

## Aula 3: Controle de Versão

### Apresentação

---

Nesta aula será dado enfoque em um tópico basilar da Gerência de Configuração, no caso o do Controle de Versão. Serão explicados os fundamentos do tópico, bem como alguns tipos de versões básicas a serem adotados para que se tenha um Controle de Versão minimamente funcional. Serão explicadas algumas boas práticas gerais específicas para o Controle de Versão, e será dada uma visão geral sobre duas ferramentas consagradas de Controle de Versão, no caso Subversion e Git.

### Objetivos

---

- Conhecer os fundamentos do Controle de Versão;
- Identificar tipos de Controle de Versão;
- Conhecer boas práticas gerais do Controle de Versão;
- Identificar e conhecer a ferramenta Subversion;
- Identificar e conhecer a ferramenta Git.

## O Controle de Versão e seus Fundamentos

---

Vimos anteriormente que várias coisas se classificam como Itens de Configuração, e que nosso foco se dará sobre aquelas compostas por software (código-fonte, módulos, sistemas inteiros etc.) e documentação em geral. Dado o contexto de uso nas organizações em que estes itens de configuração se encontram inseridos, além da relativa facilidade com que podem ser alterados, é natural esperar que ocorram muitas mudanças em suas estruturas, conteúdos e informações de configuração ao longo dos seus ciclos-de-vida.

O Controle de Versão basicamente se focará em uma coisa: registrar e rastrear as mudanças que ocorrem nestes itens de configuração. Estas são capacidades fundamentais em um projeto de software, ou mesmo em empreendimentos considerados mais operacionais, tais como manutenção contínua de software, pois traz uma série de benefícios, entre os quais:

## Múltiplos membros trabalhando simultaneamente em um mesmo projeto



O Controle de Versão é fundamental para a colaboração ordeira e organizada entre membros de um mesmo projeto ou empreendimento operacional, pois permite que cada membro edite sua própria cópia de itens de configuração, compartilhando estas cópias com o restante do time quando necessário ou quando certo de que as cópias são estáveis o bastante;

## Permite a integração do trabalho que foi realizado simultaneamente, mas separadamente



O Controle de Versão permite a junção do trabalho realizado por membros diferentes em uma única versão unificada. Permite ainda a visibilidade sobre conflitos de edição, ou seja, os casos em que membros diferentes fazem alterações diferentes em um mesmo trecho de código-fonte por exemplo. Isto gera subsídio para a tomada de ação no sentido de eliminar o conflito. Embora este ponto seja mais fácil de ser atingido no controle de versão distribuído, também é possível de ser atingido no controle de versão centralizado;

## Permite manter o histórico dos itens de configuração




Provavelmente o benefício mais tangível do controle de versão, é uma espécie de seguro contra alterações problemáticas, corrupção não esperada do conteúdo de um item de configuração, perda de dados, etc. O controle de versão permite o levantamento do histórico dos itens de configuração, além de possibilitar a remediação de alterações, reabilitando uma versão que se sabe ser estável. Também importante é a capacidade de saber quem, quando e onde realizou a alteração que causou problemas, permitindo a tomada de ação corretiva e preventiva.

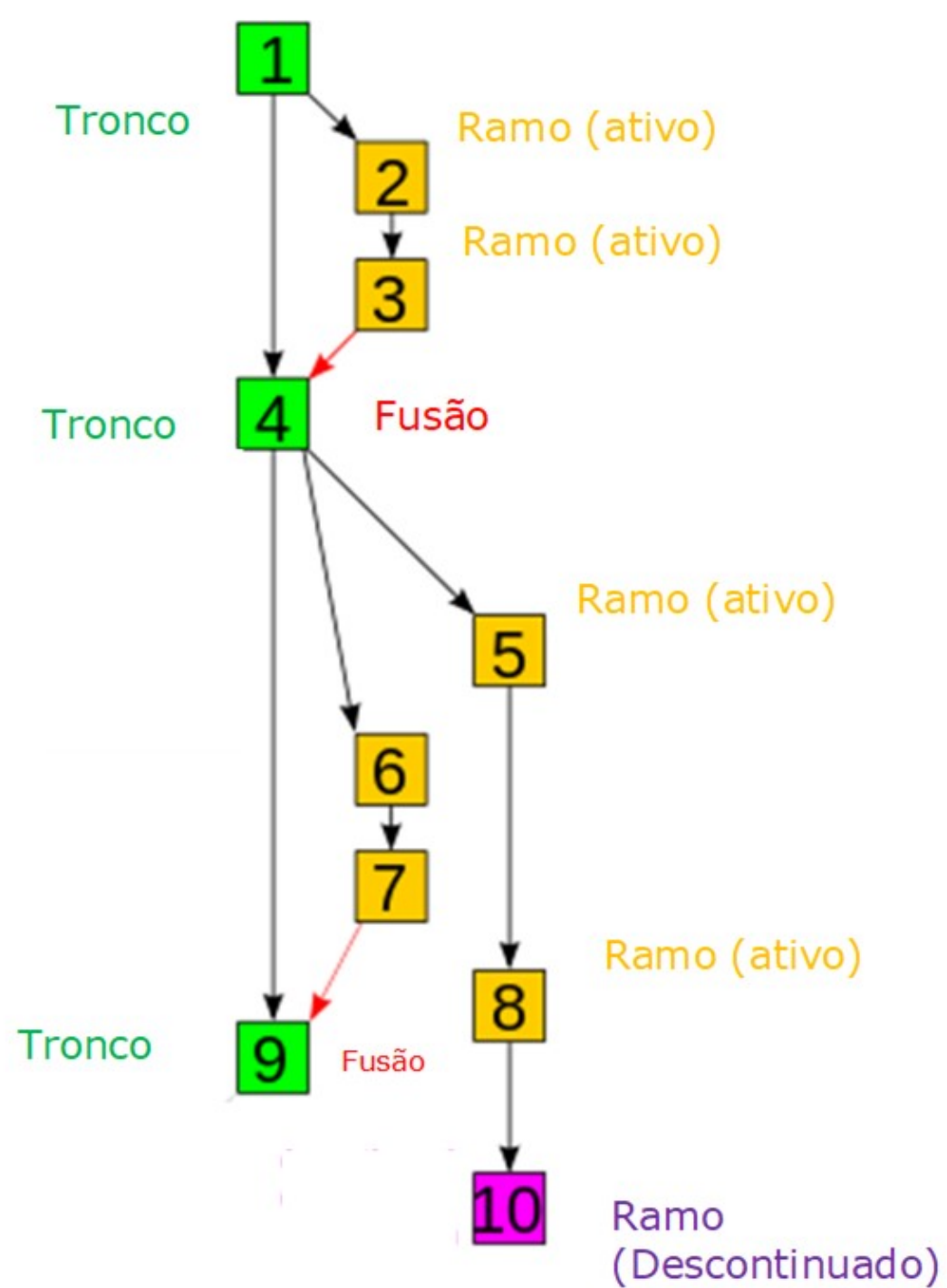
Tais capacidades normalmente serão fornecidas por algum sistema de controle de versão, que por sua vez fornecerá dois elementos chave: o *front-end*, ou seja a interface com a qual os usuários irão interagir diretamente; e um *back-end*, representado por alguma forma banco de dados contendo todas as edições e versões históricas dos itens de configuração em questão. A estes bancos de dados dá-se o nome de “**repositório**”.

Temos outros conceitos fundamentais habilitadores dos sistemas de controle de versão. Veja a seguir.

 Conceitos habilitadores dos sistemas de controle de versão

 Clique no botão acima.

- **Cópia de trabalho (working copy, sandbox ou checkout):** itens de configuração que foram selecionados por um usuário e que agora se encontram bloqueados para edição para aquele usuário, enquanto são realizadas as alterações necessárias. Estas são as cópias locais em que o usuário estará trabalhando.
- **Confirmação (Commit):** também conhecido como “realizar um commit”, é uma ação de confirmação, sendo uma ação deliberada por parte de um usuário para as mudanças em um item de configuração que ele tenha realizado, enviando-as ao servidor, tornando-as oficiais e disponíveis a outros usuários.
- **Grupo de mudanças (changeset):** é um novo registro gerado automaticamente no repositório para cada commit realizado. Cada changeset ou grupo de mudanças pode conter inúmeras modificações em um ou mais itens de configuração, e seu tamanho e extensão depende de quão frequentemente os usuários escolhem realizar commits. Por exemplo, um usuário pode escolher realizar commits para cada alteração significativa, o que implica em vários commits em um curto espaço de tempo, como minutos ou horas, ou pode escolher realizar commits para períodos maiores de tempo, incluindo dias ou até semanas.
- **Revisão (revision):** Número identificador único dado a cada changeset. A ideia dos revisions é facilitar a rastreabilidade das mudanças.
- **Etiqueta (tag ou label):** Uma revisão que foi congelada, normalmente se tornando “somente-leitura” e que muito provavelmente é candidata a ser liberada para produção.
- **Atualização (Update):** também conhecido como “realizar uma atualização”, ocorre quando usuários deliberadamente realizam o download de um changeset do repositório, de forma a possuírem a última versão do código fonte ou documentação.
- **Diferenciação (diffing):** é um mecanismo de visualização das diferenças entre revisões. Facilitando, portanto, a comparação entre versões de um mesmo item de configuração, bem como proporciona conhecimento sobre quem e quando realizou a alteração em questão. A comparação pode ser feita linha a linha para as versões dos itens de configuração, para revisões sequenciais e até mesmo entre duas revisões em qualquer ponto no histórico dos itens de configuração.
- **Ramificação (branching):** serve para criação de cópias alternativas ou ramos (branches) do código fonte / documentação a ser alterados, permitindo a execução das alterações sem modificação direta do tronco ou código / documentação mestre (master). Branching é importante pois dá subsídio à realização de testes e da consequente estabilização das versões, levando à redução dos riscos resultantes de alterações não-intencionais ou mesmo de alterações planejadas.
- **Fusão (merging):** serve para unificar ou fundir trechos de alterações ou mesmo alterações completas de autoria de usuários diferentes, em uma única nova versão, a ser combinada com a versão mestre disponível no repositório. Como se pode imaginar, é também o mecanismo usado para reunificar o código de um ou mais ramos ao código mestre, uma vez que exista confiança que estes ramos são estáveis e que não irão causar problemas.
- **Conflitos de versão:** ocorrem quando dois ou mais usuários alteram o mesmo trecho ou linha do mesmo pedaço de código-fonte ou documentação, seja por engano, seja quando criam ramos diferentes para isso. Normalmente requerem fusão para eliminação do conflito, no entanto é possível que a ferramenta de controle de versão não consiga sozinha determinar a versão válida, requerendo portanto escolha e intervenção manual.



**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

## Diferentes Tipos de Controle de Versão

Repositórios são organizados em pastas (ou diretórios) e arquivos. Conforme usuários vão realizando mudanças nas estruturas destas pastas e / ou arquivos (tais como adição e movimentação de arquivos, deleção de uma pasta, alteração no nome de um arquivo ou qualquer outra coisa que afete o estado daquele objeto) é tarefa do controle de versão manter a rastreabilidade dessas mudanças.

Com o intuito de cumprir este objetivo, os Sistemas de Controle de Versão se organizam de duas maneiras distintas:

### Controle de Versão Centralizado

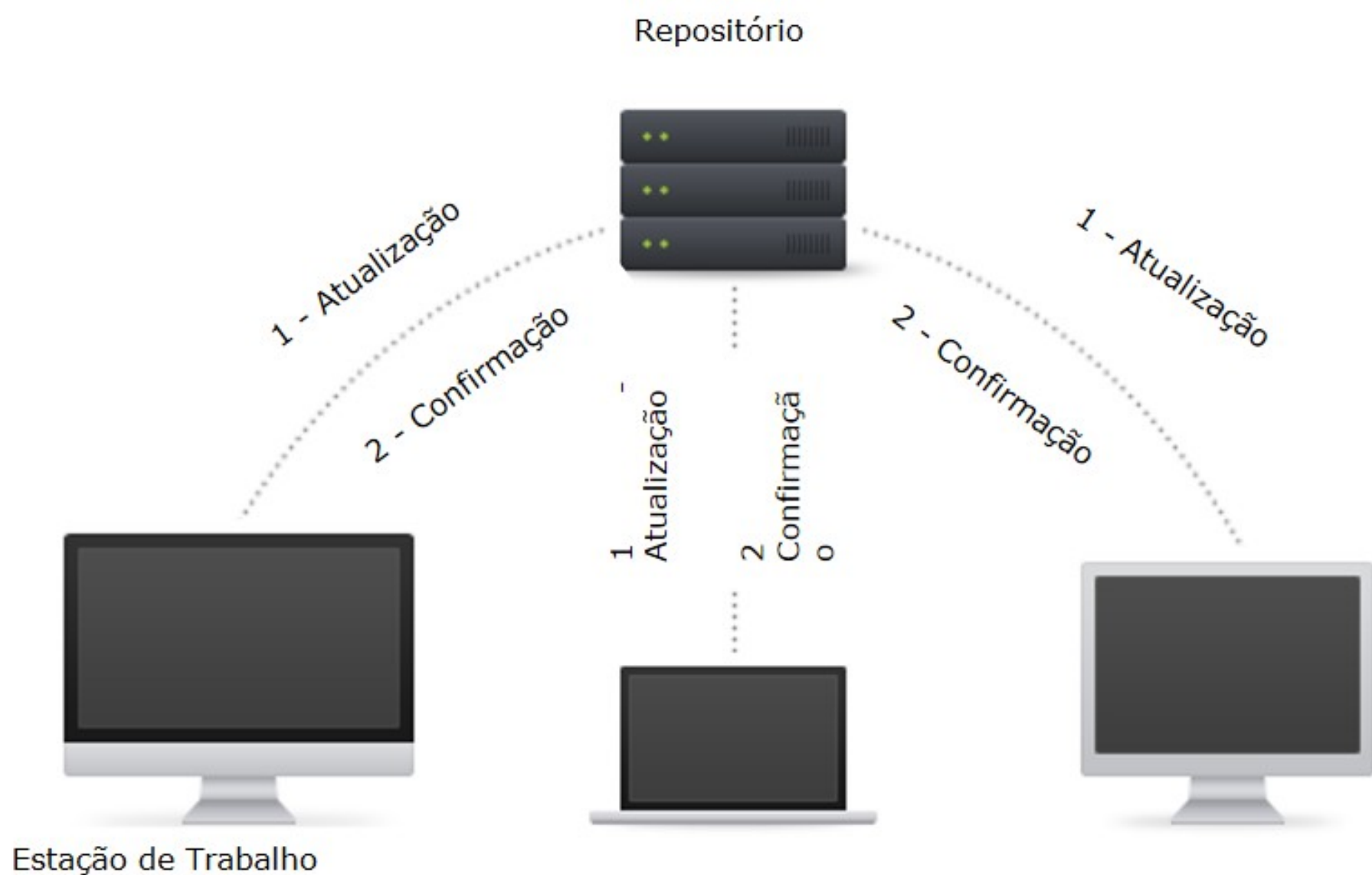
## Passo 1

O sistema de controle de versão irá esperar que o usuário autor das mudanças as envie as mudanças realizadas para “registro oficial”. Em outras palavras, o sistema de controle de versão requer o envio deliberado do changeset pelo usuário-autor, via ações de *confirmação* ou *commit*.

## Passo 2

Ocorre quando demais usuários precisam realizar uma *atualização* ou *update*, ou seja, precisam deliberadamente realizar o download do changeset, de forma a possuírem a última versão do código fonte ou documentação.

A centralização do controle de versão tem a vantagem de que torna relativamente fácil saber quais partes do código-fonte e documentação encontram-se *bloqueadas para edição* (**check-out** ou **checked-out**) pelos usuários. Estas são as cópias locais que estarão editando. A ferramenta Subversion é um exemplo bem conhecido de implementação do paradigma do controle de versão centralizado.



📷 Controle de Versão Centralizado. Fonte: próprio autor



📷 Ordem das Ações no Controle de Versão Centralizado. Fonte: próprio autor

A centralização do controle de versão tem a vantagem de que torna relativamente fácil saber quais partes do código-fonte e documentação encontram-se *bloqueadas para edição* (**check-out** ou **checked-out**) pelos usuários. Estas são as cópias locais que estarão editando. A ferramenta *Subversion* é um exemplo bem conhecido de implementação do paradigma do controle de versão centralizado.

## Controle de Versão Distribuído

Difere da abordagem centralizada já que nesta modalidade, além de um repositório centralizado, cada usuário possuirá sua própria cópia local deste repositório inteiro, formando assim uma espécie de rede de repositórios interconectados. A realização de *commits* não irá enviar as alterações a outros repositórios e torná-las disponíveis aos demais usuários, pois aqui o *commit* se limita a confirmar no repositório local as alterações realizadas pelo autor nos itens de configuração que estavam *checked-out*.

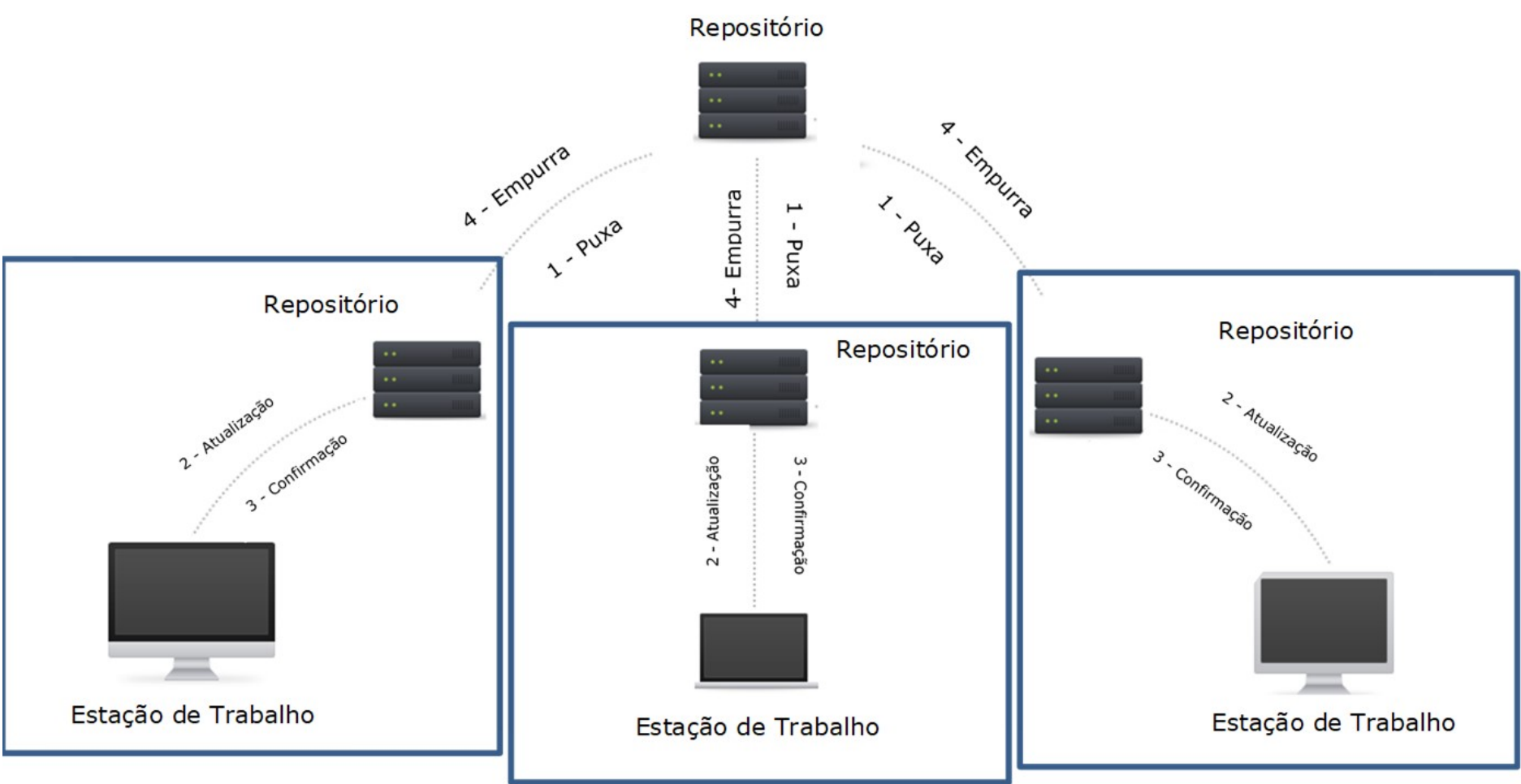
Para que os demais repositórios sejam sincronizados, além das já conhecidas ações de confirmação e atualização, são necessárias ainda mais duas ações adicionais, totalizando quatro ações:

- **Ação empurra (push):** o autor das alterações, após commit em seu repositório local, empurra as alterações deste repositório local para o repositório centralizado, disponibilizando-a para os demais usuários.
- **Ação puxa (pull):** os usuários puxam ou realizam o download das atualizações agora disponíveis no repositório central para seus próprios repositórios locais. Somente aí a ação de atualização será executada, fazendo com que os seus checkouts, ou as cópias dos itens de configuração em que estão efetivamente trabalhando reflitam as últimas alterações disponíveis.

Fica claro portanto que no controle de versão distribuído, as operações de confirmação e atualização apenas movem mudanças entre os checkouts e o repositório local, sem afetar nenhum outro repositório. Já as operações do tipo puxa e empurra movem mudanças entre o repositório local e o repositório central, sem afetar os checkouts.

Da mesma forma que no controle centralizado de versão, no controle de versão distribuído também existe potencial para conflitos de versão, ou seja, situações em que dois ou mais usuários realizem alterações em uma mesma linha de código ou documentação antes de.

A ferramenta Git é um bom exemplo de implementação do paradigma distribuído.







 Ordem das Ações no Controle de Versão Distribuído. Fonte: próprio autor

## Boas Práticas no Controle de Versão

---

Dada o emprego muito específico do Controle de Versão, boas práticas existem e são recomendadas que sejam adotadas. Cobriremos aqui algumas práticas gerais, bem como práticas específicas, tanto para o Controle de Versão Centralizado quanto para o Controle de Versão Distribuído. Fazem parte destas práticas:

## Usar mensagens de confirmação que sejam o mais descritivas possível



A cada commit, é importante que seja redigida uma boa mensagem que encapsule bem o significado daquela confirmação. Isto não só é útil ao próprio autor da mudança, já que conseguirá mais facilmente recordar e rastrear a evolução do seu próprio trabalho, mas também irá ser útil a outros usuários que estão a examinar a mudança, já que a descrição indica o propósito da mudança;

## Trate cada confirmação de forma atômica



É interessante que cada commit tenha um propósito claro e que o implemente de forma completa, sem dependências, construindo portanto **uma unidade lógica**. Isto facilita a localização de mudanças relacionadas com alguma funcionalidade específica ou correção de defeito, pois as disponibiliza em um local único. A utilidade do controle de versão é diluída se um commit atende a múltiplos propósitos ou se código ou documentação para um propósito em particular se encontram espalhados ao redor de vários commits. Obviamente pode ser impraticável em cem por cento do tempo realizar commits atômicos, mas na medida do possível é importante procurar fazê-lo, pois esta organização renderá dividendos no futuro ;

## Evite realizar confirmações de maneira indiscriminada



É interessante que a cada commit, arquivos específicos sejam fornecidos para o commit, evitando-se realizar commits gerais, que irão confirmar todo e qualquer arquivo que tenha sido alterado. Evitar commits completos desnecessários é importante pois evita-se confirmar arquivos que tenham sido alterados de maneira temporária ou que ainda não estejam completos e maduros para o commit;

## Incorpore mudanças de terceiros frequentemente



Procure trabalhar com versões de arquivos atualizados o máximo possível. Isto evita situações custosas, como por exemplo aquelas em que alguém trabalhou anteriormente na edição do mesmo trecho do item de configuração em que você está prestes a trabalhar mas que você não atualizou antes de começar tais edições. O resultado disto será muito provavelmente um conflito de versão que exigirá intervenção manual para ser resolvido;

## Compartilhe suas próprias mudanças frequentemente



Uma vez que você tenha gerado alterações constituintes de uma unidade lógica, procure compartilhá-las de forma imediata. A razão para isso é basicamente a mesma do ponto anterior, já que alguém poderá trabalhar no mesmo trecho em paralelo, exigindo conciliação posterior do item de configuração;

## Coordene com seus pares



Procure estar informado bem como informar seus pares a respeito dos itens de configuração em alteração. Novamente, a ideia é evitar conflitos de versão, logo é interessante que se coordene o início e o fim de alterações em cada item de configuração, ou ao menos o mais importantes;



## Evite confirmar arquivos gerados ou não-editáveis



Arquivos como PDFs, e arquivos resultantes de compilação não são destinados à edição por pessoas. Assim, pode-se argumentar que não precisam ser inseridos em um repositório de configuração. Embora isto possa ser um ponto polêmico, é importante reconhecer que os sistemas de controle de versão normalmente dispõem de capacidades para ignorar dados tipos de arquivo. Estas funcionalidades existem por um motivo: evitar situações como conflitos, bancos de dados sobrecarregados com arquivos desnecessários e que reduzem a performance do sistema etc. É interessante tecer considerações a este respeito, de forma a tomar uma decisão consciente e que faça sentido quanto à destinação deste tipo de arquivo.

## Invista no conhecimento da funcionalidade de fusão



Conflitos são inevitáveis, e por melhor que seja o controle de versão, eles irão ocorrer em algum momento. Boa parte da resolução de conflitos recairá de alguma forma sobre o mecanismo de fusão (merge) disponível. Hoje em dia, é comum inclusive que o sistemas de controle de versão aceitem ferramentas de fusão de forma “plug and play”, ou seja, você escolhe a que melhor atende suas necessidades. Assim sendo, teste e entenda bem o funcionamento destas ferramentas, de forma a escolher uma que lhe dê confiança e conforto no trabalho.

## Habilite notificação por e-mail



Esta é uma forma de automação simples mas poderosa, pois toda vez que algum usuário empurrar ou confirmar alterações, todas as partes interessadas serão notificadas. Isto proporciona ganho de eficiência à medida que a consciência situacional também aumenta.

# Identificar e conhecer a ferramenta Subversion

**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

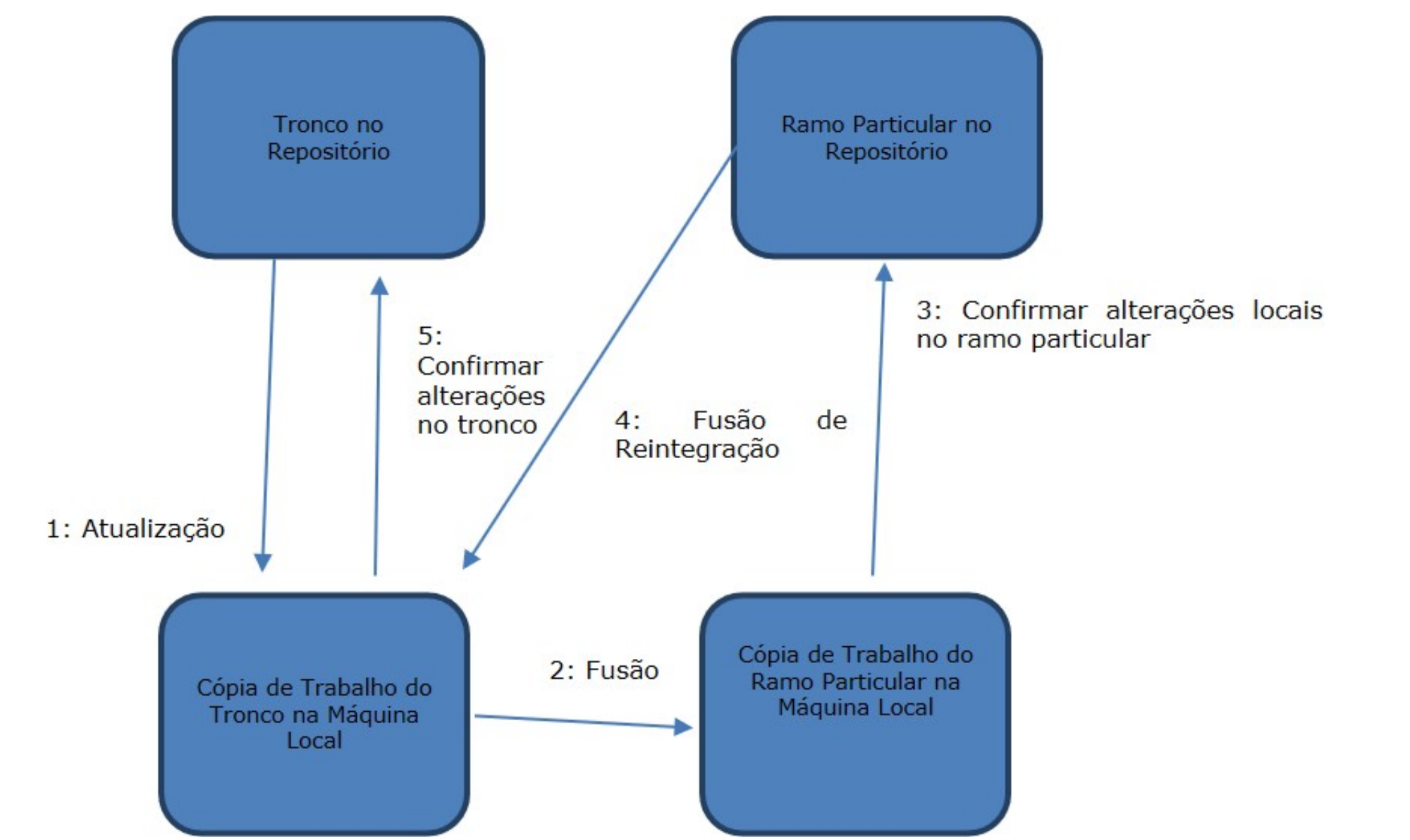
A ferramenta Subversion, mais popularmente conhecida simplesmente como SVN é uma ferramenta de controle de versão com longo histórico de uso pela comunidade de Gestão de Configuração, remetendo a meados da década de 2000. Por ser uma ferramenta com amplo histórico, implementa suas funcionalidades de maneira considerada tradicionalista, ou seja, não necessariamente representa o estado da arte, sendo hoje considerada praticamente como uma ferramenta de backup para diretórios, documentos e outros arquivos.

A estrutura do repositório do Subversion tem no tronco o principal ramo ou linha de desenvolvimento, ou seja, ele é o ponto central ou alvo de onde todos os demais ramos ou linhas paralelas de desenvolvimento (também chamadas de ramos particulares) surgem e eventualmente se fundem novamente. O modelo de ramificação do SVN aliás representa uma das maiores reclamações dos usuários, pois toda vez que uma parte do repositório central é ramificada e checked-out, ela se torna parte do diretório local no computador do usuário que realizou o procedimento de check-out, bem como se transforma em diretórios no servidor.

Este modelo tende a causar conflitos conhecidos como *conflitos de árvore* (tree conflicts). Estes conflitos ocorrem quando mudanças são realizadas nesta estrutura de diretórios, ocorrendo, portanto, frequentemente. Outro ponto que atrai reclamações de usuários atuais é o fato de o SVN ser centralizado, o que se traduz na necessidade de estar conectado ao servidor para realizar operações de commit, o que na prática, faz com que o SVN se torne inútil caso essa conexão seja perdida.

Ainda pelo fato de ser uma ferramenta centralizada, o SVN, além de precisar ser disponibilizado no servidor, precisa também ser disponibilizado nos computadores clientes. O aplicativo SVN mais comum para o papel de cliente é o TortoiseSVN, mas outros existem. O SVN encontra-se disponível para Windows e Linux. No caso do Windows, a instalação é especialmente fácil, pois várias versões que dispõem de interface gráfica existem, tal como a CollabNet. No caso do Linux, a instalação e operação geralmente requerem extenso uso de linha de comando via Terminal.

Do ponto de vista moderno, o uso do SVN ainda se justifica pelo baixo custo, pois a ferramenta em si é open source, sendo que as organizações precisam apenas se preocupar é investir em capacitação, infraestrutura física para hospedagem dos servidores envolvidos. Outro ponto é que muitas organizações que um dia estiveram na vanguarda da adoção da Gerência de Configuração incorporaram o SVN como ferramenta de controle de versão, sendo que agora, além da ampla base de usuários possuem uma grande quantidade de itens de configuração armazenados pela ferramenta, exigindo grande esforço de migração.



Fluxo Operacional da Ferramenta SVN. Fonte: SVN /Subversion Crash-Course & Quick Reference

# Identificar e conhecer a ferramenta Git

---

Git é uma ferramenta de controle de versão distribuída que pensa nos dados sob seu controle de maneira diferente da maioria das demais ferramentas de mesmo propósito. Enquanto ferramentas como o Subversion por exemplo considera estes dados como um grupo de arquivos e pastas e controla as mudanças feitos em cada um deles ao longo do tempo, o Git na verdade pensa nos dados como fotografias (snapshots) de um sistema de arquivos em miniatura.

Já que o Git é distribuído, existe pouca ou nenhuma latência de rede, já que cópias dos arquivos com todo os seus históricos correspondentes se encontram em repositórios locais nos discos rígidos dos computadores dos usuários. Isto facilita ações, tais como operações de diferenciação entre arquivos, já que não é necessário acessar servidores remotos, fazendo assim com que estas operações pareçam ser instantâneas. Outra vantagem é a possibilidade de trabalhar boa parte do tempo de maneira offline ou desconectada.

O git implementa três estados em que um arquivo pode se encontrar:

## Confirmado (committed)

Significa que o dado se encontra armazenado e a salvo no repositório ou banco de dados local.

## Modificado

Significa que o arquivo foi alterado por você, mas ainda não foi confirmado em seu banco de dados local e nem está “marcado” para ser confirmado.

## Preparado (staged)

Significa que o arquivo foi modificado e que foi marcado ou preparado para ser confirmado na próxima fotografia.

**O Git na verdade implementa três repositórios. O principal é o Repositório Git, que é o repositório central, onde todos os metadados e o banco de dados de objetos estão armazenados. O repositório de trabalho representa um checkout único de uma versão do projeto, e é um clone do repositório Git, além de ser armazenado localmente no disco rígido dos usuários para eles possam fazer alterações e modificações como queiram.**

Assim sendo, todas as vezes em que um commit é realizado, basicamente uma foto de como todos os arquivos no momento da fotografia é obtida e então uma referência é atrelada aquela fotografia. Para manter a eficiência, se durante um novo commit apenas alguns arquivos tiverem mudado, então o Git não armazena novamente os arquivos intactos, e ao invés disso apenas estabelece um link para os arquivos anteriores idênticos que já se encontravam armazenados.

O Git trata os dados portanto como uma corrente de fotografias, e toda vez que um commit é realizado, na verdade o Git armazena um objeto que contém um ponteiro ou referência para uma fotografia, nome e e-mail do autor, a mensagem de commit, além de ponteiros para o commit ou commits que antecederam o commit atual (os pais). Esta sistemática também se reflete nas ramificações, pois cada ramo passa a ser somente um ponteiro para um commit, tornando toda a operação muito leve. A ação de fusão (merge) também é facilitada, pois ao invés de requerer que o usuário determine manualmente qual é a melhor base a ser fundida, no Git esta escolha é determinada pela própria ferramenta por meio de uma ação denominada “merge commit”.

O índice (index) ou área de preparação (staging area) é normalmente representado por um arquivo que armazena informações sobre o que irá ser “oficializado” no próximo commit, ou seja, representa o que foi alterado e escolhido para ser “subido” do repositório de trabalho para o repositório git. Esta área pode ser omitida, sendo que é possível marcar todo arquivo alterado automaticamente como “staged”, eliminando-se assim a distinção prática e explícita entre os estados modified e committed, fazendo com que todo arquivo alterado seja marcado para ser confirmado.



 Fotografias dos dados armazenados pelo Git ao longo do tempo. Fonte: The Ultimate Guide for Beginners To Learn Git

O Git pode ser inteiramente usado via linha de comando ou via uma das várias interfaces gráficas compatíveis disponíveis, embora esta segunda opção tenha a desvantagem de que cada interface gráfica normalmente só implemente um grupo reduzido de funcionalidades disponíveis via linha de comando. O Git encontra-se disponível para Linux e Mac (terminal) e para Windows (prompt de comando e powershell).

**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

Nesta aula você aprendeu sobre Controle de Versão, seus conceitos fundamentais e boas práticas. Embora seja uma atividade de configuração simples, tem uma importância central ao tema de Gestão de Configuração, pois fornece a estrutura elementar para habilitação da rastreabilidade e segurança a que a Gestão de Configuração se propõe a oferecer. Vimos ainda uma introdução básica a duas ferramentas muito conhecidas que implementam o controle de versão, no caso as ferramentas Subversion e Github.

Agora é com você! Responda algumas questões para maior fixação do conteúdo.

# Atividade

---

Git é uma ferramenta:

- a) De Controle de Mudanças.
  - b) De Controle de Versão.
  - c) De Controle de Liberação.
  - d) De Controle de Configuração.
  - e) De Controle de Construção.
- 

Sandbox é outro nome dado a:

- a) Configuração.
  - b) Versão.
  - c) Checkout.
  - d) Ramo.
  - e) Tronco.
- 

Commits basicamente são:

- a) O download de um item de configuração alterado por um usuário.
  - b) A fusão de alterações paralelas a um item de configuração.
  - c) O descarte de alterações em um item de configuração.
  - d) A criação de um ramo no repositório.
  - e) A confirmação de alterações em um item de configuração.
- 

Não é uma boa prática de Controle de Versão:

- a) Manter o repositório central atualizado.
  - b) Manter o repositório local atualizado.
  - c) Inserir arquivos não-editáveis nos repositórios.
  - d) Fornecer descrições para commits.
  - e) Tratar commits de forma atômica.
-

O controle de versão distribuído é realizado com:

- a) Quatro ações.
- b) Uma atividade.
- c) Três ações e uma atividade.
- d) Duas atividades.
- e) Quatro atividades.

## Notas

### Título modal <sup>1</sup>

Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos.

### Título modal <sup>1</sup>

Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos.

## Referências

HAAS, J. **Configuration Management Principles and Practice**. 1 ed. Addison Wesley, 2003.

BOURQUE, P.; FAIRLEY, R. **Software Engineering Body of Knowledge (SWEBOK)**. 3 ed. IEEE Computer Society, 2017.

AIELLO, B. **Configuration Management Best Practices**. 1 ed. Pearson, 2013.

GARNER, J. **GIT: The Ultimate Guide for Beginners: Learn Git Version Control**. 1 ed. Independently Published, 2020

BLISS, N. **SVN / Subversion Version Control Crash-Course & Quick Reference**. 1 ed. CreateSpace Independent Publishing Platform, 2013

## Próxima aula

- Conceituação e tipos de mudança;
- Boas práticas no Controle de Mudanças;



- Software / Ferramentas de Controle de Versão.
- Redmine
- Gitlab

## Explore mais

---

- Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto. Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.