

Isso porque o fetch não faz merge.

Para ver as alterações, vamos fazer o merge do nosso branch local com o que veio pelo fetch, para issos precisamos pegar a hash que se refere a essa branch, ela está no arquivo FETCH_HEAD

Vamos acessá-lo e copiar a hash:

vi .git/FETCH_HEAD

A hash será algo do tipo:

073c7a930b935150b292dbc17e3e3282505c77ce

Tendo a hash vamos fazer merge com nosso branch local.

git merge 073c7a930b935150b292dbc17e3e3282505c77ce

Pronto! Nosso arquivo Hello.php terá se fundido com o Hello.php vindo do servidor.

Percebemos porém que ocorreram conflitos. Veremos agora como resolvê-los!

9 - Conflitos

Ao abrirmos o arquivo Hello.php com o editor vi, veremos a seção head que consiste no estado do arquivo antes do merge, essa seção é iniciada por:

<<<<<<< HEAD

e terminada por:

=======

Depois dessa seção veremos a alteração trazida pelo merge. As alterações começam depois da seção head e vão até:

>>>>>>>hashBranch

O que estiver fora dessas duas seções são partes do código que não tiveram conflitos, então não precisarão de resolução alguma.

Para resolvermos conflitos devemos nos atentar se as alterações trazidas pelo merge são benéficas ou não. Caso queiramos mantê-las devemos apagar o que está na região head.

Caso desejamos deixar o arquivo como estava antes do merge, basta apagar a região pós head.

Tendo realizado as alterações no arquivo que estava com conflitos, vamos dar commit no branch.

Para isso:

git commit -a -m “Resolução conflitos”

Pronto! Resolvemos os conflitos causados pelo merge.

10 – Pull

O comando pull engloba duas operações já vistas, são elas: fetch e merge.

Para testa-lo vamos fazer:

git pull origin branchA

Verificamos então que o conflito que tínhamos resolvido voltou, isso se deve ao fato do comando ter baixado o branch e feito merge com nosso branch local.

11 – Diff

Com esse comando conseguimos ver as diferenças entre arquivos de branchs diferentes.

O comando obedece o seguinte padrão:

git diff file1 file2

Como exemplo estando na master, vamos ver a diferença entre o novo.php dela com o do branchA:

git branchA novo.php

Como saída teremos algo do tipo:

-- a/novo.php

++ b/novo.php

a/ representa nossa master e b/ a branchA, podemos notar que de a para b foram adicionados linhas no arquivo.

12 - Log

Com o comando log podemos ver o histórico de mudanças em um arquivo. Esse histórico pode ser resumido, apenas com dados de cada commit ou mais completo mostrando as alterações realizadas em cada commit.

Na versão resumida temos:

git log file

Para mais detalhes temos:

git log -p file

Ciclo de um commit