Ruben Antunes 08/01/2021

1. O que é o Git?

É um sistema de controle de versões que é usado para desenvolvimento de software, mas que pode ser usado para registrar o histórico de edições de qualquer tipo de arquivo (Exemplo: alguns livros digitais são disponibilizados no GitHub e escrito aos poucos publicamente). O Git foi inicialmente projetado e desenvolvido por Linus Torvalds para o desenvolvimento do kernel Linux, mas foi adotado por muitos outros projetos.

Cada diretório de trabalho do Git é um repositório com um histórico completo com acompanhamento e revisões, não dependente de acesso a uma rede ou a um servidor central. O Git também facilita a reprodutibilidade científica em uma ampla gama de disciplinas, da ecologia à bioinformática, arqueologia à zoologia.

2. Alternativas de ferramentas de controlo de versões:

CVS, Subversion, SVN, TFS eMercurial.

3. Quais as vantagens de se utilizar um sistema de controle de versão como o Git?

Desempenho, Segurança e Flexibilidade:

De longe, o sistema de controle de versão moderno mais usado no mundo hoje é o Git. O Git é um projeto de código aberto maduro e com manutenção ativa desenvolvido em 2005 por Linus Torvalds, o famoso criador do kernel do sistema operacional Linux. Um número impressionante de projetos de software depende do Git para controle de versão, incluindo projetos comerciais e de código-fonte aberto. Os desenvolvedores que trabalharam com o Git estão bem representados no pool de talentos de desenvolvimento de software disponíveis e funcionam bem numa ampla variedade de sistemas operativos e IDEs (Ambientes de Desenvolvimento Integrado).

Tendo uma arquitetura distribuída, o Git é um exemplo de DVCS (portanto, Sistema de Controle de Versão Distribuído). Em vez de ter apenas um único local para o histórico completo da versão do software, como é comum em sistemas de controle de versão outrora populares como CVS ou Subversion (também conhecido como SVN), no Git, a cópia de trabalho de todo desenvolvedor do código também é um repositório que pode conter o histórico completo de todas as alterações. Além de ser distribuído, o Git foi projetado com desempenho, segurança e flexibilidade em mente.

Desempenho

As características brutas de desempenho do Git são muito fortes quando comparadas a muitas alternativas. Fazer o commit de novas alterações, branches, mesclagem e comparação de versões anteriores – tudo é otimizado para desempenho. Os algoritmos implementados no Git aproveitam o conhecimento profundo sobre atributos comuns de

árvores de arquivos de código-fonte reais, como costumam ser modificados ao longo do tempo e quais são os padrões de acesso.

Diferente de alguns softwares de controle de versão, o Git não se deixa enganar pelos nomes dos arquivos ao determinar qual deve ser o armazenamento e o histórico de versões da árvore de arquivos. Em vez disso, o Git concentra-se no conteúdo do arquivo. Afinal, os arquivos de código-fonte são renomeados, divididos e reorganizados com frequência. O formato do objeto dos arquivos de repositório do Git usa uma combinação de codificação delta (armazenamento de diferenças de conteúdo) e compactação e armazena com clareza o conteúdo do diretório e os objetos de metadados da versão.

Segurança

O Git foi projetado com a integridade do código-fonte gerido como uma prioridade. O conteúdo dos arquivos, bem como os verdadeiros relacionamentos entre arquivos e diretórios, versões, tags e commits, todos esses objetos no repositório do Git são protegidos com um algoritmo de hash de criptografia seguro chamado SHA1. Isso protege o código e o histórico de alterações contra alterações acidentais e maliciosas e garante que o histórico tenha rastreabilidade total.

Flexibilidade

Um dos principais objetivos de design do Git é a flexibilidade. O Git é flexível em vários aspetos: suporte a vários tipos de fluxos de trabalho de desenvolvimento não lineares, em eficiência em projetos pequenos e grandes e em compatibilidade com muitos sistemas e protocolos existentes.

O Git foi projetado para tratar os branches e tags como cidadãos de primeira classe (ao contrário do SVN) e operações que afetam branches e tags (como mesclagem ou reversão) também são armazenadas como parte do histórico de alterações. Nem todos os sistemas de controle de versão apresentam esse nível de rastreamento.

Hoje, o Git é a melhor escolha para a maioria das equipes de software. Embora cada equipe seja diferente e deva fazer a própria análise, aqui estão os principais motivos pelos quais o controle de versão com Git é preferido em vez de alternativas:

O Git tem a funcionalidade, desempenho, segurança e flexibilidade que a maioria das equipas e desenvolvedores individuais precisa. Esses atributos do Git são explicados acima. Nas comparações lado a lado com a maioria das outras alternativas, muitas equipes acham que o Git é muito favorável.

O Git é um projeto de código aberto muito bem suportado, com mais de uma década de administração sólida. Os mantenedores do projeto mostraram um julgamento equilibrado e uma abordagem madura para atender às necessidades de longo prazo dos usuários, com lançamentos regulares que melhoram a usabilidade e a funcionalidade. É fácil examinar a qualidade do software de código aberto, e inúmeras empresas dependem muito dessa qualidade.

O Git tem excelente suporte da comunidade e uma vasta base de usuários. A documentação é excelente e abundante, incluindo livros, tutoriais e sites dedicados. Existem também podcasts e tutoriais em vídeo. O código aberto reduz o custo para desenvolvedores amadores, pois eles podem usar o Git sem pagar uma taxa. Para uso em projetos de código aberto, o Git é sem dúvida o sucessor das gerações anteriores de sistemas bem-sucedidos de controle de versão de código aberto, SVN e CVS.

4. Qual o comando para indicar que uma pasta deve ser gerenciado pelo Git?

Deve-se iniciar o git dentro da pasta, e o comando para ativar a gestão pelo Git é git init

5. Ao adicionar/editar arquivos dentro de uma pasta gerenciada pelo Git, quais os comandos necessários para adicionar estas mudanças, de fato, ao Git?

git status (verifica se há ficheiros por adicionar)

git add . (para adicionar todos os ficheiros ao git, ou git add file.py por exemplo para adicionar apenas um ficheiro)

git commit -m "descrição da alteração" (grava as alterações no repositório com uma descrição para orientação do programador)

6. Qual a função do comando "checkout"?

Faz a transição entre versões (branches) do repositório do projeto. Atualiza os ficheiros na árvore de trabalho para corresponder à versão do índice ou da árvore especificada.

7. Pesquise e explique o que é uma "branch"?

Uma ramificação no git é um ponteiro para as alterações feitas nos arquivos do projeto. É útil em situações nas quais se pretende adicionar um novo recurso ou corrigir um erro, gerando uma nova ramificação garantindo que o código instável não seja mesclado nos arquivos do projeto principal. Depois de concluir a atualização dos códigos da ramificação, pode-se mesclar a ramificação com a principal, geralmente chamada de master.

Cria-se sempre um branch a partir de um branch existente. Normalmente, cria-se um novo branch a partir do branch-padrão do repositório. Assim poderá trabalhar-se nesse novo branch isolado das mudanças que outras pessoas estão fazendo no repositório.

8. Diversas plataformas existem no mercado para suportar o uso de Git via Web:

1. QGit

QGit é um Git GUI gratuito para Linux que pode mostrar de maneira gráfica as diferentes ramificações dos projetos, assim como ver conteúdos de patches e mudanças nos arquivos. Com essa ferramenta, você pode ver árvores de arquivos, históricos de registros, revisões e diffs.

Com o QGit, também há a possibilidade de comparar arquivos e de alterar visualmente o conteúdo modificado. Outras possibilidades envolvem aplicar ou formatar séries de patches de commits selecionados e mover esses commits entre duas instâncias do próprio QGit.

Você pode usar a mesma semântica dos commits do GIt para criar novos patches e implementar comandos StGit comuns. Sequências de scripts e de comandos podem ser conectadas a uma ação customizada.

2. Gitg

A interface de user do Gitg é simples e direta de se utilizar. Ela pode abrir repositórios Git existentes salvos no seu computador. Você pode baixar o software gratuitamente e ele tem uma licença GPLv2. Repositórios remotos também podem ser visualizados usando o Gitg.

O Gitg permite que você execute operações comuns do Git, navegue através de commits e faça a pré-visualização de arquivos. Você pode ver mensagens de commit e também commits que não estejam rastreados ou com etapas definidas através de uma ferramenta de visualização.

O lado negativo dessa ferramenta é que arquivos muito grandes tendem a carregar de maneira mais lenta, além de que ela não pode exibir o histórico de um projeto.

3. Git Force

Git Force é uma ferramenta de front-end visual que roda tanto no Linux quanto no Windows. O melhor de tudo: o seu download é gratuito. Esse software é bem útil para iniciantes, pois a interface é intuitiva e tem uma ferramenta de arrastar e soltar. Ele também pode ser usado de maneira isolada, sem necessitar do uso de uma ferramenta Git de linha de comando.

Você pode criar múltiplos repositórios e ramificações Git e gerenciar todos usando o Git Force. A ferramenta é capaz de suportar um ou mais repositórios remotos e também pode rapidamente escanear os repositórios locais.

O trabalho que você faz num repositório Git será melhor reconhecido pelo Git Force na primeira atualização. Contudo, ele só funciona com os comandos Git mais comuns. Por causa disso, ele não mantém nenhuma informação detalhada de estado.

4. Sourcetree

Sourcetree é um cliente Git GUI gratuito que funciona tanto no Windows quanto no Mac. Essa ferramenta é simples de usar mas também é muito poderosa. Isso a torna perfeita tanto para iniciantes quanto para usuários avançados. Sua interface limpa e elegante permite que você navegue sem esforços e de uma maneira agradável.

Essa é uma interface gráfica de utilizador cheia de recursos, tornando seus projetos de Git mais fáceis e eficientes. Ela suporta arquivos grandes de Git e deixa que você os visualize usando diagramas de ramificação detalhados. Isso torna mais fácil para você e a sua equipe gerenciarem o seu progresso.

A pesquisa de commits locais permite que você encontre mudanças de arquivos, commits e ramificações. Enquanto isso, o gerente de repositórios remotos permite que você pesquise e clone repositórios remotos dentro do próprio Sourcetree. Ainda é possível obter commits claros e limpos através da ferramenta “interactive rebase”.

5. GitHub

Se o seu repositório remoto está no GitHub, então essa ferramenta será a mais útil para você. O software é basicamente uma extensão do seu fluxo de trabalho no GitHub. Basta realizar o login usando a sua conta do GitHub que você já pode começar a trabalhar nos seus repositórios.

O GitHub desktop é um cliente Git GUI gratuito e de código aberto. Ele tem uma interface intuitiva que permite gerenciar os códigos sem a necessidade de digitar comandos. Você pode criar novos ou adicionar repositórios locais e executar operações Git com facilidade.

O processo de criar ramificações e trocar para aquelas existentes não traz nenhuma complicação — o mesmo pode ser dito para a fusão de códigos com a ramificação mestre. Ademais, você pode rastrear as suas mudanças com o GitHub Desktop.