

(NOME DOS INTEGRANTES)

**CONTROLE DE VERSIONAMENTO**

NOME DOS INTEGRANTES

**CONTROLE DE VERSIONAMENTO**

Trabalho de fixação de conteúdo apresentado á Proz Educação, como requisito parcial para aprovação em módulo do curso de Técnico em Desenvolvimento de Sistemas

Professor: Juliano **(sobrenome)**

**Resumo**

O desenvolvimento de softwares é uma tarefa complexa, geralmente demorada, e comumente conta com o envolvimento de varias pessoas. Neste contexto, podemos citar a criação de novos softwares desde o zero e correções ou melhorias em softwares já desenvolvidos. A medida que um projeto evolui, é preciso ter o controle de tudo que foi feito dentro do código-fonte para que se possa regredir o código em caso de falhas, ou ter o controle da eficiência das mudanças feitas. Em caso de projetos que contam com colaboração múltipla, esse controle se faz necessário para que se possa rastrear quem mudou o quê, por quê e pra quê. Assim sendo, este trabalho busca explorar as possibilidades de controle de versionamento e seus impactos no desenvolvimento de softwares, através de diferentes ferramentas e metodologias, mas com foco na apresentação da ferramenta de maior popularidade para versionamento de software: O Git.

**PALAVRAS-CHAVE:** Versionamento; Github; Commit; SemVer; Repositório.

**Abstract**

Software development is a complex and often time-consuming task that frequently involves the collaboration of multiple individuals. In this context, we can consider both the creation of new software from scratch and the maintenance, bug fixes, or enhancements of existing software. As a project evolves, it becomes essential to maintain control over all changes made within the source code. This control allows for code rollback in case of failures and ensures an understanding of the effectiveness of the modifications. For projects with multiple collaborators, this control becomes crucial to track who changed what, why, and for what purpose. Consequently, this work aims to explore the possibilities of version control and its impact on software development. We will examine various tools and methodologies, with a particular focus on introducing the most popular version control tool: Git.  
  
**KEY-WORDS:** Version control; GitHub; Commit; SemVer; Repository

1. **Introdução**

O desenvolvimento de softwares dificilmente é uma tarefa solitaria. Projetos robustos e multifuncionais pedem a colaboração de equipes multidisciplinares com profissionais como desenvolvedores front-end, back-end, full-stack, mobile, desenvolvedores de banco de dados, Dev Ops, Ethical Hackers dentre outros. E comumente essas equipes precisam trabalhar simultaneamente dentro do mesmo código fonte, seja no mesmo bloco de programação, mesma função do código ou blocos e funções diferentes.

Além da necessidade de atuação simultanea em diferentes parte do código fonte, é necessário que haja um controle do que é feito dentro do código, principalmente em cenários em que o código já esteja estruturado e operacional, e precisa-se garantir que as alterações não impossibilitarão o software de funcionar corretamente. Em outras palavras, é como se precisassem de uma máquina do tempo para que pudessem alterar o código-fonte mas com uma medida de segurança para retornar o código ao seu estado original em caso de problemas com as alterações.

Existem ferramentas que funcionam como essas máquinas do tempo. Elas registram o código em sua forma original, aponta as alterações que foram feitas em relação ao estado anterior do código, indica quem as fez em caso de trabalho coletivo, e possibilita o desenvolvimento paralelo do mesmo bloco de código, permitindo a fusão dessas linhas paralelas de desenvolvimento posteriormente. E, se tudo der errado, basta voltar ao estado inicial do código. Desde a década de 90 existe esse tipo de ferramenta como CVS, ClearCase, Source-Safe e SVN.

Mas foi somente em meados de 2000 que surgiram os sistemas de controle de versão mais modernos como Mercurial, Bazaar e o que possui maior abrangência no mercado hoje: O Git.

1. **Sistemas de versionamento e a evolução das ferramentas de versionamento**

Apesar da história da programação se dar inicio na primeira geração de computadores em 1940 com o surgimento do ENIAC, que funcionava com válvulas e cartões perfurados e consistia numa programação de baixo nível com instriuções escritas manualmente, o surgimento dos sistemas de controle de versão aconteceu em meados de 1972. Nessa década surgiu o SCCS (Source Code Control System), desenvolvida na Bell Labs por Marc Rochkind, para computadores IBM System/370 com o sistema operacional OS/360. Entretanto o sistema tinha algumas limitações notáveis: O SCCS naõ permitia trabalhar com um projeto inteiro, de forma que tudo fosse trabalhado de forma individualizada, o que dificultava o controle de versão dado o grande volume de dados; Era um SCV (Sistema de controle de versionamento) centralizado; Mantinha apenas o estado atual do codigo fonte e registrava as alterações feitas, mas não permitia o resgate de versoes anteriores; A sintaxe das versões era complicada, dito que ainda não existia o SemVer (Semantic Versioning); Não havia como trabalhar com *branchs* (Ramificações no projeto original); Tinha capacidade de trabalho reduzida ao trabalhar com arquivos binários; Não possuia mecanismos de defesa contra manipulações diretas nos arquivos de controle de versão.

Cerca de 10 anos depois, em 1982, surge a primeira resposta direta ao SCCS: O RCS (Revision Control System), desenvolvido por Walter F. Tichy enquanto ele estava na Universidade de Purdue. O RCS ainda era um sistema centralizado, mas com uma interface mais fácil e com resposta de armazenamento e leitura mais eficientes por salvar as alterações localmente e só depois as sincronizar com açoes de *push* e *pull no servidor central*. Trouxe também a criação de *branchs* e a possibilidade de recuperação de versões anteriores dos códigos.   
  
Nos anos 90, surge o CVS (Concurrent Version System), que permitia a colaboração remota, superando as limitações do RCS. E no inicio dos anos 2000 surge o SVN (Apache Subversion), com capacidade de rastrear arquivos renomeados, agrupar mudanças em um único commit, e maior eficiencia na comunicação entre cliente e servidor, dentre outras vantagens diante do CVS. Mas até então, todos eram centralizados.

Somente em 2005 que tivemos o surgimento do GIT, que resolveu a maioria esmagadora das limitações dos concorrentes mais velhos, e trouxe novos conceitos que iriam revolucionar os sistemas de versionamento. O Git foi lançado em 2005 por Linus Torvalds e outros colaboradores Linux, como um software de código aberto, o que tornava o GIT um SCV muito flexível e dinâmico. O GIT apresentava velocidade de oepração, tornando ações como *commit*, *branch* e *merge* tarefas rápidas e simples; Possuia *commits* atomicos, o que possibilitava o rastreamento preciso dos *commits* e histórico completo de alterações, inclusive renomeio e movimentação de arquivos, facilitando as ações de *rollback* em casos de falhas; Prestava bom suporte a links simbólicos e outros tipos de arquivos, e o principal: Era o primeiro SCV distribuído. Ou seja: As alterações eram feitas localmente na máquina do desenvolvedor, podendo serem feitas até mesmo em ambiente offline, e posteriormente eram sincronizadas com o código fonte que estava hospedado em um serviço centralizado, o que possibilitava o trabalho simultaneo de desenvolvedores de qualquer lugar, em qualquer área do código. Desde então, o GIT é a ferramenta de versionamento mais versátil e mais usada até nos dias de hoje.

Mas em 2008 o GIT passou por uma integração que consolidou sua liderança no mercado. Chris Wanstrath, J. Hyett, Tom Preston-Werner e Scott Chacon lançaram o GitHub.

O GitHub é uma plataforma que hospeda os códigos utilizados pelo GIT para versionamento e oferece alguns recursos adicionais como controle de acesso, rastreamento de problemas, integração contínua e outras funcionalidades. Ele facilitou o emprego do SCV distribuído, permitindo que desenvolvedores colaborem em projetos de código aberto e criem *pull requests* como sugestões de melhoras ou correções em projetos públicos, trouxe wikis criando informação centralizada e acessível para todos, e hospeda gratuitamente sites estáticos, o que o torna numa ótima opção para portfólios e documentações.   
  
Apesar de existir concorrentes como GitLab, BitBucket, SourceForge, Gitea e Mercurial, o GitHub ainda é o mais utilizado para pequenos e grandes projetos e/ou empresas.

1. **Beneficios do versionamento**

Partindo da premissa que raramente o desenvolvimento de softwares é uma tarefa individual, o controle de versionamento se torna indispensável para garantir a segurança, a celeridade e eficiencia no desenvolvimento do projeto. Segundo Dias (2016), 4 perguntas devem ser feitas para identificar a necessidade de uso de um SCV. Se a resposta de 1 delas for “sim”, então um SCV deve ser implementado. São elas:

1. “Alguém já subscreveu o código de outra pessoa por acidente e acabou perdendo as alterações ?”
2. “Têm dificuldades em saber quais as alterações efetuadas em um programa, quando foram feitas e quem fez ?”
3. “Tem dificuldade em recuperar o código de uma versão anterior da que está em produção ?”
4. “Têm problemas em manter variações do sistema ao mesmo tempo ?”

Um SCV apropriado as necessidades do projeto e da equipe, podem impactar positivamente atraves de:

* **Histórico de alterações:** O SCVpermite criar um histórico com todas as alterações feitas dentro do código fonte. Desde a inclusão de linhas de comando, exclusão e modificações até a movimentação de seções de código, arquivos ou um simples renomeamento de algum elemento.
* **Controle de versão e segurança/confiabilidade:** Alguns SCV permitem retornar rapidamente o código a um estado anterior em caso de falhas, insatisfações com as mudanças, ou em caso de alterações inadvertidas/não autorizadas, garantindo confiabilidade ao código. No GitHub, há o uso do *Pull request,* onde uma alteração só é feita caso um moderador autorize a alteração no código fonte raiz.
* **Teste e experimentação:** Com versões controladas, podendo escolher qual versão atual ou passado deve ser executada, testes de novas funcionais, correções de bugs ou quaisquer outras demandas ficam mais fáceis e eficientes de serem executadas no projeto.
* **Ramificação e mesclagem** : Os SCV mais modernos permitem o uso de *branchs,* o que permite o desenvolvimento simultaneo em áreas diferentes do código, ou na mesma área, sem que isso impacte negativamente no processo, com uma gestão inteligente em caso de conflitos após a mescla das alterações, como alterações na mesma linha de código, por exemplo.
* **Colaboração eficiente:** Especialmente nos SCV com ambiente distribuído, a colaboração em equipe fica muito mais fácil, uma vez que o trabalho pode ser feito por diversas pessoas de varidas localidades, de forma online ou offline e com eficiência e tempo de resposta curto, uma vez que as mudanças são feitas inicialmente em cópias locais do código e somente após ações de *pull* são submetidas ao servidor para *merge.*

Mas, de acordo com Paracelso (Século XVI), “todas as substâncias são venenos. Não existe nada que não seja veneno. Somente a dose correta diferencia o veneno do remédio”. Então, um SCV inadequado as necessidades do projeto e da equipe, uma equipe sem entendimento comum da ferramenta, ou uma gestão inadequada do SCV pode ter efeito reverso, e dificultar muito o desenvolvimento do projeto.

**Controle de versão**

O controle de versão é um sistema que registra alterações em arquivos ao longo do tempo, permitindo revisitar versões específicas.

**Conceitos Básicos:**

* Repositório: Armazena todos os arquivos e suas versões.
* Commit: Registro de uma versão específica do projeto.
* Branch: Linha de desenvolvimento paralela.
* Merge: Junta mudanças de diferentes branches.
* Tag: Marca versões importantes do projeto.

**Sistemas de Controle de Versão:**

* Git: Sistema distribuído, popular pela eficiência.
* Subversion (SVN): Sistema centralizado.
* Mercurial: Sistema distribuído semelhante ao Git.

**Fluxo de Trabalho Comum em Git:**

1. Clonar repositório.
2. Criar branch.
3. Fazer alterações e commit.
4. Puxar alterações remotas.
5. Fundir branches.
6. Enviar alterações.

**Melhores Práticas:**

* Fazer commits frequentes.
* Usar branches de funcionalidades.
* Realizar revisões de código.
* Utilizar integração contínua.

**Resolução de Conflitos:**

* Editar manualmente arquivos conflitantes e fazer commit.

**Ferramentas Populares:**

* GitHub: Plataforma para hospedagem e revisão de código.
* GitLab: Similar ao GitHub com integração contínua.
* Bitbucket: Suporta Git e Mercurial.