**UNIRON**

**União das Escolas Superiores de Rondônia**

PAULO JOSÉ DOS SANTOS

**ENGENHARIA DE SOFTWARE**

**CONTROLE DE VERSÕES GIT**

**PORTO VELHO/RO**

**2016**

**INTRODUÇÃO**

O presente trabalho tem por objetivo apresenta algumas características e conceitos de tecnologia em controle de versões, tem como foco principal o Git, uma excelente ferramenta para o controle e gerenciamento de versões de projetos diversos, além de ser uma tecnologia que tem crescido muito nos últimos tempos, também será apresentando seus principais comandos utilizados para o gerenciamento dos projetos versionados.

**Uma Breve História do Git**

Assim como muitas coisas boas na vida, o Git começou com um tanto de destruição criativa e controvérsia acirrada. O kernel (núcleo) do Linux é um projeto de software de código aberto de escopo razoavelmente grande. Durante a maior parte do período de manutenção do kernel do Linux (1991-2002), as mudanças no software eram repassadas como patches e arquivos compactados. Em 2002, o projeto do kernel do Linux começou a usar um sistema DVCS proprietário chamado BitKeeper.

Em 2005, o relacionamento entre a comunidade que desenvolvia o kernel e a empresa que desenvolvia comercialmente o BitKeeper se desfez, e o status de isento-de-pagamento da ferramenta foi revogado. Isso levou a comunidade de desenvolvedores do Linux (em particular Linus Torvalds, o criador do Linux) a desenvolver sua própria ferramenta baseada nas lições que eles aprenderam ao usar o BitKeeper. Alguns dos objetivos do novo sistema eram:

Velocidade

Design simples

Suporte robusto a desenvolvimento não linear (milhares de branches paralelos)

Totalmente distribuído

Capaz de lidar eficientemente com grandes projetos como o kernel do Linux (velocidade e volume de dados)

Desde sua concepção em 2005, o Git evoluiu e amadureceu a ponto de ser um sistema fácil de usar e ainda assim mantém essas qualidades iniciais. É incrivelmente rápido, bastante eficiente com grandes projetos e possui um sistema impressionante de branching para desenvolvimento não-linea

**Principais características**

Enfim, em poucas palavras, o que é Git? Essa é uma seção importante para assimilar, pois se você entender o que é Git e os fundamentos de como ele funciona, será muito mais fácil utilizá-lo de forma efetiva. À medida que você aprende a usar o Git, tente não pensar no que você já sabe sobre outros VCSs como Subversion e Perforce; assim você consegue escapar de pequenas confusões que podem surgir ao usar a ferramenta. Apesar de possuir uma interface parecida, o Git armazena e pensa sobre informação de uma forma totalmente diferente desses outros sistemas; entender essas diferenças lhe ajudará a não ficar confuso ao utilizá-lo.

**Snapshots, E Não Diferenças**

A maior diferença entre Git e qualquer outro VCS (Subversion e similares inclusos) está na forma que o Git trata os dados. Conceitualmente, a maior parte dos outros sistemas armazena informação como uma lista de mudanças por arquivo. Esses sistemas (CVS, Subversion, Perforce, Bazaar, etc.) tratam a informação que mantém como um conjunto de arquivos e as mudanças feitas a cada arquivo ao longo do tempo.

Git não pensa ou armazena sua informação dessa forma. Ao invés disso, o Git considera que os dados são como um conjunto de snapshots (captura de algo em um determinado instante, como em uma foto) de um mini-sistema de arquivos. Cada vez que você salva ou consolida (commit) o estado do seu projeto no Git, é como se ele tirasse uma foto de todos os seus arquivos naquele momento e armazenasse uma referência para essa captura. Para ser eficiente, se nenhum arquivo foi alterado, a informação não é armazenada novamente - apenas um link para o arquivo idêntico anterior que já foi armazenado.

**Quase Todas Operações São Locais**

A maior parte das operações no Git precisam apenas de recursos e arquivos locais para operar — geralmente nenhuma outra informação é necessária de outro computador na sua rede. Se você está acostumado a um CVCS onde a maior parte das operações possui latência por conta de comunicação com a rede, esse aspecto do Git fará com que você pense que os deuses da velocidade abençoaram o Git com poderes sobrenaturais. Uma vez que você tem todo o histórico do projeto no seu disco local, a maior parte das operações parece ser quase instantânea.

**Git Tem Integridade**

Tudo no Git tem seu checksum (valor para verificação de integridade) calculado antes que seja armazenado e então passa a ser referenciado pelo checksum. Isso significa que é impossível mudar o conteúdo de qualquer arquivo ou diretório sem que o Git tenha conhecimento. Essa funcionalidade é parte fundamental do Git e é integral à sua filosofia. Você não pode perder informação em trânsito ou ter arquivos corrompidos sem que o Git seja capaz de detectar.

O mecanismo que o Git usa para fazer o checksum é chamado de hash SHA-1, uma string de 40 caracteres composta de caracteres hexadecimais (0-9 e a-f) que é calculado a partir do conteúdo de um arquivo ou estrutura de um diretório no Git.

**Os Três Estados**

Agora preste atenção. Essa é a coisa mais importante pra se lembrar sobre Git se você quiser que o resto do seu aprendizado seja tranquilo. Git faz com que seus arquivos sempre estejam em um dos três estados fundamentais: **consolidado** (committed), **modificado** (modified) e **preparado** (staged). Dados são ditos consolidados quando estão seguramente armazenados em sua base de dados local. Modificado trata de um arquivo que sofreu mudanças mas que ainda não foi consolidado na base de dados. Um arquivo é tido como preparado quando você marca um arquivo modificado em sua versão corrente para que ele faça parte do snapshot do próximo commit (consolidação).

**Os principais comandos**

Comando para criar um novo repositório.

* **git init**

Comando para obtenha um repositório

* **git clone /caminho/para/o/repositório**

Quando estiver usando um servidor remoto, seu comando será:

* **git clone usuário@servidor:/caminho/para/o/repositório**

Quando for necessário adicionar um arquivo no repositório utiliza-se o comando abaixo:

* **git add <arquivo>**
* **git add \***

Esses comandos citados acima, é o primeiro passo no fluxo de trabalho básico do git. Para realmente confirmar estas mudanças (isto é, fazer um commit), use:

* **git commit -m "comentários das alterações"**

Após o comando commit o arquivo é enviado para o HEAD, mas ainda não para o repositório remoto.

Para enviar estas alterações ao seu repositório remoto, execute o comando:

* **git push origin master**

**Conclusão**

Com a pesquisa realizada na elaboração desse trabalho, concluo que o Git é hoje a melhor ferramenta para controle de versões, pude notar que com essa poderosa ferramenta é possível controlar não somente versões de software, mais também versões de documentos de todo tipo. Podemos fazer várias alterações nos projetos tanto individual como em equipe, que o Git consegue controlar todas as alterações realizadas, tornando a redundância de informações quase que a nenhuma. É possível com alguns comandos simples desfazer alterações que foram feitas de maneira indesejadas.

Com esses benefícios propostos pelo Git é possível concluir que, a ferramenta tem sido ainda mais utilizada tanto por profissionais ligados à área de tecnologia quanto usuários de todos os campos que a necessidade do trabalho com projetos.

**REFERÊNCIA BIBLIOGRÁFICA**

**Acessado em 23-09-2016**

http://rogerdudler.github.io/git-guide/index.pt\_BR.html

<http://tableless.com.br/alguns-comandos-git/>

<https://git-scm.com/book/pt-br/v1/Primeiros-passos-No%C3%A7%C3%B5es-B%C3%A1sicas-de-Git>

https://git-scm.com/book/pt-br/v1/Primeiros-passos-Uma-Breve-Hist%C3%B3ria-do-Git