SCC0504 - Programação Orientada a Objetos

Git e JSON

Prof.: Leonardo Tórtoro Pereira

leonardop@usp.br

Git

O Que é Git?

- Sistema de controle de versões (versionamento)
- Rastreia mudanças em arquivos
- Coordena o trabalho de vários usuários nestes arquivos
- Existem diversos repositórios diferentes
  - GitHub, BitBucket, GitLab, etc.
- Existem diversas interfaces e IDEs com Git integrado que ajudam (bastante) a mexer nos repositórios
  - SourceTree, GitKraken, VisualStudio, etc.

O Que é Git?

- Ótimo para trabalhar em equipe!
- Permite criar *branches* (ramos) em que você pode testar uma ideia de código, programar o quanto quiser e, se der tudo errado, só descartar e voltar para o código fora desse ramo, com tudo funcionando como antes!
- Caso uma alteração nova dê problema, é possível voltar ao estado (*commit*) anterior em que tudo estava certo

O Que é Git?

- Se duas pessoas editaram o mesmo código, o sistema fornece a opção de juntar (*merge*) os dois códigos
  - Isso normalmente é feito automaticamente, juntando tudo dos dois sem problemas!
  - Caso o algoritmo de *merge* não encontre uma solução perfeita, o usuário tem a opção de escolher o que manter de cada versão do código.

O Que é Git?

- Uma explicação mais detalhadas dos benefícios do controle de versão:

Benefícios do Controle de Versão [1]

- ➔ Um histórico de alterações completo em longo prazo de cada arquivo. Isso significa cada alteração feita por muitos indivíduos no passar dos anos.
- ➔ As alterações incluem a criação e a exclusão de arquivos, bem como edições no conteúdo.
- ➔ Esse histórico também deve incluir o autor, a data e notas escritas sobre o propósito de cada alteração.
- ➔ Ter o histórico completo permite voltar a versões anteriores para ajudar na análise da causa de bugs e é crucial quando se precisa corrigir problemas em versões mais antigas do software.

Benefícios do Controle de Versão [1]

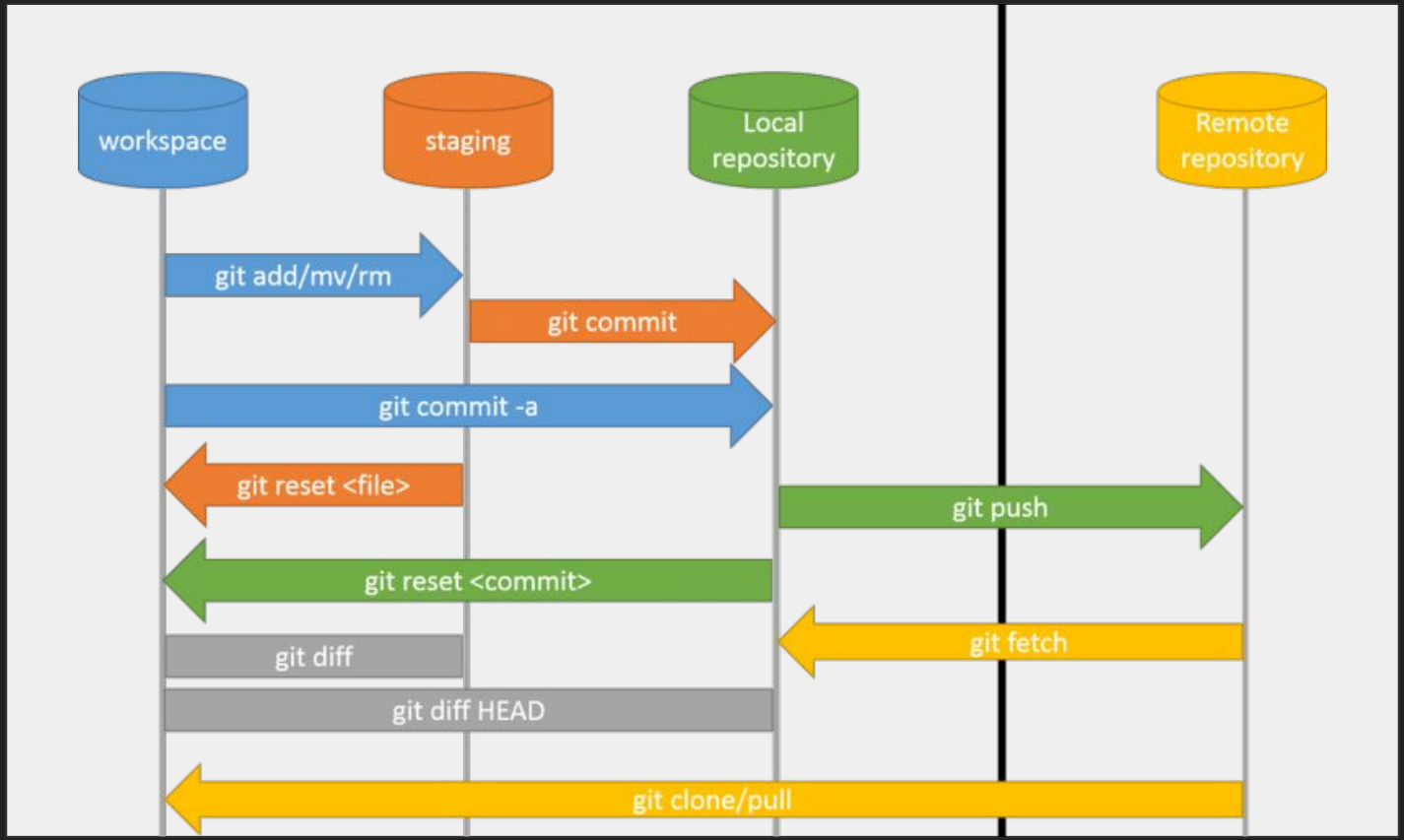
- ➔ Branch e merge.
- ➔ Fazer com que membros da equipe trabalhem ao mesmo tempo é muito fácil, mas mesmo indivíduos trabalhando por conta própria podem se beneficiar da capacidade de trabalhar em fluxos independentes de alterações.
- ➔ Criar um "branch" em ferramentas VCS mantém vários fluxos de trabalho independente uns dos outros enquanto também oferece a facilidade de mesclar esse trabalho e unir de novo, permitindo que os desenvolvedores verifiquem se as alterações em cada branch não estão em conflito.

Benefícios do Controle de Versão [1]

- ➔ Rastreabilidade.
- ➔ Ser capaz de rastrear todas as alterações feitas no software e as conectar ao software de gerenciamento de projeto e rastreamento de bugs, como o Jira, e ser capaz de comentar cada alteração com uma mensagem descrevendo o propósito e intenção da alteração pode ajudar não apenas na análise da causa raiz e outras análises forenses.
- ➔ Ter o histórico anotado do código na ponta dos dedos ao ler o código, tentar entender o que ele está fazendo e por que foi criado assim pode permitir que os desenvolvedores façam alterações corretas e harmoniosas que estejam de acordo com o projeto a longo prazo do sistema.
- ➔ Isso pode ser muito importante para trabalhar com eficiência no código legado e é crucial para permitir que os desenvolvedores estimem o trabalho futuro com alguma precisão.

Git

➔ Existem muitas funcionalidades do Git


workspace
staging
Local repository
Remote repository
git add/mv/rm
git commit
git commit -a
git reset <file>
git push
git reset <commit>
git fetch
git diff
git diff HEAD
git clone/pull

Fonte: [2]

Git

- ➔ Vamos usar apenas as mais simples hoje
- ➔ Primeiro, vamos criar um novo repositório local
- ➔ Depois, adicionar ele em um repositório remoto
- ➔ Vamos adicionar uma classe de Pessoa como exemplo
- ➔ Vamos elencar as mudanças e escrever o commit
- ➔ Vamos dar o commit e, para levá-lo ao repositório remoto, dar um push
- ➔ Depois, vamos criar uma branch nova!

Git

- ➔ Agora, nessa nova *branch*, vamos salvar nossas pessoas em arquivos JSON e fazer a leitura do arquivo
- ➔ Depois vamos dar merge da branch na *master*

JSON

JSON [3]

- Primeiro de tudo, é preciso baixar a biblioteca JSON
  - https://github.com/stleary/JSON-java
- Crie um novo diretório (*package*) filho do diretório *src*
  - Chame de *org.json*
  - Importe tudo que extrair da biblioteca para essa pasta
- Pronto, agora podemos usar JSON

JSON [4]

- Mas o que é JSON?
  - *JavaScript Object Notation*
  - Formato de troca de dados leve
  - Fácil para humanos entenderem e escreverem
  - Fácil para máquinas parsear e gerar
  - Independente de linguagem, mas usa convenções familiares a programadores de linguagens da família C (C, C++, C#, Java, Python, etc.)

JSON [3]

- Agora podemos criar objetos JSON
  - Uma coleção não ordenada de pares nome/valor
- Usamos os métodos *get()* e *opt()* para acessar os valores por nome
- Usamos o método *put()* para modificar valores

JSON [3]

```
JSONObject my_obj = new JSONObject();
my_obj.put("titulo", "JSON x XML: a Batalha Final");
my_obj.put("ano", 2012);
my_obj.put("genero", "Ação");
String json_string = my_obj.toString();
System.out.println("objeto original -> " + json_string);

my_obj.put("titulo", "JSON x XML: o Confronto das Linguagens");
json_string = my_obj.toString();
System.out.println("objeto com o título modificado -> " + json_string);

String titulo = my_obj.getString("titulo");
Integer ano = my_obj.getInt("ano");
String genero = my_obj.getString("genero");
System.out.println("titulo: "+titulo+" ano: "+ano+" genero: "+genero);
```

JSON [3]

- Também podemos criar arrays em JSON
- Podem ser valores de tipo básico ou objetos JSON

JSON [3]

```
//instancia um novo JSONObject
JSONObject my_obj = new JSONObject();

//preenche o objeto
my_obj.put("titulo", "JSON x XML: a Batalha Final");
my_obj.put("ano", 2012);

//cria um JSONArray e preenche com valores string
JSONArray my_genres = new JSONArray();

my_genres.put("aventura");
my_genres.put("ação");
my_genres.put("ficção");

//insere o array no JSONObject com o rótulo "generos"
```

Git

➔ Agora, vamos dar o branch na master

GitFlow

Git

- Padrão de ramificação do Git
- Resumidamente você deve criar uma branch para cada nova feature e, ao terminá-la, dar *merge* com a *develop*
- Essa *develop* é uma branch ESTÁVEL de desenvolvimento, e que deve ser combinada com a master APENAS em *releases*
- https://datasift.github.io/gitflow/IntroducingGitFlow.html

Referências

1. https://www.atlassian.com/br/git/tutorials/what-is-version-control
2. https://gustavohenrique.net/2011/03/comandos-basicos-do-git/
3. https://www.devmedia.com.br/trabalhando-com-json-em-java-o-pacote-org-json/25480
4. https://www.json.org/json-en.html
5. https://datasift.github.io/gitflow/IntroducingGitFlow.html

Referências

Bons guias para git na linha de comando:

- ➔ http://rogerdudler.github.io/git-guide/
- ➔ https://www.sitepoint.com/git-for-beginners/
- ➔ https://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide
- ➔ https://tableless.com.br/tudo-que-voce-queria-saber-sobre-git-e-github-mas-tinha-vergonha-de-perguntar/

Se você precisar criar um .ignore

- ➔ https://www.gitignore.io/