Desenvolvimento de Sistemas

Gerenciamento de configuração: conceitos, *softwares* e ferramentas

Introdução

Olá, aluno(a)! Até este momento do curso, você já percorreu várias etapas de aprendizado, explorando variados estágios do desenvolvimento de um sistema. Nesse processo, você provavelmente conseguiu perceber que, em algumas situações, ainda que seu projeto estivesse funcionando, era necessário acrescentar novos ajustes, alterações e funcionalidades, que muitas vezes podiam se sobrepor aos códigos já elaborados anteriormente, gerando versões diferentes de um mesmo projeto (os clássicos “atividade1”, “atividade1 atualizado”, “atividade1 versaoFinal”, “atividade1essaÉpraEnviar”).

Quando se está nessa situação, até que o ajuste seja concluído, pode acontecer de você não conseguir executar corretamente o que foi finalizado anteriormente e estava funcionando, ou mesmo perder funcionalidades que tinha escrito anteriormente. Caso tenha desenvolvido ajustes para muitos arquivos diferentes, pode acabar sendo necessário remover boa parte dos novos códigos adicionados para recomeçar a lógica desde o início. O controle desses procedimentos pode ser um pouco trabalhoso, não é mesmo?

Agora, imagine esse cenário em um grande projeto em que diversas pessoas precisam programar códigos diferentes ao mesmo tempo! Como seria possível juntar todos os ajustes (ou a remoção destes) sem causar nenhum conflito? Como saber

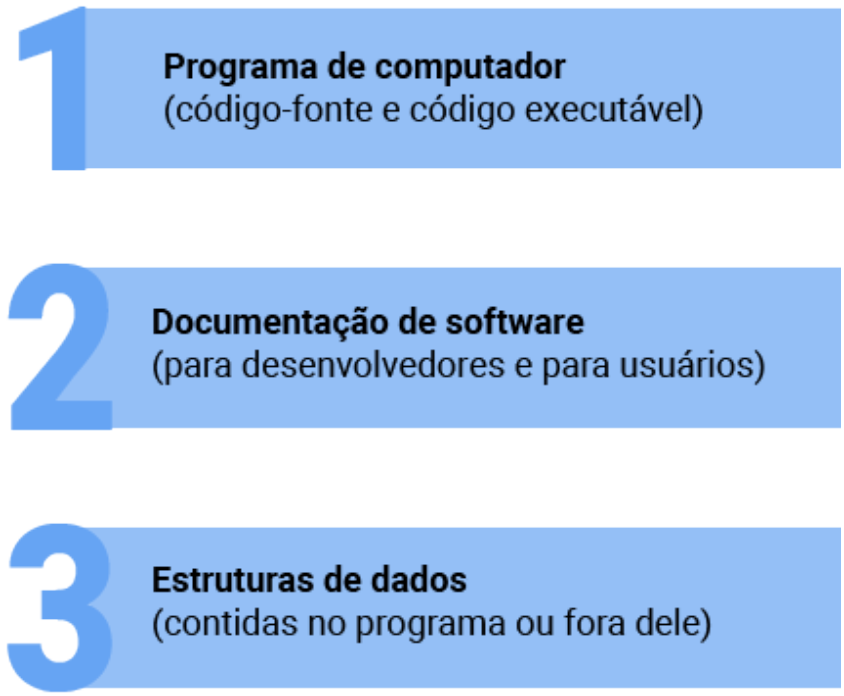
quem foi o responsável por determinada alteração que está gerando conflito com outro ajuste? Entre todos os projetos, como saber qual é a versão mais recente para seguir atualizando? Como saber qual era a última versão em que ainda estava funcionando o projeto?

Esses processos podem causar uma grande “dor de cabeça” caso você não esteja familiarizado com gerenciamento de configuração.

Gerenciamento de configuração

O gerenciamento de configuração é uma atividade abrangente presente na engenharia de *software* e que visa identificar mudanças em um projeto de *software*, controlar as mudanças, garantir sua adequada implementação e relatar essas mudanças a outras pessoas que possam ter interesse.

Configuração de *software* é um conjunto de artefatos produzidos e presentes em um projeto, divididos em três categorias:



O que é versionamento de código?

Uma versão de código é cada agrupamento de atualizações desenvolvidas para o sistema. Um versionamento de código, portanto, nada mais é do que o ato de gerenciar o desenvolvimento de várias etapas de um sistema e, quando concluídas corretamente, uni-las em uma nova versão, mantendo também o histórico de versões anteriores, sendo possível analisar os progressos e as alterações aplicadas ou até mesmo voltar em uma etapa em que o sistema ainda funcionava de outra maneira, além de ter outras diversas funcionalidades.

Esse processo tornou-se uma necessidade na área da programação, sendo uma forma de obter maior controle do que acontece no decorrer de um projeto.

No mercado de trabalho, é comum que vários desenvolvedores estejam trabalhando em um mesmo sistema e, com isso, um mesmo arquivo seja alterado por várias pessoas diferentes. Considerando que é um trabalho que ocorre de forma simultânea, não é possível aguardar que um termine para que outro desenvolvedor possa editar o arquivo atualizado. Com o versionamento, cada usuário edita a própria funcionalidade separadamente para unir, ao término, todos os ajustes em uma única versão final.

Outro cenário em que o versionamento deve ser aplicado pode ser analisado em casos em que desenvolvedor almeja modificar funcionalidades do sistema, entretanto mantendo a versão antiga ainda ativa para uso. Um exemplo bem comum em que se pode observar esse cenário é o sistema operacional Android. Veja, a seguir, um exemplo da progressão de versões desse sistema.

- Android 13 – Última versão: 15/08/2022
- Android 12 – Última versão: 04/10/2021
- Android 11 – Última versão: 08/09/2020
- Android 10 – Última versão: 22/08/2019
- Android 9.0 Pie – Última versão: 06/08/2018
- Android 8.1 Oreo – Última versão: 05/12/2017
- Android 8.0 Oreo – Lançamento da versão: 05/12/2017
- Android 7.1.2 Nougat – Última versão: 02/04/2017
- Android 7.1.1 Nougat – Lançamento da versão: 06/12/2016
- Android 7.1 Nougat – Lançamento da versão: 04/10/2016
- Android 7.0 Nougat – Lançamento da versão: 01/09/2016
- Android 6.0.1 Marshmallow – Última versão: 07/03/2016
- Android 6.0 Marshmallow – Lançamento da versão: 29/09/2015
- Android 5.1.1 Lollipop – Última versão: 21/04/2015
- Android 5.1 Lollipop – Lançamento da versão: 09/03/2015
- Android 5.0.2 Lollipop – Lançamento da versão: 20/12/2014
- Android 5.0.1 Lollipop – Lançamento da versão: 02/12/2014
- Android 5.0 Lollipop – Lançamento da versão: 17/10/2014
- Android 4.4.4 KitKat – Última versão: 19/06/2014
- Android 4.4.3 KitKat – Lançamento da versão: 01/04/2014
- Android 4.4.2 KitKat – Lançamento da versão: 10/12/2013
- Android 4.4 KitKat – Lançamento da versão: 31/10/2013

Por meio dessa listagem de versões, é possível observar que, embora existam versões mais recentes, atualizadas e otimizadas do sistema operacional, versões antigas ainda seguem ativas, sendo utilizadas em *smartphones* que têm um *hardware*

mais antigo. Dessa forma, o sistema é inovado e suas funcionalidades são atualizadas, visando atender novos usuários com tecnologias mais modernas ao mesmo tempo que retém o público de versões mais antigas.

Esses são alguns dos exemplos em que se pode perceber que aplicar os processos de versionamento de código auxilia na organização e no desenvolvimento de um projeto. Existem diversas outras situações em que os desenvolvedores são beneficiados com o uso dessa prática. Com base no que você leu e em suas experiências, consegue imaginar outros cenários favorecidos por esse versionamento?

Apesar de o exemplo anterior tratar de versões finais de sistemas operacionais, durante o processo de desenvolvimento existe também versionamento dos arquivos até que a última versão seja lançada. Observe a imagem a seguir.

(imgs/fig1.png)

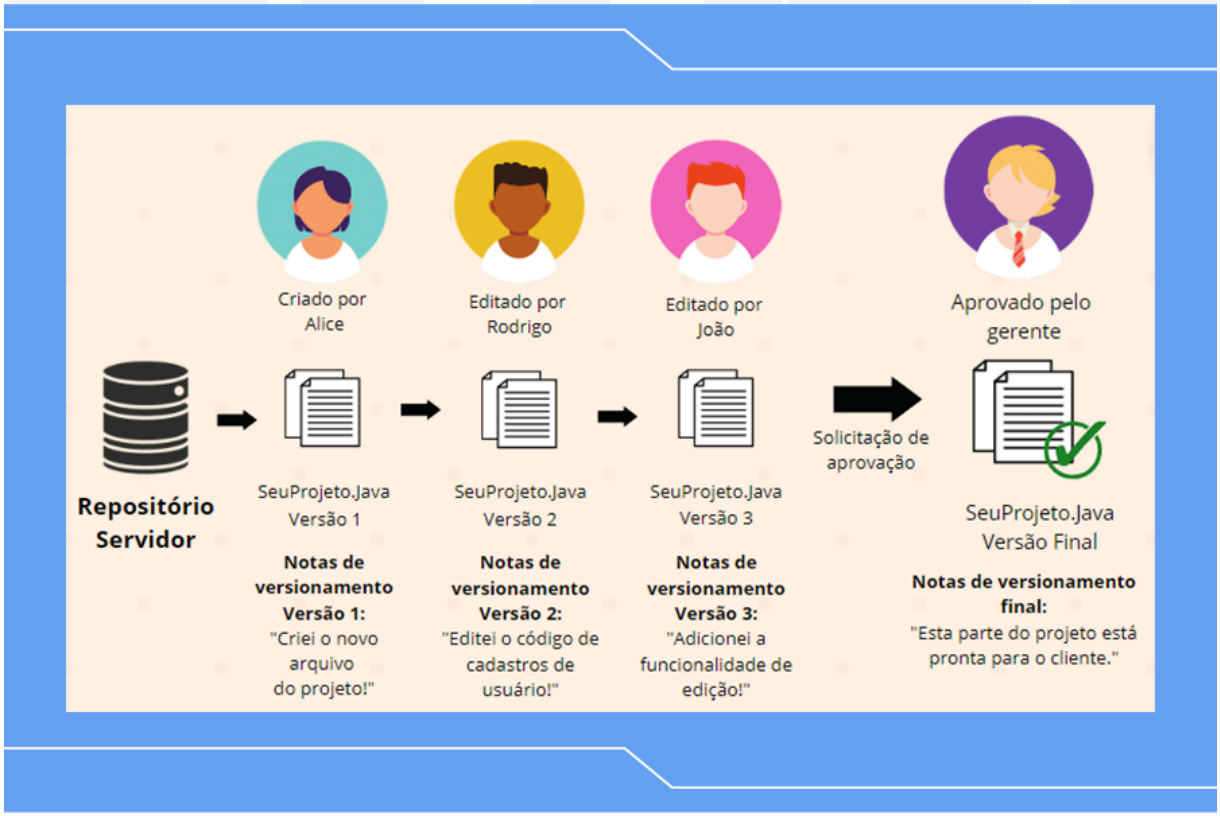


Figura 1 – Versionamento de código em arquivos de um projeto

Fonte: adaptado de Document Locator (c2023)

Veja que nesse processo o arquivo que chegou até a versão final foi manipulado por diversos desenvolvedores. Cada um deles criou ou ajustou algo diferente em um mesmo arquivo que faz parte de um projeto, fazendo com que ele estivesse pronto para uso do cliente. Esse arquivo que chegou à versão final fará parte do conjunto de arquivos atualizados que comporão uma nova versão final do sistema por completo.

Embora a ideia geral seja muito proveitosa, é necessário utilizar ferramentas adequadas para realizar versionamento de código. Criar várias pastas com cópias do projeto no computador ou colocar várias cópias do mesmo projeto na nuvem, por exemplo, não somente será pouco eficaz como também poderá causar mais desorganização e confusão no código.

Softwares e ferramentas de gerenciamento de configuração

Ainda que o conceito de gerenciamento de configuração seja aplicado no seu projeto, caso seja aplicado de forma leviana, não trará bons resultados. Como mencionado anteriormente, fazer várias cópias de arquivo no próprio computador ou na nuvem não é uma solução adequada. Considerando isso, é importante que algumas ferramentas de *software* sejam usadas para esse processo. É possível separá-las em três categorias: ferramentas de controle de modificação, ferramentas de controle de versão e ferramentas de gerenciamento de construção.

Ferramentas de controle de modificação

Os *softwares* de controle de modificação podem ser usados para realizar o acompanhamento das alterações realizadas no *software*, registrando as tarefas necessárias e o histórico da cada uma. O sistema é, portanto, responsável por armazenar o andamento das solicitações de mudança e por notificar essas informações aos interessados. Veja alguns exemplos de sistema de controle de modificação:

Bugzilla

Sistema *open source* de *bug tracking* (ou acompanhamento de defeitos) baseado em *web* no qual é possível registrar defeitos e solicitações de um *software*. É ideal para trabalho em times.

Jira

Sistema de gerenciamento de trabalho em equipe baseado no Kanban, mais abrangente que o Bugzilla, pois, além de *bug tracking*, traz ferramentas mais avançadas para controlar o fluxo de trabalho em equipe.

Trac

Sistema que conta com recursos de criação e acompanhamento de *tickets* para ajustes no *software*, integrando também controles de versão e documentações *wiki*.

Ferramentas de gerenciamento de construção

As ferramentas de gerenciamento de construção são responsáveis por auxiliar ou automatizar as tarefas de construção (*build*) do código-fonte em código executável. É o processo, por exemplo, de transformar código Java em arquivos JAR executáveis. Veja dois exemplos de ferramentas de construção:

Apache Ant

Permite centralizar a configuração de elementos e opções de compilação para um projeto Java, coordenando os processos de construção do executável.

Maven

Usada geralmente em projetos Java, facilita o gerenciamento das dependências do código, baseando-se em um repositório central de onde o sistema, automaticamente no processo de compilação, baixa a versão da biblioteca e inclui na construção do *software*.

Ferramentas de controle de versão

Ferramentas de versionamento são os sistemas utilizados para realizar todo o gerenciamento de desenvolvimento e evolução do código para novas versões. Pode-se afirmar que o processo de versionamento de código e as ferramentas de versionamento estão intrinsecamente ligados para a obtenção dos melhores resultados.

De um modo geral, embora existam diversas ferramentas de versionamento, é possível identificar alguns itens como benefícios em comum. Veja a seguir.

- Todas as alterações aplicadas no código ficam registradas em um histórico, sendo possível verificar sempre que um ajuste foi aplicado. Esse histórico possibilita saber quem foi o usuário que aplicou as alterações e quais alterações foram desenvolvidas de uma versão para a outra.
- Várias pessoas podem colaborar simultaneamente com o projeto de forma segura, facilitando o trabalho em equipe.
- Somente pessoas autorizadas terão acesso e poderão modificar arquivos, facilitando o gerenciamento dos desenvolvedores envolvidos no projeto e reduzindo a quantidade de erros nesse processo.
- Facilidade em fazer alterações no código sem prejudicar outras versões do projeto.

Outro aspecto importante a ser considerado antes de conhecer as ferramentas é entender que existem dois tipos de sistemas de controle de versão. Veja a seguir quais são eles.

Centralizado

Usa-se um servidor central para alocar um único repositório central do projeto, que será acessado e manipulado pelos integrantes do time ao realizar modificações nos arquivos do *software*. Cada integrante precisa conectar-se a esse servidor para usar o versionamento.

Distribuído

Cada usuário que for desenvolver código terá uma cópia completa do repositório central diretamente em seu ambiente local, contando com alguns comandos de versionamento local. Em algum momento, no entanto, deve refletir as alterações locais para um servidor central, para que os outros integrantes do time recebam as atualizações.

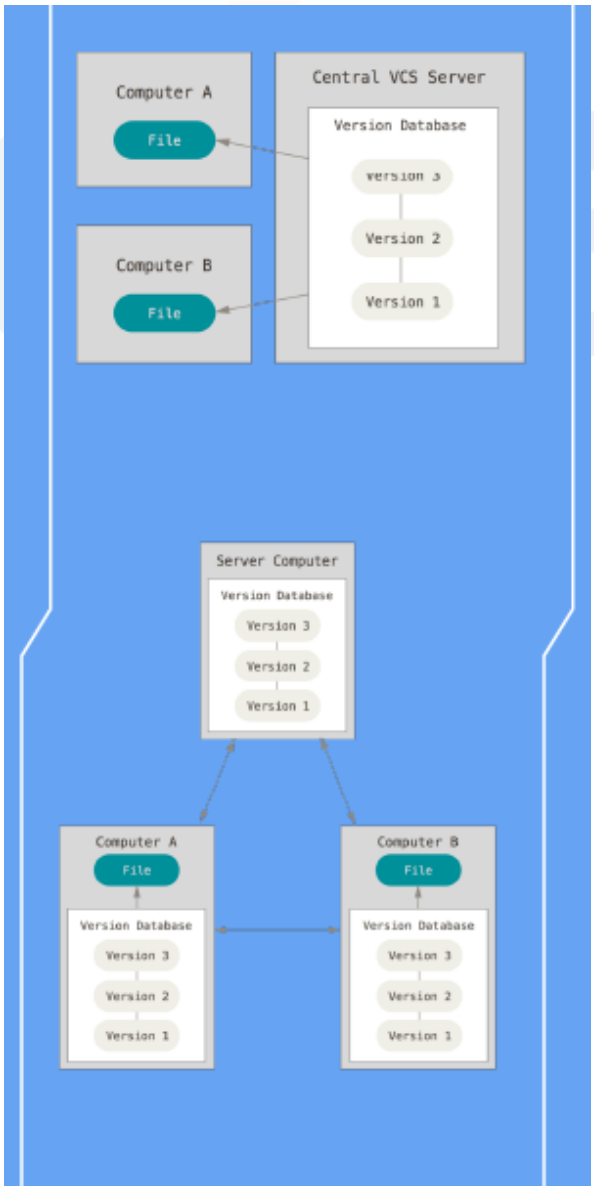


Figura 2 – Sistema de controle de versão centralizado

Fonte: Senac EAD (2023)

Figura 3 – Sistema de controle de versão distribuído
Fonte: Senac EAD (2023)

Veja, a seguir, as ferramentas de versionamento mais populares que se encontram no mercado.

Git

O Git é um dos sistemas de versionamento *open source* mais usado pela comunidade de desenvolvedores, adequando-se com facilidade a projetos de pequena e de grande escala. Ele tem uma grande variedade de comandos, podendo ser um pouco complexo à primeira vista, entretanto sua rápida *performance* compensa. Nesta unidade curricular (UC), você aprenderá como trabalhar com esse sistema.

SVN (Apache Subversion)

O Apache Subversion surgiu no ano de 2000 como uma alternativa ao controle de versão CVS (Concurrent Version System), que foi lançado em 1986. O SVN é um sistema de controle de versão centralizado, gratuito e *open source*.

Mercurial

Também lançado em 2005, o Mercurial é outra opção quando se busca um sistema de controle de versão gratuito distribuído. Ele tem comandos e operações mais simples, similares ao Subversion.

Git

O Git é um dos sistemas de versionamento *open source* mais usado pela comunidade de desenvolvedores, adequando-se com facilidade a projetos de pequena e de grande escala. Ele tem uma grande variedade de comandos, podendo ser um pouco complexo à primeira vista, entretanto sua rápida *performance* compensa. Nesta unidade curricular (UC), você aprenderá como trabalhar com esse sistema.

SVN (Apache Subversion)

O Apache Subversion surgiu no ano de 2000 como uma alternativa ao controle de versão CVS (Concurrent Version System), que foi lançado em 1986. O SVN é um sistema de controle de versão centralizado, gratuito e *open source*.

Mercurial

Também lançado em 2005, o Mercurial é outra opção quando se busca um sistema de controle de versão gratuito distribuído. Ele tem comandos e operações mais simples, similares ao Subversion.

Veja, a seguir, as ferramentas de versionamento mais populares que se encontram no mercado.

Git e GitHub no *open source*

Quando se fala sobre sistemas de controle de versionamento, um fator muito importante é o local onde os dados serão disponibilizados para que os desenvolvedores tenham acesso aos arquivos e possam atualizá-los. Esse local é chamado de repositório.

Quando se menciona o uso do Git como ferramenta mais utilizada atualmente, é possível lembrar do GitHub em razão da similaridade dos nomes. Embora os nomes sejam similares, as funcionalidades dessas ferramentas são bem diferentes e complementares.

Vale destacar também que existem diversas outras opções de repositório, como GitLab, BitBucket, GitKraken, entre outros. Entretanto, em razão de seu maior uso no mercado, no conteúdo desta UC o foco será o GitHub.

Sabe-se que o Git serve para gerenciar todas as alterações realizadas em um projeto, vinculando usuários e históricos e, por fim, gerando novas versões de um projeto. Mas e quanto ao GitHub, você já tem alguma ideia do que é? Saiba agora, vendo a definição apresentada pelo próprio *site* do GitHub:

GitHub é uma plataforma de hospedagem de código para controle de versão e colaboração. Permite que você e outras pessoas trabalhem em conjunto em projetos de qualquer lugar (LET'S, c2023).

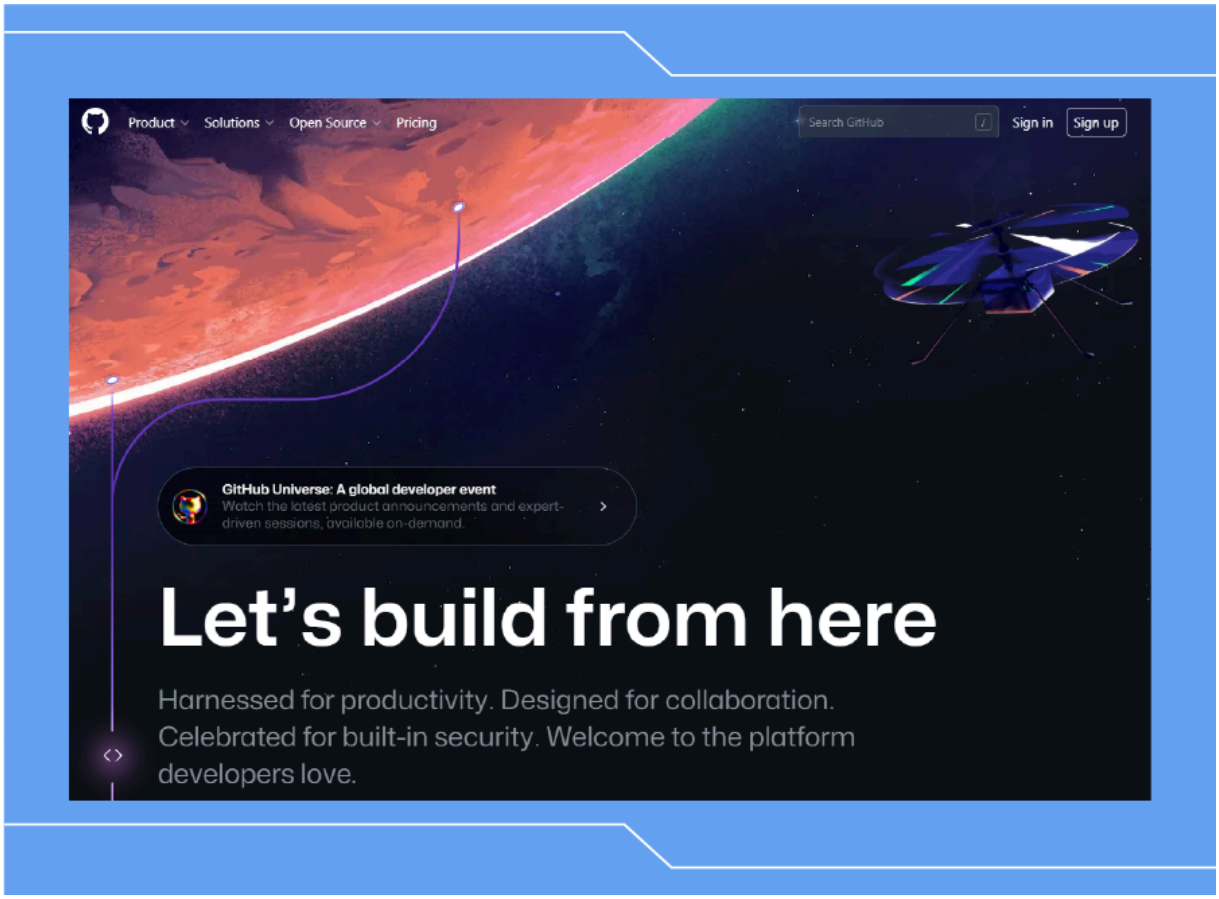


Figura 4 – Página inicial do GitHub

Fonte: GitHub (c2023)

Essa ampla visão que o GitHub apresenta abre inúmeras possibilidades quando se utiliza o Git em conjunto com a plataforma, pois o GitHub não atua somente como repositório, afinal ele tem várias outras funcionalidades que facilitam a comunicação entre desenvolvedores. Esse tipo de estrutura foi um enorme estímulo aos desenvolvedores para que compartilhassem seus projetos e tentassem auxiliar o desenvolvimento de outros.

Projetos disponibilizados publicamente onde qualquer pessoa pode acessar, analisar e adaptar o código são chamados de projetos *open source*. As razões para esse tipo de código ser disponibilizado são várias, por exemplo: proporcionar possibilidade de estudo de código a quem estiver interessado; auxiliar a comunidade a desenvolver novas versões melhores de determinado sistema; e disponibilizar bibliotecas gratuitas para uso pessoal ou comercial. As vantagens são inúmeras, desde a curiosidade até o uso do código de forma prática.

Embora a ideia de projetos *open source* pareça bastante ampla, vale destacar que existem várias licenças *open source*, ou seja, mesmo que o projeto seja aberto ao público, poderá ter algumas limitações ou obrigatoriedades em seu uso.

Veja, a seguir, alguns exemplos de licenças.

Apache License 2.0

Permissões

Uso comercial: o material licenciado e os derivados podem ser usados para fins comerciais.

Distribuição: o material licenciado pode ser distribuído.

Modificação: o material licenciado pode ser modificado.

Uso de patente: essa licença fornece uma concessão expressa de direitos de patente de contribuidores.

Uso privado: o material licenciado pode ser usado e modificado em particular.

Condições

Aviso de licença e direitos autorais: uma cópia da licença e o aviso de direitos autorais devem ser incluídos com o material licenciado.

Mudanças de estado: as alterações feitas no material licenciado devem ser documentadas.

Mozilla Public License 2.0

Permissões

Uso comercial: o material licenciado e os derivados podem ser usados para fins comerciais.

Distribuição: o material licenciado pode ser distribuído.

Modificação: o material licenciado pode ser modificado.

Uso de patente: essa licença fornece uma concessão expressa de direitos de patente de contribuidores.

Uso privado: o material licenciado pode ser usado e modificado em particular.

Condições

Divulgação da fonte: o código-fonte deve ser disponibilizado quando o material licenciado for distribuído.

Aviso de licença e direitos autorais: uma cópia da licença e o aviso de direitos autorais devem ser incluídos com o material licenciado.

Mesma licença (arquivo): modificações de arquivos existentes devem ser liberadas sob a mesma licença ao distribuir o material licenciado. Em alguns casos, uma licença semelhante ou relacionada pode ser usada.

GPLv3

Permissões

Uso comercial: o material licenciado e os derivados podem ser usados para fins comerciais.

Distribuição: o material licenciado pode ser distribuído.

Modificação: o material licenciado pode ser modificado.

Uso de patente: essa licença fornece uma concessão expressa de direitos de patente de contribuidores.

Uso privado: o material licenciado pode ser usado e modificado em particular.

Condições

Divulgação da fonte: o código-fonte deve ser disponibilizado quando o material licenciado for distribuído.

Aviso de licença e direitos autorais: uma cópia da licença e o aviso de direitos autorais devem ser incluídos com o material licenciado.

Mesma licença: as modificações devem ser liberadas sob a mesma licença ao distribuir o material licenciado. Em alguns casos, uma licença semelhante ou relacionada pode ser usada.

Mudanças de estado: as alterações feitas no material licenciado devem ser documentadas.

MIT

Permissões

Uso comercial: o material licenciado e os derivados podem ser usados para fins comerciais.

Distribuição: o material licenciado pode ser distribuído.

Modificação: o material licenciado pode ser modificado.

Uso privado: o material licenciado pode ser usado e modificado em particular.

Condições

Aviso de licença e direitos autorais: uma cópia da licença e o aviso de direitos autorais devem ser incluídos com o material licenciado.

BSD 3

Permissões

Uso comercial: o material licenciado e os derivados podem ser usados para fins comerciais.

Distribuição: o material licenciado pode ser distribuído.

Modificação: o material licenciado pode ser modificado.

Uso privado: o material licenciado pode ser usado e modificado em particular.

Condições

Aviso de licença e direitos autorais: uma cópia da licença e o aviso de direitos autorais devem ser incluídos com o material licenciado.

Diante desses tipos de licença encontrados entre os projetos *open source* (Apache License 2.0; Mozilla Public License 2.0; GPLv3, sigla para General Public License *version* 3; licença MIT, sigla para Massachusetts Institute of Technology; e licença BSD 3, sigla para Berkeley Software Distribution 3), é importante estar atento às permissões e às condições apresentadas. Muitas das permissões são comuns entre si, entretanto as condições de uso podem variar e devem ser analisadas com atenção. Note também que essa listagem apresenta somente as mais utilizadas, mas existem outros tipos de licenças que podem estar vinculadas a um projeto *open source*.

Encerramento

Logo no início deste conteúdo, você pode até ter se questionado: mas, afinal, qual é a diferença entre utilizar ferramentas de versionamento e fazer cópias do meu sistema na nuvem?

Ao longo do material, você pôde acompanhar os diversos benefícios do uso de um *software* para versionamento. Embora você possa fazer cópias de suas pastas ou salvá-las na nuvem, há alguns aspectos que tornam essencial a utilização de ferramentas corretas: os registros de alterações, a facilidade para trabalhar em grupo e a facilidade para criar novas versões de sistema e juntá-las com ajustes realizados por outros desenvolvedores, por exemplo.

A possibilidade de trabalhar em projetos *open source* ou a possibilidade de utilizar essas bibliotecas no próprio código também são enormes benefícios. Não esqueça também de que, atualmente, boa parte das empresas analisa os projetos publicados no GitHub para realizar novas contratações, fazendo com que essa plataforma se torne um ótimo portfólio para a sua jornada.

Prepare-se para, nos próximos conteúdos, aprofundar-se mais em alguns dos aspectos mencionados neste material!