Aula 3º

Gerenciamento de versões

Olá, alunos(as),

Bem-vindos(as) a nossa terceira aula da disciplina "Gerência de Configuração". Nesta aula, vamos prosseguir pela nossa incursão pelas atividades da Gerência de Configuração. Veremos como funciona o Gerenciamento de Versões, que cuida da forma como a gestão das versões ocorre em nosso software.

Assim, não se esqueça de ler esta aula, assistir os nossos vídeos e fazer as atividades. Se tiverem alguma dúvida, vocês podem usar o nosso quadro de avisos, que eu responderei.

Bons estudos!

Objetivos de aprendizagem

Ao término desta aula, vocês serão capazes de:

- saber a importância da adoção do Gerenciamento de Versões;
- conhecer as principais características de um software de Gerência de Versões.

Seções de estudo

1 – Gerenciamento de versões

1 - Gerenciamento de versões

Vamos começar o nosso estudo a respeito do Gerenciamento de Versões propondo uma reflexão a você: você já trabalhou em um projeto de *software* que envolveu a entrega de mais de uma versão? Se sim, você prestigiou alguma vez um projeto que envolveu o controle de versões do primeiro cenário da Aula 01? Para recapitularmos, o cenário descrevia um gerenciamento manual das versões do sistema.

Talvez você não tenha visto o primeiro cenário, mas sim o terceiro cenário, em que uma pessoa sobrescreveu um arquivo, causando um bug no sistema que será trabalhoso de consertar, visto que a versão anterior do arquivo não existe mais?

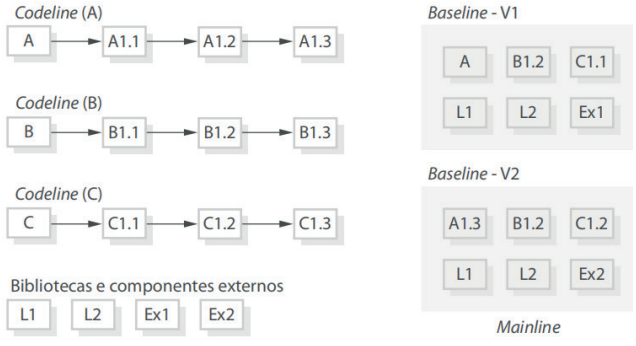
E se te contarmos a você que essas das situações que descrevemos na Aula 01 podem ser resolvidas se usarmos um sistema de gerenciamento de versões? Vamos explicar o porquê ao final dessa seção.

Primeiramente, vamos recapitular alguns conceitos que são importantes para o Gerenciamento de Versões: baseline, codeline e mainline. Em primeiro lugar, vamos retomar o conceito de uma **baseline**:

> Uma baseline é uma coleção de versões de componentes que compõem um sistema. As baselines são controladas, o que significa que as versões dos componentes que constituem o sistema não podem ser alteradas. Isso significa que deveria sempre ser possível recriar uma baseline a partir de seus componentes (SOMMERVILLE, 2011, p.477).

Já o **codeline** não é uma coleção, mas sim "um conjunto de versões de um componente de software e outros itens de configuração a qual esse componente depende", enquanto uma **mainline** se refere a uma sequência de baselines, ou seja, um conjunto de diferentes versões de um sistema. (SOMMERVILLE, 2011, p.477)

Figura 6 - Apresentação visual dos conceitos de codeline, baseline e mainline.



Fonte: SOMMERVILLE, 2011, p. 482.

No cenário 1 da Aula 01, gerenciamos as nossas codelines de maneira manual. Nesse caso, temos vários problemas que podem ocorrer, sendo que alguns deles foram demonstrados na aula 01: Falta de sincronização entre os programadores, ausência de histórico de mudanças, entre outros problemas.

Para resolvermos isso, é necessário adotarmos ferramentas e processos para facilitar o gerenciamento de versões. Assim, temos a **atividade de gerenciamento de versões**, que é definido por Sommerville (2011, p. 481) como o **"processo de acompanhamento de diferentes versões de componentes de software ou itens de configuração e os sistemas em que os componentes são usados"**.

Ainda segundo Sommerville (2011, p. 481), o processo de gerenciamento de versões envolve a garantia de que as mudanças de um desenvolvedor não interfiram nas outras. Assim, o autor afirma que o gerenciamento de versões pode ser pensado "como o processo de gerenciamento de *codelines* e *baselines*".

Essa atividade da Gerência de Configuração é representada pelas ferramentas de Controle de Versão, que permitem diversas possibilidades para os programadores em um projeto de software. Sommerville (2011, p. 482) afirma que essas ferramentas "identificam, armazenam e controlam o acesso a diferentes versões de componentes". Existem diferentes tipos de ferramentas de controle de versão, que veremos a seguir.

As ferramentas de controle de versão, apesar de parecer um assunto emergente na Informática, já eram discutidas na década de 1980, com as ferramentas RCS e SCCS, que eram bastante limitadas em relação às ferramentas utilizadas atualmente, permitindo o gerenciamento de um arquivo por vez e proibindo a concorrência (ou seja, um programador pode editar um arquivo por vez) (SINK, 2011).

Em seguida, surgiram novas ferramentas na década de 1990, que passaram a permitir o trabalho distribuído, porém, algumas dessas tarefas eram tediosas e trabalhosas. Essa geração foi representada pelos gerenciadores SVN e CVS, que dominaram o mercado por anos.

Os problemas que os desenvolvedores enfrentaram nesses gerenciadores de versão foram usados como subsídios e motivação para criar uma nova geração de gerenciadores de versão, sendo mais confiáveis e fáceis de serem operados. Um desses exemplos é o Git, que surgiu em 2005, com base na experiência de Linus Torvalds e a equipe que cuida do código-fonte do Kernel do Linux. Em pouco tempo, o Git (junto com as plataformas de hospedagem de repositórios Git) se tornou muito popular entre os desenvolvedores, se consolidando como um novo padrão de mercado. (GIT, s.d.)

Essa nova geração tem como principal vantagem o desenvolvimento distribuído, que permite que várias pessoas trabalhem em um mesmo projeto, que está concentrado em um repositório mestre. Cada programador possui uma versão pessoal desse repositório – denominado por Sommerville (2011) como Espaço Privado de Trabalho. Cada programador pode modificar o seu repositório local, implantando as suas mudanças, de maneira independente do repositório central. Ao final, o programador atualiza a sua versão do repositório local, fazendo o *download* das atualizações de seus colegas e submete as suas modificações ao repositório central, passando

a disponibilizar as suas modificações aos seus colegas.

Sommerville denomina o download das alterações dos outros programadores como *check-out* e o *upload* das alterações feitas pelo programador como *check-in*.

Check-out e Check-in, segundo Sommerville

A maior parte do processo de desenvolvimento de software é uma atividade de equipe, por isso, muitas vezes ocorrem situações em que os membros de diferentes equipes trabalham no mesmo componente ao mesmo tempo. Por exemplo, digamos que Alice está fazendo algumas mudanças em um sistema, o que envolve a mudança dos componentes A, B e C. Ao mesmo tempo, Bob está trabalhando em mudanças e estas requerem mudanças nos componentes X, Y e C. Portanto, Alice e Bob, estão mudando C. É importante evitar que tais mudanças interfiram umas nas outras — as mudanças em C feitas por Bob sobrescrevem as de Alice ou vice-versa.

Para apoiar o desenvolvimento independente sem interferência, sistemas de gerenciamento de versões usam o conceito de um repositório público e um espaço de trabalho privado. Os desenvolvedores realizam check-out de componentes de um repositório público em seu espaço de trabalho privado e podem mudá-los como quiserem em seu espaço de trabalho privado. Quando as mudanças orem concluídas, eles realizam o check-in de componentes para o repositório. (...). Se duas ou mais pessoas estiverem trabalhando em um componente ao mesmo tempo, cada uma deve realizar o check-out de componente do repositório. Se em um componente for realizado o check-out, o sistema de gerenciamento de versões normalmente avisará aos outros usuários interessados em realizar o check-out desse componente que já oi realizado check-out de componente por outra pessoa. O sistema também garantirá que, quando os componentes modificados forem registrados, serão atribuídos identificadores de diferentes versões, as quais serão armazenadas separadamente.

Figura 7 - Check-in e check-out a partir de um repositório de versões.

Fonte: SOMMERVILLE, 2011, p. 483

Assim, podemos dizer que há três gerações de ferramentas de controle de versão, que estão descritas na tabela a seguir:

Tabela 1 – As três gerações das ferramentas de controle de versão

Geração	Distribuição	Operações	Concorrência	Exemplos
Primeiro	Nenhuma	Um arquivo por vez	Bloqueio – uma pessoa pode editar um arquivo por vez.	RCS e SCCS
Segundo	Centralizado	Múltiplos arquivos	Merge antes do commit – as alterações são mescladas antes	CVS, SourceSafe, SVN (Subversion), Team Foundation Server
Terceira	Distribuída	Listas de mudanças	Commit antes do merge – as alterações são registradas antes do merge	Git, Bazaar e Mercurial

Fonte: Elaborado com informações de SINK, 2011.

Agora que você entende um pouco do contexto histórico das ferramentas de Controle de Versão, vamos descrever outras características dessas ferramentas. Vamos lá?

2.1 – Características dos gerenciadores de versão

Nesta subseção, vamos descrever algumas características que são comuns aos gerenciadores de versão, que facilitam a vida dos programadores em relação à adoção de um sistema manual de controle de versões.

A primeira característica já mencionamos aqui nesta aula. É o **desenvolvimento independente**, que permite que cada programador trabalhe nas suas mudanças em um repositório local privado, independente do repositório central. Quando as mudanças terminarem de ser implantadas, o programador envia as mudanças ao repositório central, disponibilizando a mudança aos outros programadores.

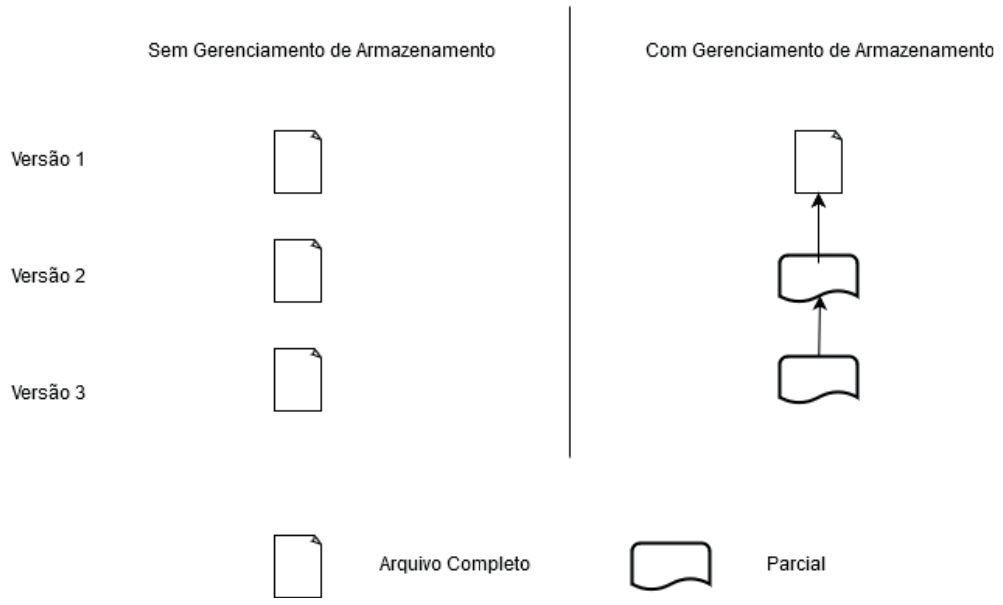
A segunda característica é a **identificação da versão e do release**, que permite que o programador identifique a versão, para que o programador possa dar um número progressivo, identificando se a versão possui novos recursos ou correção de *bugs*:

> Versões gerenciadas recebem identificadores quando são submetidas ao sistema. Normalmente, esses identificadores se baseiam no nome do item de configurações (por exemplo, ButtonManager), seguido por um ou mais números. Por isso, o ButtonManager 1.3 significa a terceira versão na codeline 1 do componente ButtonManager. Alguns sistemas de CM também permitem a associação de atributos com versões (por exemplo, mobile, smallscreen), os quais também podem ser usados para identificação de versão. Um sistema de identificação consistente é importante porque ele simplifica o problema de definição de configurações. Ele simplifica o uso de referências de forma abreviada (por exemplo, *.V2, significando a versão 2 de todos os componentes. (SOMMERVILLE, 2011, p. 482)

A terceira característica é o **gerenciamento de armazenamento**. Essa é uma característica muito importante. Ao invés de armazenar cópias exatas de cada arquivo a cada nova versão, o que poderia acarretar um aumento do espaço consumido, o gerenciador apenas armazena as diferenças entre duas versões, permitindo que o gerenciador reconstrua com eficiência uma determinada versão de um arquivo.

Para entendermos isso, vejamos este exemplo: suponha um repositório com gerenciamento de armazenamento e um repositório sem gerenciamento de armazenamento. Cada repositório armazena três versões. Enquanto o repositório que não adota o gerenciamento de armazenamento armazena três versões completas de um mesmo arquivo, o repositório com gerenciamento de armazenamento armazena a versão completa do arquivo na primeira versão e as diferenças (linhas que foram adicionadas e removidas) nas versões seguintes, em relação à primeira versão.

Figura 8 – Diferença entre um repositório com gerenciamento de armazenamento e um repositório sem gerenciamento de armazenamento.

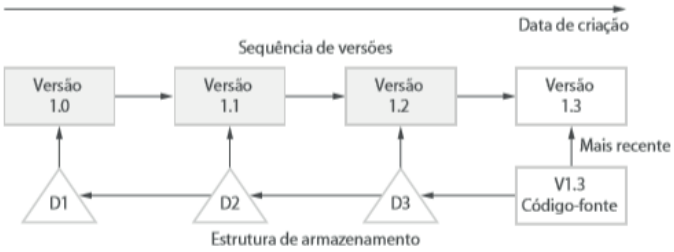


Fonte: Acervo Pessoal.

Isso propicia economia do armazenamento dos repositórios, permitindo que mais arquivos e versões sejam armazenados. Sommerville denomina essas diferenças como deltas.

> Quando foram desenvolvidos os sistemas de gerenciamento de versões, o gerenciamento de armazenamento foi uma de suas funções mais importantes. Os recursos de gerenciamento de armazenamento em um sistema de controle de versões reduzem o espaço requerido em disco para manter todas as versões de sistema. Quando uma nova versão é criada, o sistema simplesmente armazena um delta (uma lista de diferenças) entre a nova versão e a mais antiga, usada para criar essa nova versão (...). Normalmente, os deltas são armazenados como listas de linhas alteradas e, ao aplicá-los automaticamente, uma versão de um componente pode ser criada a partir de outro. Como é mais provável que a versão mais recente de um componente será usada, a maioria dos sistemas armazena essa versão na íntegra. Os deltas definem como recriar versões anteriores de sistema (SOMMERVILLE, 2011, p. 483).

Figura 9 - Gerenciamento de armazenamento usando deltas.

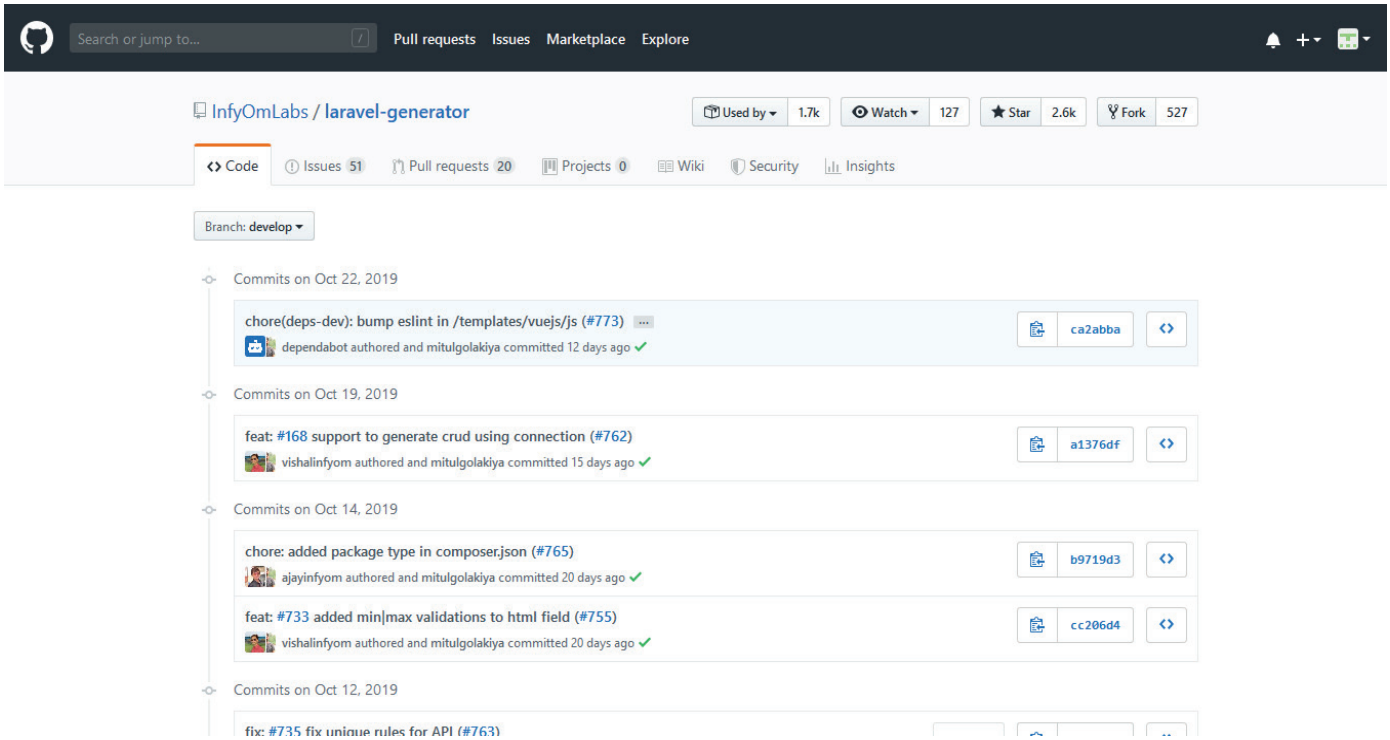


Fonte: SOMMERVILLE, 2011, p. 483

As diferenças entre versões podem ser registradas em relação a última versão ou em relação a primeira versão. Isso depende da forma em que o sistema foi implementado.

Prosseguindo os nossos estudos, temos a característica do **Registro do Histórico de Mudanças**, que nos permite verificar e listar quais foram as mudanças realizadas e quem fez. Adicionalmente, as "mudanças podem ser usadas para selecionar uma versão específica do sistema", envolvendo a marcação dos componentes com as palavras-chave que descrevem quais foram essas mudanças. (SOMMERVILLE, 2011, p. 482)

Figura 10 – Histórico de versões de um repositório público disponível no Github. Perceba que cada mudança está associada a uma pessoa, identificando quem fez a mudança.



Fonte: Gerado a partir do site: https://github.com/InfyOmLabs/laravel-generator/commits/develop. Acesso em 03 nov. 2019.

Além disso, um sistema de controle de versões pode oferecer **suporte a projetos**, permitindo "o apoio de desenvolvimentos de vários projetos que compartilham componentes" (SOMMERVILLE, 2011, p. 482).

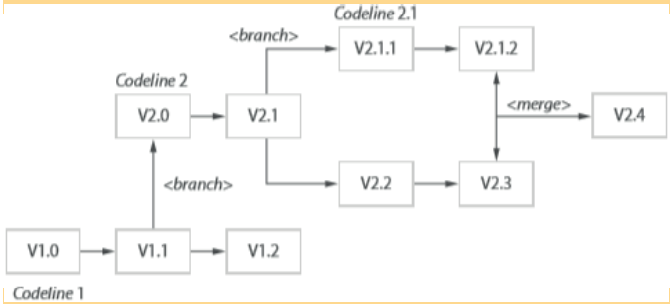
Por fim, temos a possibilidade dos desenvolvedores fazerem **ramificações no histórico do código**, permitindo que mudanças sejam aplicadas sem afetar o histórico base do sistema. Você já viu a respeito disso na aula 01. Estamos nos referindo a **branches e merges**. Vamos ler o que Sommerville diz a respeito disso?

Ramificação do histórico de mudanças

Uma consequência do desenvolvimento independente do mesmo componente é que codelines podem se ramificar. Em vez de uma sequência linear de versões que refletem as mudanças para o componente ao longo do tempo, pode haver várias sequências independentes (...). Isso é normal no desenvolvimento de sistemas, em que diferentes desenvolvedores trabalham independentemente em diferentes versões do código-fonte e fazem mudanças de maneiras diferentes.

Em algum momento, pode ser necessário fundir ramificações de codelines para criar uma nova versão de um componente que inclui todas as mudanças realizadas. (...). Se as mudanças feitas envolverem partes completamente diferentes do código, as versões de componente poderão ser fundidas automaticamente pelo sistema de gerenciamento de versões por meio da combinação dos deltas que se aplicam ao código. Frequentemente, existem sobreposições entre as mudanças feitas, e estas interferem umas nas outras. Um desenvolvedor deve verificar se existem conflitos e modificar as mudanças para que estas sejam compatíveis.

Figura 11 – Branching e Merging



Fonte: SOMMERVILLE, 2011, p. 484-483

E, com isso, encerramos esta seção a respeito de Gerenciamento de Versões. Mas vamos voltar a ver novamente, quando estudarmos o sistema de controle de versões Git, a partir da aula 06.

Esta seção marca também o fim da nossa aula. Na próxima aula, vamos estudar as outras duas atividades de Gerência de Configuração: Gerenciamento de Releases e Construção de Sistema. Até lá!

Retomando a aula

Chegamos ao final da nossa terceira aula. Vamos relembrar?

1 – Gerenciamento de versões

Você viu que a atividade de Gerenciamento de Versões pode ser definida como um processo de acompanhamento de diferentes versões de componentes de software ou itens de configuração e os sistemas em que os componentes são usados. Você viu aqui também um breve histórico dessas ferramentas e quais os recursos que vemos em uma ferramenta que permita o controle de versões.

Vale a pena

Vale a pena **ler**

ENGHOLM JR., Hélio. *Engenharia de software na prática*. São Paulo: Novatec, 2010.

SOMMERVILLE, Ian. *Engenharia de Software*. 9. ed. São Paulo: Pearson, 2011.

AQUILES, Alexandre; FERREIRA, Rodrigo. *Controlando versões com Git e GitHub*. São Paulo: Casa do Código, 2017.

Vale a pena **acessar**

FREE SOFTWARE FOUNDATION. GNU RCS. Disponível em: http://www.gnu.org/software/rcs/. Acesso em 03 nov. 2019.

GIT. Uma Breve História do Git. s.d. Disponível em: https://git-scm.com/book/pt-br/v1/Primeiros-passos-Uma-Breve-Hist%C3%B3ria-do-Git. Acesso em: 03 nov. 2019.

ORACLE. Chapter 5 SCCS Source Code Control System. s.d. Disponível em: https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dhp/index.html. Acesso em 03 nov. 2019.

SINK, Eric. Version Control by Example. 2011. Disponível em: https://ericsink.com/vcbe/html/index.html. Acesso em: 03 nov. 2019