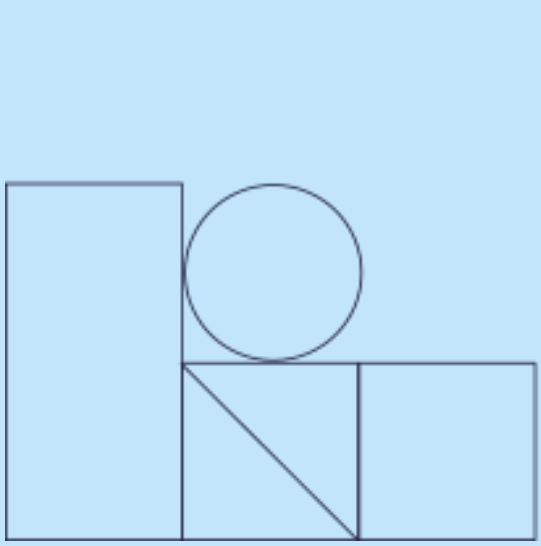
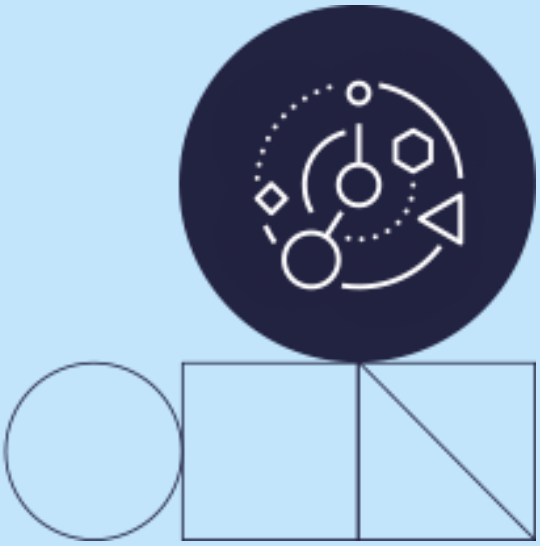
IBM SkillsBuild

IBM

Git e Github

O que é Git e Github





Índice

Introdução

Git é um projeto open source que começou em 2005 e cresceu até se tornar um dos VCSs mais populares no mercado: cerca de **87% dos developers** usam Git para seus projetos.

Git é um sistema de controle de versão. Uma ferramenta utilizada para armazenar versões de nosso código.

Ao contrário dos sistemas de controle de versão centralizada, Git oferece filiais de características. Isto significa que cada engenheiro de software da equipe pode dividir um ramo de recursos que fornecerá um repositório local isolado para fazer alterações no código.

Os ramos de características não afetam o ramo principal, que é onde se encontra o código original do projeto. Uma vez que as mudanças tenham sido feitas e o código atualizado esteja pronto, o ramo de recursos pode ser fundido novamente no ramo principal, que é como as mudanças no projeto entrarão em vigor.

Também nos permite manter um histórico de versões, ou seja, várias versões paralelas do mesmo software, corrigir simultaneamente bugs...etc. Quando várias pessoas estão trabalhando no mesmo software ao mesmo tempo, o GIT se torna especialmente útil.

Antes do aparecimento desta ferramenta, a forma de trabalho era salvar manualmente as diferentes versões de nosso código localmente. Assim, armazenamos em nosso disco rígido a versão V1.0 do programa, V2.0....etc. Desta forma, poderíamos ter um histórico de versões de código e, ao trabalhar em grupo, cada equipe tinha uma maneira de compartilhar o conteúdo e implementar o código.

Muitas vezes os programadores "pisavam" no trabalho uns dos outros e havia sempre a ameaça de que um passo em falso, ou um erro de um programador, acabaria com o trabalho de todos.

GIT tira um instantâneo de nosso projeto para nós no estado em que se encontra, para que, se ocorrer um problema, possamos voltar atrás e recarregar esse instantâneo (ou os anteriores) e não perder o trabalho. A esta foto a GIT anexa um código alfanumérico de registro e a descrição que lhe dizemos. Assim, teremos um histórico de versões de nosso código com suas respectivas descrições para voltar atrás, se necessário.

Também permite que vários programadores trabalhem no mesmo código ao mesmo tempo, para que o trabalho de um programador não interfira com o dos outros. Ele é capaz de detectar que dois ou mais programadores estão trabalhando na mesma parte do código e gerencia a fusão de ambos os trabalhos através de filiais para que não haja conflitos entre os dois códigos.

Git tem três "áreas" diferentes para nosso código:

- **Diretório de trabalho**: a área onde você fará todo o seu trabalho (criação, edição, eliminação e organização de arquivos).
- **Área de armazenamento**: a área onde serão listadas as mudanças que fizemos no diretório de trabalho.
- **Repositório**: onde Git armazena permanentemente as mudanças que fizemos, tais como diferentes versões do projeto.

Por que usar Git?

Git e Github são usados na vida diária das pessoas que criam software por uma razão muito simples: ter uma maneira fácil de gerenciar a aplicação, o sistema e o código fonte do produto.

Em uma pequena equipe, algumas pessoas ainda tentam cuidar desses arquivos de forma algo questionável: compartilhando diretórios na rede, usando ferramentas como Dropbox ou mantendo tudo em um servidor FTP. Práticas que hoje são fortemente desencorajadas.

Não é suficiente poder acessar o código de outros contribuintes. Precisamos manter a história de nossos arquivos. Mais: de nossas modificações, já que muitas vezes mudamos arquivos como um grupo, em um único movimento (um compromisso). Dessa forma, podemos voltar e recuperar o estado do sistema como era ontem ou no ano passado, e comparar as mudanças, encontrar um bug, estudar otimizações... etc.

Todos os nossos arquivos, assim como seus históricos, estão em um repositório e existem vários sistemas que gerenciam tais repositórios, tais como CVS e SVN.

Git é uma alternativa que funciona ainda melhor: ele é distribuído e todos têm uma cópia completa do repositório, não apenas o "servidor principal".

***Git* é um sistema de controle de versões amplamente adotado e distribuído**. *Git* nasceu e ocupou o espaço de outros sistemas de controle. Seu principal criador é o mesmo que o Linux: Linus Torvalds e conquistou os corações das pessoas que trabalham com código aberto.

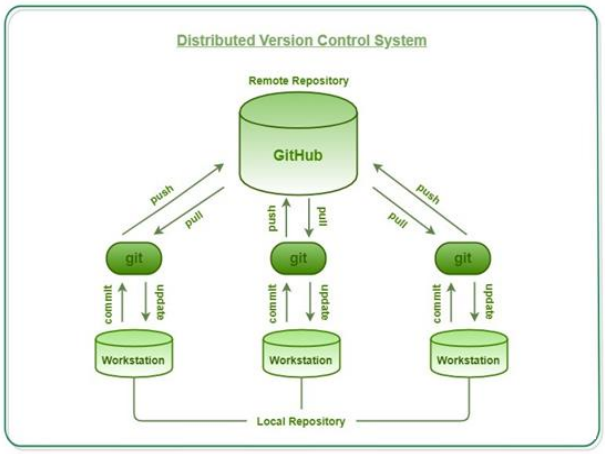
Características do Git

Sistema distribuído

Os sistemas distribuídos são aqueles que permitem aos usuários realizar trabalhos em um projeto de todo o mundo. Um sistema distribuído contém um repositório central que pode ser acessado por muitos colaboradores remotos através de um sistema de controle de versão. Git é um dos sistemas de controle de versão mais populares em uso hoje em dia.

Ter um servidor central resulta em um problema de perda ou desconexão de dados no caso de uma falha no sistema do servidor central. Para resolver este tipo de situação, Git espelha todo o repositório em cada instantâneo da versão que está sendo verificada pelo usuário. Neste caso, se o servidor central falhar, a cópia dos repositórios pode ser recuperada dos usuários que fizeram o download do último instantâneo do projeto.

Ao ter um sistema distribuído, Git permite aos usuários trabalhar simultaneamente no mesmo projeto, sem interferir com o trabalho de outros. Quando um determinado usuário termina sua parte do código, ele compromete as mudanças no repositório e estas mudanças são atualizadas na cópia local de todos os outros usuários remotos que verificam a última cópia do projeto.



Compatibilidade

Git é compatível com todos os sistemas operacionais em uso nos dias de hoje. Os repositórios Git também podem acessar os repositórios de outros sistemas de controle de versões, como SVN, CVK, etc. Git pode acessar diretamente os repositórios remotos criados por esses SVNs. Portanto, os usuários que não estavam usando Git em primeiro lugar também podem mudar para Git sem passar pelo processo de copiar seus arquivos de outros repositórios VCS para Git-VCS. Git também pode acessar os repositórios centrais de outros VCSs. Portanto, pode-se trabalhar no Git-SVN e usar o repositório central como o mesmo. Git tem uma emulação de servidor CVS, que permite o uso de clientes CVS e plugins IDE existentes para acessar os repositórios Git.

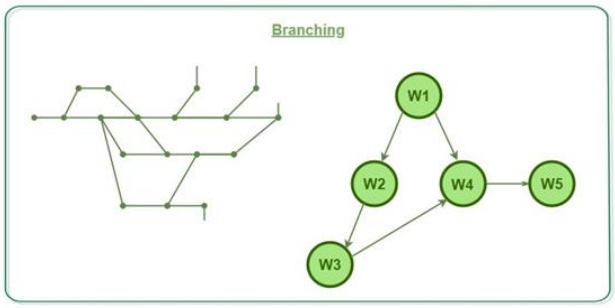
Desenvolvimento não-linear

Git permite aos usuários de todo o mundo realizar operações em um projeto remotamente. Um usuário pode selecionar qualquer parte do projeto e realizar a operação necessária e, em seguida, atualizar o projeto. Isto pode ser feito usando o comportamento de desenvolvimento não-linear de Git. Git apoia a rápida ramificação e fusão e inclui ferramentas específicas para visualizar e navegar em um histórico de desenvolvimento não-linear. Um pressuposto importante em Git é que uma mudança será fundida mais freqüentemente do que está escrito. Git registra o estado atual do projeto sob a forma de uma árvore. Um novo ramo pode ser adicionado à árvore a qualquer momento e será fundido no projeto final uma vez que esteja completo.

Ramificação

Git permite que seus usuários trabalhem em uma linha que funciona paralelamente aos arquivos principais do projeto. Essas linhas são chamadas de ramificações (ou ramificações). As filiais em Git oferecem um recurso para fazer mudanças no projeto sem afetar a versão original. O ramo mestre de um lançamento sempre conterá o código de qualidade de produção. Quaisquer novas características podem ser testadas e trabalhadas em filiais, e podem ser fundidas no ramo principal.

A ramificação e a fusão podem ser feitas muito facilmente com a ajuda de alguns comandos Git. Uma única versão de um projeto pode conter n número de filiais, de acordo com as necessidades do usuário.



Ligeiro

Git armazena todos os dados do repositório central no repositório local durante a clonagem. Pode haver centenas de usuários trabalhando no mesmo projeto e, portanto, os dados no repositório central podem ser muito grandes. Pode-se estar preocupado que a clonagem de tantos dados em máquinas locais possa resultar em falhas no sistema, mas Git já cuidou desse problema. Git segue uma abordagem de compressão sem perdas que comprime os dados e os armazena no repositório local com uma pegada mínima. Sempre que necessário para estes dados, ele segue a técnica inversa e economiza muito espaço de memória.

Velocidade

Como Git armazena todos os dados relacionados a um projeto no repositório local através do processo de clonagem, é muito eficiente obter dados do repositório local ao invés de fazer o mesmo a partir do repositório remoto. Git é muito rápido e escalável em comparação com outros sistemas de controle de versões, o que resulta no manuseio eficiente de grandes projetos.

O poder de busca de um repositório local é aproximadamente 100 vezes mais rápido do que o que é possível com o servidor remoto.

Segundo um teste realizado pela Mozilla, Git é uma ordem de magnitude mais rápida, que é cerca de 10 vezes mais rápida do que outras ferramentas VCS. Isto porque Git é escrito em linguagem C, ao contrário de outros idiomas, muito semelhante à linguagem da máquina e, portanto, torna o processamento muito rápido.

Código aberto

Git é um sistema de controle de versão livre, de código aberto e distribuído, projetado para lidar com tudo, de pequenos a muito grandes projetos, com rapidez e eficiência. É chamado de código aberto porque oferece a flexibilidade de modificar seu código fonte de acordo com as necessidades do usuário. Ao contrário de outros sistemas de controle de versão que fornecem recursos pagos, tais como espaço de repositório, privacidade do código, precisão e velocidade, e assim por diante. Git é todo software de código aberto que fornece estas características de forma gratuita e até melhor que outros sistemas pagos.

Como Git é open source, ele permite que várias pessoas trabalhem no mesmo projeto ao mesmo tempo e colaborem umas com as outras de uma maneira muito fácil e eficiente. Portanto, Git é considerado o melhor sistema de controle de versão disponível atualmente.

Confiabilidade

Ao fornecer um repositório central que é clonado cada vez que um usuário realiza uma operação pull, os dados do repositório central são sempre copiados para o repositório local de cada contribuinte. Portanto, no caso de uma falha do servidor central, os dados nunca serão perdidos, pois qualquer máquina local do desenvolvedor pode facilmente recuperá-los. Uma vez que o servidor central esteja totalmente reparado, qualquer um dos múltiplos colaboradores pode recuperar os dados. Há uma probabilidade muito baixa de que os dados não estejam disponíveis com nenhum desenvolvedor porque aquele que trabalhou no projeto pela última vez terá definitivamente a última versão do projeto em sua máquina local. O mesmo é o caso no final do cliente. Se um desenvolvedor perder seus dados devido a alguma falha técnica ou qualquer uma das razões imprevistas, ele pode facilmente puxar os dados do repositório central e obter a última versão dos dados em sua máquina local. Portanto, o envio de dados para o repositório central torna o Git mais confiável para se trabalhar.

Seguro

Git mantém um registro de todos os compromissos feitos por cada colaborador com a cópia local do desenvolvedor. Um arquivo de registro é mantido e enviado para o repositório central toda vez que a operação de compromisso é realizada. Portanto, se surgir um problema, o desenvolvedor pode facilmente localizá-lo e manuseá-lo.

Git usa SHA1 para armazenar todos os troncos como objetos no Hash. Cada objeto colabora uns com os outros usando estas chaves de hash. SHA1 é um algoritmo criptográfico que converte o objeto de compromisso em um código hexadecimal de 14 dígitos. Ele ajuda a armazenar o registro de todos os compromissos assumidos por cada um dos desenvolvedores. Portanto, é fácil diagnosticar o que causou um erro no código.

Econômico

Git liberado sob a Licença Pública Geral (GPL) e, portanto, está disponível gratuitamente. Git cria um clone do repositório central na máquina local e, portanto, todas as operações são realizadas na máquina local do desenvolvedor antes de se comprometer com o repositório central. O push é realizado somente depois que a versão na máquina local estiver funcionando perfeitamente e estiver pronta para ser empurrada para o servidor central. Não há nenhuma experimentação com os arquivos no servidor central. Isto ajuda a economizar muito dinheiro em servidores caros. Todo o levantamento pesado é feito no lado do cliente e, portanto, não há necessidade de máquinas pesadas do lado do servidor.

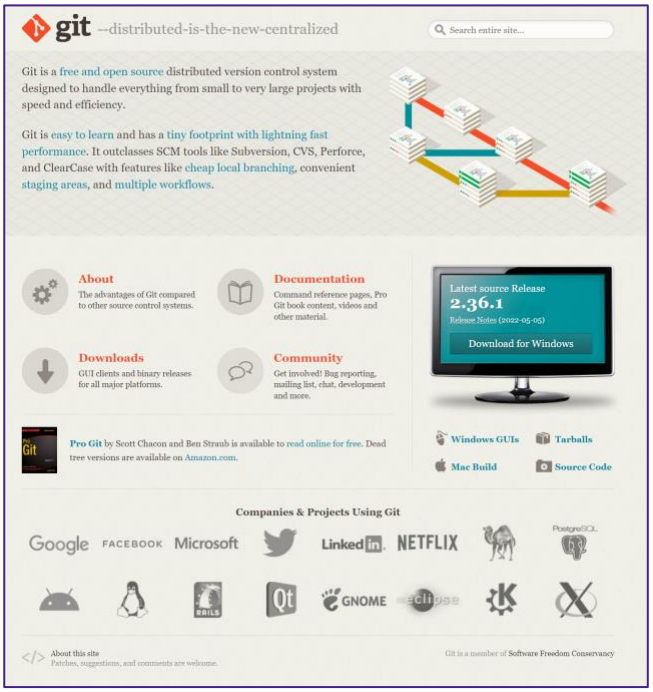
Estas características fizeram de Git o sistema de controle de versão mais confiável e amplamente utilizado de todos os tempos. O repositório central de Git é chamado GitHub. GitHub permite que todas as operações de Push and Pull sejam realizadas usando Git. Estes e outros comandos complementares serão abordados em tópicos posteriores.

Instalação de GIT

Faremos o download da versão atual do GIT a partir de seu site oficial:

https://git-scm.com/

As versões disponíveis são Mac, Windows, Linux/Unix.



Uma vez baixado e instalado, várias ferramentas GIT estão disponíveis, incluindo o console Git Bash:

Com GIT podemos trabalhar tanto a partir do console como a partir de nossa própria IDE, como atualmente, a maioria das IDEs tem integração com GIT, seja nativamente, como é o caso do Visual Studio Code, ou através de plug-ins.

O que é controle de versão?

Mencionamos anteriormente que Git é um utilitário de gerenciamento de versões. Mas o que é exatamente o controle de versão?

O controle de versão é um sistema que ajuda a rastrear e gerenciar as mudanças feitas em um arquivo ou conjunto de arquivos. Usado principalmente por engenheiros de software para acompanhar as modificações feitas no código fonte, o sistema de controle de versão permite que eles analisem todas as mudanças e as revertam sem repercussões se um erro for cometido.

Em outras palavras, o controle de versão permite que os desenvolvedores trabalhem em projetos simultaneamente. Permite-lhes fazer tantas mudanças quantas forem necessárias sem infringir ou atrasar o trabalho de seus colegas.

Se essas mudanças no código fonte atrapalharem o projeto quando forem implantadas, o GitHub facilita a sua reversão com alguns cliques, e a versão anterior do projeto será restaurada.

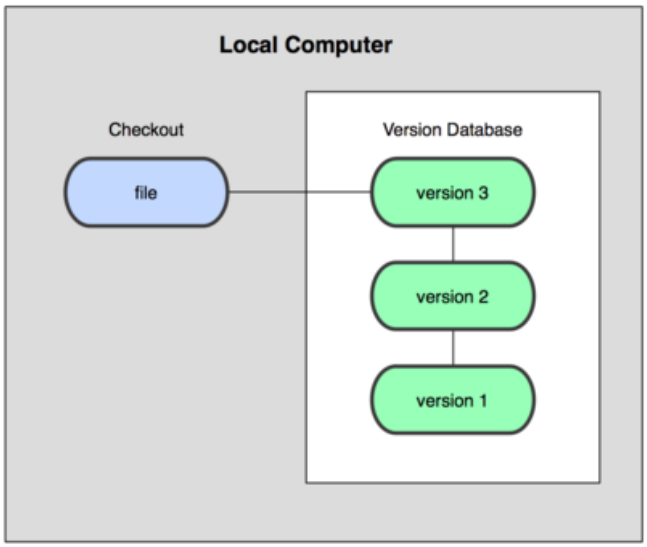
Em resumo, o controle de versão elimina o risco e o medo de cometer muitos erros. Em vez disso, proporciona a liberdade de colaborar e se desenvolver sem muitas preocupações.

Classificação dos sistemas de controle de versão

Dentro do campo de controle de versões, podemos encontrar diferentes tipos. A seguir, uma classificação das mais importantes.

Sistemas de controle de versão local

Um dos métodos mais comuns usados pelas pessoas ao realizar algum tipo de controle de versão de suas alterações consistia em copiar o arquivo a ser modificado para um diretório em seu computador local, indicando a data da modificação, para que em caso de erro soubessem qual era a última versão salva.

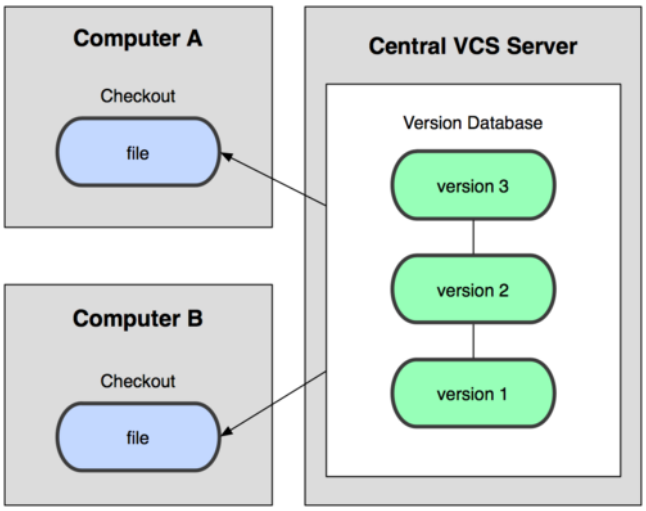


Este sistema podría valer para el desarrollo de una aplicación pequeña, pero ofrece ciertos problemas como el de no recordar dónde hemos guardado la copia o simplemente olvidarnos de hacerla.

Para hacer frente a estos problemas, los programadores desarrollaron los conocidos como *VCSs* locales, que consistían en una base de datos donde se llevaba un registro de los cambios realizados sobre los archivos.

Sistemas de controle de versão centralizada

O sistema acima mencionado pode ser útil se estivermos trabalhando sozinhos, mas um projeto geralmente envolve várias pessoas e cada uma delas é responsável por realizar uma tarefa específica. Neste caso, é necessário poder contar com um sistema colaborativo e é aqui que entram em jogo os sistemas de controle de versão centralizada.

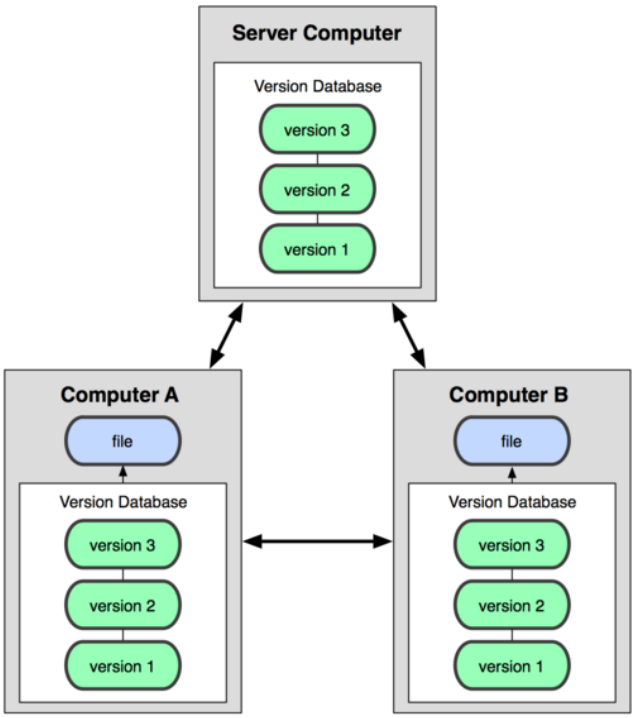


Nestes sistemas encontramos um único servidor que contém todos os arquivos versionados, e os usuários que fazem parte do projeto podem baixá-los deste servidor centralizado.

Este sistema tem um problema muito claro, que é que, usando um único servidor centralizado, no caso de um problema nesse servidor, toda a informação poderia ser perdida.

Sistemas de controle de versões distribuídas

Ao contrário do caso anterior, nestes sistemas não temos um único servidor que mantenha as informações do projeto, mas cada usuário contém uma cópia completa do projeto localmente. Desta forma, se um servidor morrer, qualquer um dos repositórios do cliente poderá ser usado para restaurar o servidor.



Git pertence a este tipo de sistema.

Github

GitHub tem muito a ver com GitHub. **GitHub é uma plataforma para gerenciar seu código e criar um ambiente colaborativo entre os desenvolvedores, usando Git como sistema de controle.** Ele tornará Git mais fácil de usar, escondendo alguns dos detalhes de configuração mais complicados.

Muitas pessoas podem pensar que Git e GitHub são a mesma coisa, pois na maioria das vezes estão intimamente relacionados, mas quando se trata disso, eles são coisas totalmente diferentes.

GitHub pode ser definido como um servidor para hospedar repositórios de projetos, adicionando funcionalidade extra para gerenciamento de projetos e códigos-fonte. Ao contrário do projeto Git, este é um serviço comercial, pois, embora tenha uma parte pública gratuita, tem a desvantagem de que todo o código que carregamos estará disponível para qualquer pessoa. Se quisermos decidir quem pode ter acesso ao nosso repositório, então seria necessário mudar para a versão paga.

O sistema baseado na web nos permite mudar arquivos online, embora isso não seja altamente recomendado, pois não teremos as vantagens de um editor, desenvolvimento e ambiente de testes. Para comunicar com o GitHub e modificar os arquivos em nosso repositório, podemos usar a linha de comando, usando o comando git e suas diretrizes de commit, pull e push. Isto será discutido em mais detalhes mais tarde.

Dentre as principais características oferecidas pela GitHub podemos destacar:

- Um Wiki que opera com Git para a manutenção das diferentes versões das páginas.
- Um sistema de rastreamento de problemas. É um sistema muito semelhante ao bilhete tradicional, onde qualquer membro da equipe ou pessoa (se nosso repositório for público) pode abrir uma consulta ou sugestão que queira fazer.
- Ferramenta de revisão de código. Permite adicionar anotações em qualquer ponto de um arquivo.
- Visualizador de filiais que permite comparar o progresso feito nas diferentes filiais de nosso repositório.

Por que GitHub é tão popular?

GitHub abriga **mais de 100 milhões de repositórios**, a maioria dos quais são projetos de código aberto. Esta estatística revela que a GitHub está entre os clientes mais populares da Git GUI e é utilizada por vários profissionais e grandes empresas, tais como a Hostinger.

Isto porque GitHub é uma plataforma de gerenciamento e organização de projetos baseada na nuvem que incorpora as características de controle de versões de Git. Isto significa que todos os usuários do GitHub podem acompanhar e gerenciar mudanças no código fonte em tempo real, enquanto têm acesso a todos os outros recursos Git disponíveis no mesmo local.

Além disso, a interface de usuário do GitHub é mais fácil de usar que a do GitHub, tornando-a acessível a pessoas com pouco ou nenhum conhecimento técnico. Isto significa que mais membros da equipe podem ser incluídos no progresso e gestão de um projeto, tornando o processo de desenvolvimento mais suave.

Diferença entre CVS e GitHub

Sistema de Versões Concorrentes (CVS)

O Sistema de Versões Concorrentes é um sistema de controle de versões funcionais desenvolvido por Dick Grune como uma série de shell scripts. Ele ajuda as equipes a permanecerem conectadas às mudanças que são medidas em um repositório quando se trabalha com software. Esta ferramenta foi usada como um sistema de controle de versão por um longo tempo.

É uma ferramenta de software confiável, mas com novos desafios, outras alternativas a tornaram cada vez menos utilizada.

- Algumas características do CVS são mostradas abaixo:
- É um dos sistemas de controle de versão confiáveis disponíveis.
- Não permite erros.
- Os roteiros são escritos em formato RCS.
- O usuário só pode armazenar arquivos diretamente no repositório.

Vantagens

- O CVS é um dos softwares de controle de versão mais confiáveis.
- As mudanças são cometidas somente quando há uma mudança completa.

Desvantagens

- As mudanças no CVS são demoradas.
- O CVS não se compromete se houver um erro no compromisso.

GitHub

GitHub é uma plataforma de hospedagem de repositório que apresenta colaboração e controle de acesso. É uma ferramenta de controle de versão para que os desenvolvedores possam processar bugs juntos para contribuir e hospedar projetos de código aberto. GitHub é projetado para desenvolvedores e usuários registrados podem usar GitHub para contribuir, mas usuários não registrados podem visualizar os repositórios.

Abaixo estão algumas características do GitHub:

- Especificar marcos e etiquetas para projetos.
- As páginas do GitHub nos permitem publicar e hospedar websites dentro do GitHub.
- Permite uma visão comparativa entre ramos.
- Permite integrações de API de terceiros para rastreamento de bugs e hospedagem em nuvem.

Vantagens

- Ajuda-nos a armazenar dados na forma de metadados.
- É usado para compartilhar o trabalho em frente ao público.

Desvantagens

- O armazenamento de arquivos grandes no GitHub torna-o lento.
- Suporta apenas o controle de versões Git.

Diferenças entre CVS e GitHub

PARÂMETRO	CVS	GITHUB
Desenvolvido por	CVS foi desenvolvido por Dick Grune.	GitHub foi desenvolvido por Chris Wanstrath, Tom Preston-Werner, PJ Hyett e Scott Chacon.
Código Aberto	É código aberto e liberado sob a Licença Pública Geral GNU.	GitHub não é código aberto.
Confirmar localização	O repositório está comprometido no servidor central.	O repositório está comprometido com o repositório local.
A clonagem do Repositório	CVS tem uma função para clonar o repositório, mas requer GIT.	GitHub permite que o usuário clone o repositório.
A navegação	CVS não permite a navegação no repositório.	GitHub permite que o usuário navegue pela usabilidade.
Compromisso do projeto	CVS deixa de cometer um erro encontrado em um nó.	GitHub permite o commit no repositório e o desenvolvedor corrige o bug.

Mensagens em objetos

Uma mensagem em um objeto é a ação de fazer uma chamada de método. Por exemplo, quando dizemos a um objeto de carro para dar partida, estamos passando-lhe a mensagem "start".

Para enviar mensagens aos objetos usamos o operador de ponto, seguido do método que queremos invocar e parênteses, como em chamadas de função.

miCoche.ponteEnMarcha()

Neste exemplo, passamos a mensagem "ponteEnMarcha" para o objeto "miCoche".

Após o nome do método que queremos invocar, temos que colocar parênteses, como quando invocamos uma função. Dentro dos parênteses estariam os parâmetros, se o método os exigir.

4 Princípios da Programação Orientada a Objetos

Encapsulamento

O encapsulamento contém todas as informações importantes de um objeto dentro do objeto e só expõe informações selecionadas para o mundo exterior.

Esta propriedade assegura que a informação de um objeto seja escondida do mundo exterior, agrupando em uma Classe as características ou atributos que têm acesso privado, e os comportamentos ou métodos que têm acesso público.

O encapsulamento de cada objeto é responsável por suas próprias informações e por seu próprio estado. A única maneira de modificar isto é através dos próprios métodos do objeto. Portanto, os atributos internos de um objeto devem ser inacessíveis do exterior, e só podem ser modificados pela chamada das funções correspondentes. Isto é para manter o estado a salvo de usos indevidos ou inesperados.

Usamos um carro como exemplo para explicar o encapsulamento. O carro compartilha informações públicas através das luzes de freio ou sinais de giro para indicar curvas (interface pública). Em contraste, temos a interface interna, que seria o mecanismo de propulsão do carro, que está escondido sob o capô. Ao dirigir um carro, é necessário indicar seus movimentos a outros motoristas, mas não expor dados particulares sobre o tipo de combustível ou temperatura do motor, pois trata-se de muitos dados, o que confundiria outros motoristas.

Abstração

Abstração é quando o usuário interage apenas com atributos e métodos selecionados de um objeto, utilizando ferramentas simplificadas de alto nível para acessar um objeto complexo.

Na programação orientada a objetos, os programas são freqüentemente muito grandes e os objetos se comunicam muito uns com os outros. O conceito de abstração facilita a manutenção de grandes códigos, onde diferentes mudanças podem ocorrer ao longo do tempo.

Assim, a abstração se baseia no uso de coisas simples para representar a complexidade. Objetos e classes representam o código subjacente, escondendo detalhes complexos do usuário. É, portanto, uma extensão do encapsulamento. Continuando com o exemplo do carro, você não precisa conhecer todos os detalhes de como o motor funciona para poder dirigi-lo.

Inheritance

A herança define relações hierárquicas entre classes, para que atributos e métodos comuns possam ser reaproveitados. As classes dos pais estendem atributos e comportamentos às classes de crianças. Definindo atributos e comportamentos básicos em uma classe, podem ser criadas classes infantis, ampliando assim a funcionalidade da classe dos pais e adicionando atributos e comportamentos adicionais.

Voltando ao exemplo dos animais, uma única classe de animais pode ser usada e um atributo de tipo animal pode ser adicionado que especifica o tipo de animal. Diferentes tipos de animais precisarão de métodos diferentes, por exemplo, as aves precisam ser capazes de pôr ovos e os peixes precisam ser capazes de nadar. Mesmo quando os animais têm um método comum, como a movimentação, a implementação precisaria de muitas declarações "se" para garantir o comportamento correto da movimentação. Por exemplo, os sapos saltam, enquanto as cobras deslizam. O princípio da herança nos permite resolver este problema.

Polimorfismo

O polimorfismo consiste em projetar objetos para compartilhar comportamentos, o que nos permite processar objetos de diferentes maneiras. É a capacidade de apresentar a mesma interface para diferentes formulários ou tipos de dados subjacentes. Ao usar a herança, os objetos podem anular comportamentos primários compartilhados com comportamentos secundários específicos. O polimorfismo permite que o mesmo método execute comportamentos diferentes de duas maneiras: sobreposição de métodos e sobrecarga de métodos.

Muitas coisas são construídas em torno desses princípios de programação orientada a objetos.

Por exemplo, os princípios SOLID, ou padrões de design, que são receitas aplicadas a problemas recorrentes que foram encontrados e se repetem em vários projetos.

Benefícios da Programação Orientada a Objetos

- Reutilização do código.
- Transforma coisas complexas em estruturas simples e reprodutíveis.
- Evita a duplicação de código.
- Permite o trabalho em equipe graças ao encapsulamento, pois minimiza a possibilidade de duplicar funções quando várias pessoas trabalham no mesmo objeto ao mesmo tempo.
- Como a classe é bem estruturada, permite a correção de erros em vários lugares do código.
- Ele protege as informações através de encapsulamento, pois os dados do objeto só podem ser acessados através de propriedades e métodos privados.
- A abstração nos permite construir sistemas mais complexos de uma forma mais simples e organizada.

Neste curso veremos o conceito de OOP aplicado às linguagens de programação Javascript e Python. No caso do Javascript, o OOP não tem muito impacto, mas Python é uma linguagem totalmente orientada a objetos. Em Python, todos os elementos são considerados como objetos, com suas propriedades e métodos. Portanto, o uso do OOP em Python será constante. Iremos mais fundo quando chegarmos a essa parte do programa de estudos.