Nota de Desenvolvimento 02 (nd02)
Utilizando o Pycharm com o Github

Sérgio Rivero

19 de março de 2020

Sumário

1 Introdução **2**

2 O que é controle de versões e o que é o Git **2**

3 Download e Instalação do Git **6**

4 Aprendendo a usar Branching e Merging **6**

5 Usando o Git no PyCharm **6**
5.1 Conectando ao GitHub 6
5.2 Clonando o Código da Aplicação (EcoSim) 6
5.3 Implementando código 6
5.4 Commit e Merge 6

1 Introdução

Olá pessoal. Aqui estamos na nossa nota de desenvolvimento 02.

Discutiremos nesta nota a utilidade e a forma de usar de umas das principais ferramentas de controle de versão de *software* que existe por aí, o **Git**.

Há muitas formas de utilização do **Git** e uma grande diversidade ferramentas para seu uso, desde o nosso velho e conhecido terminal (shell) até as próprias IDEs (como o pycharm) que tornam o processo de controle de versões relativamente transparente para o desenvolvedor.

Neste documento utilizarei principalmente o material que está no livro do git de Chacon and Straub (2014) disponível aqui. Também iremos ver um pouco de detalhe sobre o uso do github, material que pode ser encontrado aqui.

2 O que é controle de versões e o que é o Git

Um dos maiores desafios da produção de software, quando se trabalha em equipe é o controle de versões. Muitas vezes alterações em um componente do software tem impacto em outro. Além disso, há diferentes alterações que, eventualmente são feitas em partes diferentes do mesmo programa por mais de uma pessoa. Manter a consistência do software e cobrir as possibilidades de alteração e de voltar a um estado anterior do programa, caso ocorra algum erro, é uma tarefa complexa que, felizmente, pode ser feita utilizando-se um sistema de controle de versões (VCS).

Chacon and Straub (2014, p. 9) colocaram sobre controle de versões que:

> *"... é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo, para que você possa recuperar versões específicas mais tarde."*

Há muitos *VCS* por aí, mas o Git acabou se tornando um principal sistema para este tipo de tarefa no mundo. Isso se deve principalmente ao fato de que seu projeto foi especificado de maneira a atender a demanda de desenvolvedores de software em um ambiente distribuído, aberto, para *software* complexo e em um projeto de larga escala (no caso, este projeto era o kernel do Linux). Isto produziu um VCS fácil de usar e eficaz em projetos com muitos *ramos*[1] (figura 1). Um ramo é um subcomponente de um projeto ou um projeto derivado de um projeto principal que pode ou não ser reincorporado no código principal (merge) depois de várias alterações (staged changes) incorporadas no ramo (commit).

[1] Vamos usar o termo inglês **branch** traduzido como "ramo".

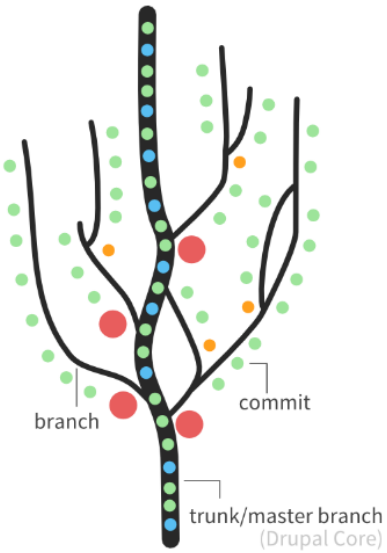

Figura 1: Um exemplo de git branching o projeto principal é o tronco maior e os ramos são subcomponentes do projeto ou projetos derivados desenvolvidos em paralelo(Burke, 2015)

.

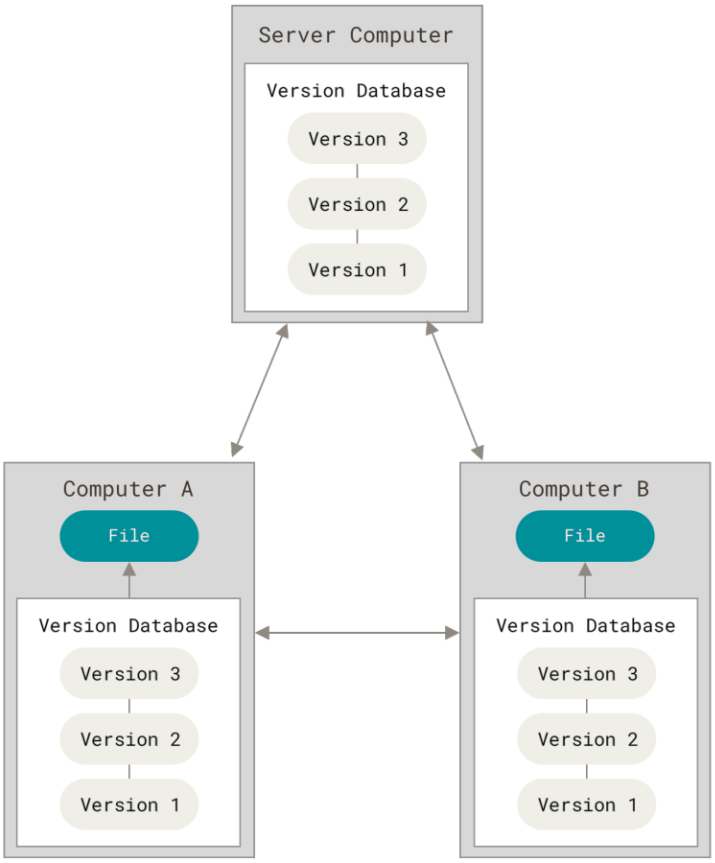

Figura 2: Um sistema de controle de versões distribuído (Chacon and Straub, 2014, p. 12)

O Git foi criado em 2005, em um projeto liderado por Linus Tovalds, para atender as necessidades citadas acima. Git, é um VCS distribuído, isto é, as versões do software ficam armazenadas, não apenas num servidor central, mas também em todos os computadores que têm versões do software. O banco de dados registra as alterações localmente e permite a atualização dos outros bancos de dados dos desenvolvedores do software, registrando as diferenças entre os programas e possibilitando inclusive o retorno a um estágio anterior do desenvolvimento, na ocorrência de um eventual erro (figura 2).

O Git funciona fazendo *"snapshots"* do código, guardando um status do desenvolvimento da aplicação a cada *checkin* no VCS. A maioria das operações de atualização do Git é local, apenas a operação de atualização em outros servidores é remota (pull-push, por exemplo). A integridade do código é garantida por um mecanismo de checksum que gera uma string de 40 caracteres a partir do código existente naquela versão do código.

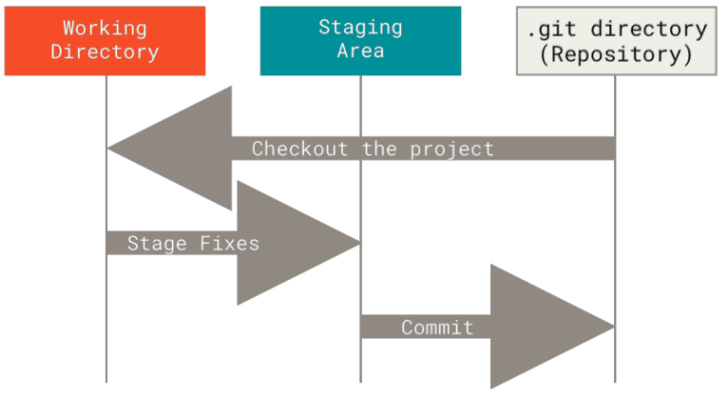

Figura 3: Estrutura do trabalho no Git (Chacon and Straub, 2014, p. 16)

O Git tem apenas 3 estdos para os seus arquivos compreender estes estados permite entender claramente o funcionamento do sistema. Estes estados são:

- **Modificado** (*modified*) - Onde você mudou localmente algum arquivos da sua aplicação mas ainda não informou o Git desta modificação (não fez o commit)
- **Encenado** (*staged*) - Neste estágio você modificou o arquivo e informou sistema que o arquivo foi modificado (com um add, por exemplo ou com os arquivos que o git automaticamente observa). Estes arquivos serão incluídos no próximo commit.
- **"Cometido"** (*commited*) - Os seus dados foram armazenados com segurança no seu banco de dados local (que está na pasta .git dentro da árvore de diretório da sua aplicação)

O fluxo de trabalho básico do Git é, em geral (Chacon and Straub, 2014, p.16):

1. Após o checkout do seu projeto você modifica arquivos no seu diretório de trabalho
2. Você "encena" as mudanças feitas, adicionando seletivamente os aquivos que quer incluir no próximo commit.
3. Você faz um commit armazenando uma "foto" da sua aplicação no diretório local do Git.

3 Download e Instalação do Git

Você pode baixar os programas necessários para executar o Git no *Windows* aqui nesta página ou nesta aqui. Pode ser utilizado tanto em um terminal *bash* para o windows quanto numa interface gráfica, quanto no próprio PyCharm. Para um tutorial sobre o uso do Git com PyCharm, vocês podem ir aqui.

Para o linux é só baixar o Git diretamente do repositório de seu sabor linux via comando de atualização de sofware (apt nos sistemas Debian)

```
sudo apt install git-all
```

4 Aprendendo a usar Branching e Merging

5 Usando o Git no PyCharm

5.1 Conectando ao GitHub

5.2 Clonando o Código da Aplicação (EcoSim)

5.3 Implementando código

5.4 Commit e Merge

Referências

Burke, C. (2015). Introduction do git.

Chacon, S. and Straub, B. (2014). *Pro git*. Apress.