**Faculdade de Tecnologia Dr. Thomaz Novelino**

**Disciplina Eletiva – Programação Web**

**Prof. Me. Fausto G. Cintra  
Relatório Acadêmico**

**Git**

**Iury Almeida de Oliveira**

**Socrates cheiragato**

**INTRODUÇÃO**

O método preferido de controle de versão por muitas pessoas é copiar arquivos em outro diretório (talvez um diretório com data e hora, se forem espertos). Esta abordagem é muito comum por ser tão simples, mas é também muito suscetível a erros. É fácil esquecer em qual diretório o usuário está e gravar acidentalmente no arquivo errado ou sobrescrever arquivos sem querer.

O controle de versão é um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo de forma que o usuário possa recuperar versões específicas.

1. O que é versionamento de código?

O controle de versão é um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo de forma que o usuário possa recuperar versões específicas.

Um designer gráfico ou um web designer e quer manter todas as versões de uma imagem ou layout, usar um Sistema de Controle de Versão (Version Control System ou VCS). Ele permite reverter arquivos para um estado anterior, reverter um projeto inteiro para um estado anterior, comparar mudanças feitas ao decorrer do tempo, ver quem foi o último a modificar algo que pode estar causando problemas, quem introduziu um bug e quando, e muito mais. Usar um VCS normalmente significa que se o usuário estragou algo ou perdeu arquivos, poderá facilmente reavê-los. Além disso, o usuário pode controlar tudo sem maiores esforços.

[Sistemas de Controle de Versão Locais](https://git-scm.com/book/pt-br/v1/Primeiros-passos-Sobre-Controle-de-Vers%C3%A3o#Sistemas-de-Controle-de-Vers%C3%A3o-Locais)

Um método muito utilizado para controle de versão por muitas pessoas é copiar arquivos em outro diretório, porém esta abordagem é muito comum por ser tão simples, mas é também muito suscetível a erros. O problema é esquecer em qual diretório o usuário está e gravar acidentalmente no arquivo errado ou sobrescrever arquivos sem querer.

****

Figura 1 Diagrama de controle de versão local.

[**Sistemas de Controle de Versão Centralizados**](https://git-scm.com/book/pt-br/v1/Primeiros-passos-Sobre-Controle-de-Vers%C3%A3o#Sistemas-de-Controle-de-Vers%C3%A3o-Centralizados)

Com a necessidade de trabalhar em conjunto com outros desenvolvedores, que usam outros sistemas, foram desenvolvidos Sistemas de Controle de Versão Centralizados (Centralized Version Control System ou CVCS). Esses sistemas, como por exemplo o CVS, Subversion e Perforce, possuem um único servidor central que contém todos os arquivos versionados e vários usuários que podem resgatar (check out) os arquivos do servidor. Por muitos anos, esse foi o modelo padrão para controle de versão.

****

Figura 2 Diagrama de Controle de Versão Centralizado.

Os usuários podem ter conhecimento razoável sobre o que os outros desenvolvedores estão fazendo no projeto. Administradores têm controle específico sobre quem faz o quê. Entretanto, esse arranjo também possui grandes desvantagens. O mais óbvio é que o servidor central é um ponto único de falha. Se o servidor ficar fora do ar por uma hora, ninguém pode trabalhar em conjunto ou salvar novas versões dos arquivos durante esse período. Se o disco do servidor do banco de dados for corrompido e não existir um backup adequado, perde-se tudo. VCSs locais também sofrem desse problema,sempre que se tem o histórico em um único local, corre-se o risco de perder tudo.

[**Sistemas de Controle de Versão Distribuídos**](https://git-scm.com/book/pt-br/v1/Primeiros-passos-Sobre-Controle-de-Vers%C3%A3o#Sistemas-de-Controle-de-Vers%C3%A3o-Distribu%C3%ADdos)

Os Sistemas de Controle de Versão Distribuídos (Distributed Version Control System ou DVCS). Em um DVCS (tais como Git, Mercurial, Bazaar ou Darcs), os usuários não apenas fazem cópias das últimas versões dos arquivos: eles são cópias completas do repositório. Assim, se um servidor falhar, qualquer um dos repositórios dos usuários pode ser copiado de volta para o servidor para restaurá-lo. Cada checkout (resgate) é na prática um backup completo de todos os dados.

****

Figura 3 Diagrama de Controle de Versão Distribuído.

Muitos desses sistemas trabalham bem com vários repositórios remotos com os quais eles podem colaborar, permitindo que o usuário trabalhe em conjunto com diferentes grupos de pessoas, de diversas maneiras, simultaneamente no mesmo projeto. Isso permite que o usuário estabeleça diferentes tipos de workflow (fluxo de trabalho) que não são possíveis em sistemas centralizados, como por exemplo o uso de modelos hierárquicos.

2.Quais as vantagens da utilização do versionamento de código?

Em termos de gestão, os DVCS contribuem para facilitar a missão das equipes de trabalho, de modo a que exerçam as suas tarefas da melhor forma, e vêm ajudar à redução de custos.  
Os principais benefícios deste tipo de opção estão relacionados com a distribuição do processamento, acarretando vantagens em termos de redundância e/ou replicação de repositórios e permitindo alargar as possibilidades de colaboração e parceria entre programadores.

Benefícios para os programadores

* Rapidez: como os processos são operados localmente, deixa de ser preciso contactar o servidor central para proceder a operações como um *commit*, *log* ou *diff*.
* Autonomia: permite trabalhar em modo desconectado, em qualquer local, só sendo necessária uma conexão com a rede para troca de revisões com outros repositórios.
* Ramos individuais: contrariamente aos chamados sistemas centralizados, combinar ramos não é uma obrigação para cada *commit* e antes uma decisão que depende do programador; o trabalho local faz-se num ramo individual que não tem interferência com os demais ramos, mesmo em processos de sincronização entre repositórios.
* Facilidade na Fusão: os DVCS usam o rastreamento automático, o que facilita de forma significativa o processo de fusão.

Benefícios para Gestão/Coordenação

* Confiabilidade: com um sistema centralizado, qualquer problema que surja no servidor vai parar todo o trabalho de desenvolvimento. Um sistema distribuído permite que a equipa continue a trabalhar e os repositórios dos programadores servem como cópias de *backup*, não havendo riscos de perder nada do projecto.
* Redução de custos com o servidor: o repositório "central" (se ele existir) funciona como repositório "oficial", ao invés de ter uma função de processador central dos pedidos. Assim, a carga de processamento é distribuída pelas máquinas dos programadores.

**3.O que é o Git?**

Iury

4. Qual o fluxo de trabalho e os principais comandos utilizados no git durante o

processo de versionamento de código, especialmente nas tarefas de:

4.1. inicialização de repositorio

Iury

4.2. Clonagem de um repositorio já existente

Iury

4.3. *Commit* de código da estação local para o repositório remoto

Você pode propor mudanças (adicioná-las ao Index) usando os seguintes comandos:

git add <arquivo>

git add \*

Este 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".

Agora o arquivo é enviado para o HEAD, mas ainda não para o repositório remoto.

4.4. Como atualizar um repositório local com o código mais recente do

repositório remoto

Para atualizar seu repositório local com a mais nova versão, basta executar o comando “git pull” na pasta de trabalho do usuário para obter e fazer o merge (mesclar) alterações remotas. Para fazer merge de um outro branch ao seu branch ativo (ex. master), use “git merge <branch>”.

Em ambos os casos o git tenta fazer o merge das alterações automaticamente. Infelizmente, isto nem sempre é possível e resulta em conflitos. Você é responsável por fazer o merge estes conflitos manualmente editando os arquivos exibidos pelo git. Depois de alterar, você precisa marcá-los como merged com “git add <arquivo>”.

Antes de fazer o merge das alterações, você pode também pré-visualizá-as usando “git diff <branch origem> <branch destino>”

5. O que é e como funcionam as tarefas de:

5.1. Fork de repositorio

iury

5.2. Criação de manutenção de Branches

iury

5.3. Pull request

Os pedidos de extração permitem que você conte aos outros sobre as alterações que você enviou para um repositório no GitHub. Uma vez que uma solicitação é aberta, você pode discutir e analisar as possíveis alterações com os colaboradores e adicionar confirmações de acompanhamento antes que as alterações sejam mescladas no repositório..

A branch master é a branch com o código final do projeto, estável. Criando uma nova branch, se você submeter o pull request para o repositório original, mas ele não for aceito, as alterações não estarão na sua branch máster.

REFERENCIAS BIBLIOGRÁFICAS

<https://git-scm.com/book/pt-br/v1/Primeiros-passos-Sobre-Controle-de-Vers%C3%A3o>

<https://github.com/MuvucaGames/Tutoriais/wiki/%5BWIP%5D-Tutorial-completo-sobre-versionamento-com-GIT>

<https://www.growunder.com/blog/dicas/65-git-como-funciona-e-quais-as-vantagens-e-desvantagens>

<https://help.github.com/articles/about-pull-requests/>