PAULO VICTOR SOUZA DE AZEVEDO

RELATÓRIO:

SEJA LIVRE, USE SOFTWARE LIVRE - GIT

RJ - RIO DE JANEIRO

2021

Controle de Versão

O Controle de versão possui duas partes básicas: O repositório e a área de trabalho. O repositório é onde se concentra todo o histórico de modificação do projeto, armazenando todas as alterações feitas entre cada novo versionamento do projeto inteiro. Enquanto que a área de trabalho é a onde estão cópias do repositório central que serão modificadas e desenvolvidas pelos programadores. (DIAS, 2016)

O versionamento centralizado, método utilizado antes da criação do Git, utilizava a topologia estrela, onde existe apenas um repositório central com várias cópias satélites da área de trabalho orbitando em volta. Qualquer mudança que fosse feita nas áreas de trabalho satélites, deveria necessariamente ser enviada para o repositório central, forçando uma comunicação bilateral, feita através dos comandos *commit* e *update* entre o repositório central e a área de trabalho específica. (DIAS, 2016)

Não existia comunicação entre as áreas de trabalho satélites, o que dificultava o gerenciamento de versões entre as áreas de trabalho. No processo de *commit* para o servidor central, era comum ter dados sobrescritos. Não era permitido também *commits* simultâneos para o repositório central entre as áreas de trabalho satélite, o que tornava o processo demorado já que existia “uma fila” de *commits* entre as áreas de trabalho. Além de que tornava a tarefa bastante manual caso um dos programadores quisesse utilizar ou testar alguma implementação que ainda não tivesse sido enviada para o repositório central. (FREITAS, 2010)

Em contrapartida, o Git, criado por Linus Torvalds, funciona no sistema de controle de versão distribuído. Onde existem vários repositórios autônomos e independentes (e não apenas um centralizado) que também possuem suas próprias áreas de trabalho, ou seja, cada programador possui seu próprio repositório de controle de centralizado, que pode ou não estar interligado com outros repositórios de controle centralizados. (DIAS, 2016)

Essa liberdade de ação, não sendo mais necessário uma comunicação constante com o repositório central, permitiu uma forma muito mais produtiva e colaborativa de trabalho entre os desenvolvedores.

GIT

No Git, como cada desenvolvedor pode ter seu próprio repositório, foi também implementada a capacidade de comunicação entre repositórios através dos comandos *pull* e *push.* Neste sistema não existe necessariamente um repositório central, porém é totalmente possível essa implementação em casos de uma necessidade de criação desse tipo de fluxo de trabalho caso seja um desejo dos desenvolvedores. (DIAS, 2016)

No método centralizado, por só existir um repositório, o controle de versão era feito através de um número identificador sequencial. Como no Git o processo de controle de versão pode ser individual, simultâneo e compartilhado, foi necessário a implementação de outro tipo de identificador.

O SHA-1, método identificador utilizado no Git, possui a capacidade de gerar um identificador de 40 dígitos hexadecimais, o que torna extremamente improvável o conflito de versões distribuídos entre letras e números para monitoramento e controle das versões alteradas. Qualquer mudança, mesmo que mínima, como um espaço em algum local do arquivo, gerará um novo identificador SHA para aquele código. (DIAS, 2016)

Como os nomes identificadores são bastantes extensos e são gerados frequentemente, é possível atrelá-los a uma etiqueta (*tag*), para que seja facilitado a identificação deles a posteriori e pelos outros desenvolvedores. Os repositórios Git que são públicos, também possuem um documento de orientação(*README.md*), que é inscrito em linguagem de marcação (*markdown*) onde se concentra todas as informações do funcionamento e funcionalidades do projeto. (RIZZO, 2020)

Outro método de trabalho inovado pelo Git foi através de *branches*, que são ramificações do projeto principal para que seja testado ou implementado novas modificações, sem que isso comprometa os arquivos alocados ao projeto principal. Usando comandos é possível criar uma nova ramificação do projeto (*branch)* e acessá-la para iniciar o desenvolvimento (*checkout)*. Usando esse método é possível criar várias ramificações do projeto que podem ter diferentes objetivos, como testes, implementações, entre outras funções. (DIAS, 2016)

Após já cessado o desenvolvimento em algum ramo auxiliar do projeto, é possível juntá-lo ao projeto base que está na ramificação principal, através do comando *merge*. Para que assim o projeto principal receba as modificações feitas

Alguns dos comandos básicos para o ciclo de trabalho no Git são:

- Copiar repositório: ***git pull***
- Fazer alterações para o stage: ***git add***
- Verificar alterações: ***git status***
- Desfazer alterações: ***git revert***
- Resolver conflitos de merge: ***git mergetool***
- Submeter alterações do stage para o repositório: ***git comitt***
- Enviar alterações do seu repositório para outro repositório: ***git push***

FERRAMENTAS ONLINE DE GERENCIAMENTO DE REPOSITÓRIOS

Com a popularização do uso da ferramenta Git para versionamento de código, também se criou a necessidade de gerenciamento desses repositórios online. A popularização de repositórios públicos, tornou bastante comum o desenvolvimento coletivo, possibilitando desenvolvedores de todo o mundo colaborarem entre si em seus projetos.

Hoje, no mercado existem várias ferramentas gerenciamento de repositórios online, porém as três mais populares são: GitHub, GitLab e Bitpocket.

O principal foco dessas ferramentas foi permitir o gerenciamento remoto dos repositórios, assim também facilitando o acesso do projeto de qualquer lugar, inclusive de computadores e notebooks variados, só sendo necessário o link do repositório, em caso de repositórios públicos, ou o acesso via *login* e senha, ou via SSH, para caso de acesso particular ou privado, para mais segurança. (RIZZO, 2020)

Essas plataformas online também permitem total visualização dos históricos de modificações. É possível visualizar por essas próprias ferramentas os *merges* e *pushs* conflituosos que foram feitos pelo mesmo desenvolvedor em diferentes momentos ou que foram feitas por programadores diferentes. Também é possível comunicação pelas próprias plataformas para resolução desses conflitos nos repositórios, decidindo qual das implementações faz mais sentido ao projeto.

CONCLUSÃO

O controle de versão é uma ferramenta básica e de extrema importância para o desenvolvimento de software. Sendo inclusive parte de exigências para melhorias do processo de desenvolvimento de várias certificações. (DIAS, 2016)

A criação do Git popularizou o compartilhamento e colaboração em desenvolvimento de software com um nível de qualidade no processo nunca antes vista, acelerando e aumentando a produtividade de toda cadeia que dependia de ferramentas de versionamento para seu crescimento.

A ferramenta adjacentemente fomentou a criação de vários softwares e plataformas online para gerenciamento de versionamento de códigos, que em muito ajudaram na popularização do código livre feito de forma segura e pouco conflituosa.

REFERÊNCIAS

DIAS, André Felipe. Conceitos Básicos de Controle de Versão de Software - Centralizado e Distribuído.Disponível em https://blog.pronus.io/en/posts/controle-de-versao/conceitos-basicos-de-controle-de-versao-de-software-centralizado-e-distribuido/. Acessado em 27/09/2021.

GIT, WIKIPÉDIA. Disponível em https://en.wikipedia.org/wiki/Git. Acessado em 27/09/2021.

RIZZO, NILTON JOSÉ. Video (2h18min). Seja livre, use software livre – GIT. Disponível em https://www.youtube.com/watch?v=ja4C7e8dC68. Acessado em 27/09/2021.

GIT, SITE OFICIAL. https://git-scm.com/. Acessado em 27/09/2021.