Introdução ao GitHub

Peter Bell e Brent Beer

Novatec

Authorized Portuguese translation of the English edition of *Introducing GitHub*, ISBN 9781491949740

Tradução em português autorizada da edição em inglês da obra *Introducing GitHub*, ISBN 9781491949740

Editor: Rubens Prates
Tradução: Lúcia A. Kinoshita
Revisão gramatical: Marta Almeida de Sá
Assistente editorial: Priscila A. Yoshimatsu
Editoração eletrônica: Carolina Kuwabata

ISBN: 978-85-7522-414-4

Histórico de impressões:

Janeiro/2015 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Email: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

CAPÍTULO 1

Introdução

Neste capítulo, começaremos apresentando o Git e o GitHub. O que são, qual a diferença entre eles e por que você iria querer usá-los? Em seguida, apresentaremos outros termos comuns que você ouvirá sendo mencionados com frequência quando as pessoas estiverem discutindo sobre o GitHub. Dessa maneira, você poderá entender e participar das discussões sobre seus projetos mais facilmente.

O que é o Git?

O *Git* é um sistema de controle de versões. Um *sistema de controle de versões* é um software concebido para manter um registro das alterações feitas em arquivos ao longo do tempo. Mais especificamente, o Git é um sistema de controle de versões *distribuído*, o que significa que todos que estiverem trabalhando em um projeto no Git terão uma cópia de todo o histórico do projeto, e não apenas do estado atual dos arquivos.

O que é o GitHub?

O *GitHub* é um site em que você pode carregar uma cópia de seu repositório Git. Ele permite que você colabore muito mais facilmente com outras pessoas em um projeto. Isso é feito por meio da disponibilização de um local centralizado para compartilhar o repositório, uma interface web para visualizá-lo e recursos como *forking*, *pull requests*, *issues* e *wikis*, que permitem especificar, discutir e revisar alterações junto à sua equipe de maneira eficiente.

Por que usar o Git?

Mesmo que você esteja trabalhando sozinho, se estiver editando arquivos texto, há inúmeras vantagens em usar o Git. Essas vantagens incluem as seguintes:

Capacidade de desfazer alterações

Se você cometer um erro, será possível retornar a um ponto anterior no tempo e recuperar uma versão mais antiga de seu trabalho.

Um histórico completo de todas as alterações

Se, algum dia, você quiser saber como era o seu projeto um dia antes, uma semana, um mês ou um ano atrás, será possível efetuar o *check out* de uma versão anterior do projeto para ver exatamente qual era o estado dos arquivos naquela época.

Documentação dos motivos pelos quais as alterações foram feitas

Com frequência, é difícil se lembrar do *motivo* pelo qual uma alteração foi feita. Com as *mensagens de commit* do Git, é fácil documentar a razão de estar fazendo uma alteração para futuras referências.

Confiança para alterar qualquer trabalho

Pelo fato de ser fácil recuperar uma versão anterior de seu projeto, você pode se sentir confiante ao fazer qualquer alteração desejada. Se essas alterações não funcionarem, é sempre possível retornar a uma versão anterior de seu trabalho.

Várias linhas de história

Você pode criar diferentes *branches* (ramos) de história para experimentar diferentes alterações em seu conteúdo ou desenvolver funcionalidades diferentes de modo independente. Você poderá então fazer o *merge* desses trabalhos (incorporá-los) de volta à história principal do projeto (o *branch master*, ou ramo principal) depois que eles estiverem prontos ou poderá apagá-los se eles acabarem não funcionando.

Ao trabalhar em uma equipe, você terá um conjunto mais amplo ainda de vantagens se usar o Git para registrar as suas alterações. Algumas das principais vantagens do Git ao trabalhar com uma equipe são:

Capacidade de resolver conflitos

Com o Git, várias pessoas podem trabalhar no mesmo arquivo ao mesmo tempo. Normalmente, o Git será capaz de combinar as alterações automaticamente. Se não puder, ele mostrará quais são os conflitos e fará com que seja fácil para você resolvê-los.

Linhas independentes de história

Diferentes pessoas no projeto podem trabalhar em *branches* diferentes, permitindo implementar funcionalidades distintas de forma independente e combiná-las quando estiverem concluídas.

Por que usar o GitHub?

O GitHub é muito mais que apenas um local para armazenar seus repositórios Git. Ele proporciona diversas vantagens adicionais, incluindo a capacidade de fazer o seguinte:

Documentar requisitos

Ao usar *Issues*, é possível documentar bugs ou especificar novas funcionalidades que você queira que a sua equipe desenvolva.

Colaborar com linhas independentes de história

Ao usar branches e *pull requests*, você poderá colaborar em branches ou funcionalidades diferentes.

Revisar um trabalho em progresso

Ao observar uma lista de pull requests, você poderá ver todas as diferentes funcionalidades em que as pessoas estão trabalhando no momento e, ao clicar em qualquer pull request específico, você poderá ver as últimas alterações bem como todas as discussões em torno delas.

Ver o progresso da equipe

Dar uma olhada no *Pulse* (pulsação) ou no *histórico de commits* permitirá ver em que a equipe está trabalhando.

Conceitos fundamentais

Há diversos conceitos fundamentais os quais você deverá entender para trabalhar de modo eficiente com o Git e o GitHub. A seguir, temos uma lista de alguns dos termos mais comuns, com uma breve descrição de cada um deles e um exemplo de como podem ser usados em uma conversa.

Commit

Sempre que suas alterações forem salvas em um ou mais arquivos para serem guardadas no histórico do Git, você estará realizando um novo commit. *Exemplo de uso: "Vamos fazer o commit dessas alterações e enviá-las ao GitHub."*

Mensagem de commit

Sempre que você fizer um commit, será necessário fornecer uma mensagem que descreva o *motivo* da alteração. Essa mensagem de commit terá um valor inestimável no futuro, quando você tentar entender por que uma determinada alteração foi implementada. *Exemplo de uso: "Não se esqueça de incluir o comentário de Susan sobre as novas diretrizes do SEC na mensagem de commit."*

Branch (Ramo)

É uma série independente de commits laterais que pode ser usada para fazer uma experiência ou para criar uma nova funcionalidade. *Exemplo de uso: "Vamos criar um branch para implementar a nova funcionalidade de pesquisa."*

Branch master (Ramo principal)

Sempre que um novo projeto Git for criado, um branch default chamado *master* será criado. Esse é o branch em que o seu trabalho deverá ser incluído em algum momento, quando estiver pronto para ser disponibilizado para produção. *Exemplo de uso: "Lembre-se de jamais fazer commits diretamente no master."*

Branch de feature (Branch de funcionalidade ou de tópico)

Sempre que uma nova funcionalidade for implementada, você deverá criar um branch para trabalhar com ela. Esse branch se chama *branch de feature* (branch de funcionalidade). *Exemplo de uso: "Temos muitos branches de feature. Vamos focar em finalizar um ou dois deles e disponibilizá-los para produção."*

Branch de release

Se você tiver um processo manual de QA ou precisa dar suporte a versões antigas de seu software para seus clientes, pode ser que seja necessário ter um branch de release como um local para efetuar quaisquer correções ou atualizações necessárias. Não há nenhuma diferença técnica entre um branch de feature e um branch de release, porém a distinção será útil quando você falar de um projeto com a sua equipe. *Exemplo de uso: "Precisamos corrigir o bug de segurança em todos os branches de release em que damos suporte."*

Merge (Incorporar)

Esse é a maneira de tomar o trabalho concluído em um branch e incorporá-lo em outro. O mais comum será fazer o merge de um branch de feature com o branch master. *Exemplo de uso: "Ótimo trabalho na funcionalidade 'minha conta'. Você poderia fazer o merge dele no master para que possamos disponibilizá-lo para a produção?".*

Tag

Uma referência a um commit histórico específico. Com muita frequência, é usada para documentar versões de produção para que você saiba exatamente quais versões do código foram entregues à produção e quando. *Exemplo de uso: "Vamos aplicar uma tag nessa versão e disponibilizá-la para produção."*

Check out

Consiste em acessar uma versão diferente da história do projeto e ver os arquivos como estavam naquele instante no tempo. O mais comum será efetuar o check out de um branch para ver todo o trabalho feito nele, porém é possível fazer check out de qualquer commit. *Exemplo de uso: "Você poderia fazer o check out da tag da última versão? Há um bug na produção que eu preciso que você reproduza e corrija."*

Pull request

Originalmente, um pull request era usado para pedir a outra pessoa que revisasse o seu trabalho em um branch e, em seguida, fizesse um merge desse com o master. Atualmente, os pull requests normalmente são usados mais cedo no processo, para iniciar uma discussão sobre uma possível funcionalidade. *Exemplo de uso: "Crie um pull request para a nova funcionalidade de votação para que possamos ver o que o restante da equipe acha dela."*

Issue

O GitHub tem um recurso chamado Issue que pode ser usado para discutir funcionalidades, monitorar bugs ou para ambas as atividades. *Exemplo de uso: "Você tem razão, o login não funciona em um iPhone. Você poderia criar uma issue no GitHub documentando os passos para reproduzir o bug?".*

Wiki

Desenvolvido originalmente por Ward Cunningham, as wikis são uma maneira leve de criar páginas web com links simples entre elas. Os projetos GitHub geralmente usam wikis para documentação. *Exemplo de uso: "Você poderia adicionar uma página na wiki para explicar como configurar o projeto de modo que ele execute em vários servidores?".*

Clone

Com frequência, você irá querer efetuar o download de uma cópia de um projeto do GitHub para poder trabalhar nele localmente. O processo de copiar o repositório para o seu computador chama-se *clonagem*. *Exemplo de uso: "Você poderia clonar o repositório, corrigir o bug e, em seguida, enviar a correção de volta ao GitHub mais tarde?".*

Fork (Bifurcar)

Às vezes, você não tem a permissão necessária para fazer alterações diretamente em um projeto. Pode ser que seja um projeto de código aberto, escrito por pessoas que você não conheça, ou um projeto desenvolvido por outro grupo em sua empresa, com o qual você não trabalhe com muita frequência. Se quiser submeter alterações a um projeto desse tipo, inicialmente você deverá criar uma cópia do projeto com a *sua* conta de usuário no GitHub. Esse processo é chamado de *forking* do repositório. Em seguida, você poderá cloná-lo, fazer as alterações e submetê-las de volta ao projeto original usando um pull request. *Exemplo de uso: "Adoraria ver como você reescreveria a cópia de marketing da página inicial. Faça um fork no repositório e submeta um pull request com as alterações que você propõe."*

Caso toda a terminologia pareça opressiva no início, não se preocupe. Depois que você começar a trabalhar com alguns projetos de verdade, tudo fará muito mais sentido! No próximo capítulo, daremos uma olhada nos vários elementos de um projeto GitHub e veremos como você poderá usá-los para ter uma ideia do progresso de um projeto.