Introdução ao Git

Resumo

Juliano Laganá

Resumo com os principais tópicos
abordados no curso

Sumário

1 Introdução

Esse documento é um resumo dos assuntos que foram abordados no curso introdutório de Git nos dias 11 e 12 de fevereiro de 2015. A sintaxe completa dos comandos está disponível em na documentação do git (use o comando `git help` para acessá-la), por clareza ela não será reproduzida aqui.

Por favor, considere manter uma versão digital desse documento ao invés de imprimi-lo.

2 Arquitetura Git

Três fatos importantes para serem lembrados sobre a arquitetura do Git.

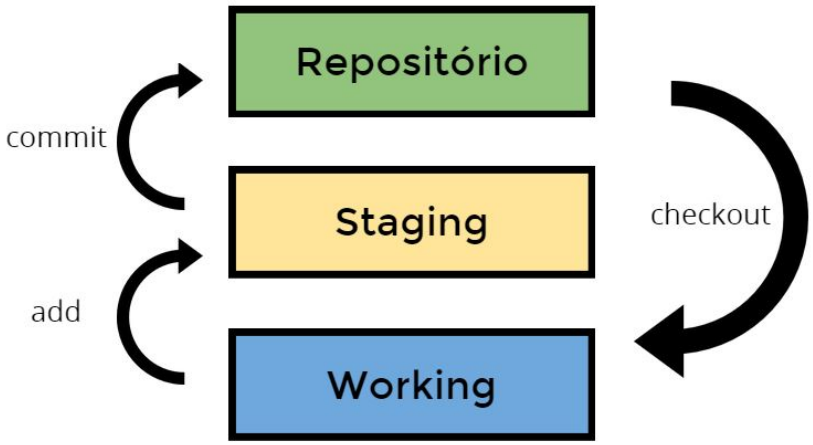

Figura 1: Arquitetura de três árvores do Git

- O Git utiliza uma arquitetura de três árvores: O working directory (diretório de trabalho), o repositório (diretório que o git gerencia; é aqui que ele guarda as versões passadas) e a staging index (etapa intermediária entre o working e o repositório).
- Para cada um dos commits, o git utiliza um algoritmo (SHA-1) que gera um código de 40 caracteres (que também é chamado de SHA-1). É pelo SHA-1 de um commit que o Git se refere à ele.
- O ponteiro HEAD é um ponteiro que sempre aponta para o último commit que foi feito. O próximo commit sempre é armazenado logo depois de onde estiver o HEAD no momento.

3 Criando commits

Comandos utilizados para criar seu primeiro commit:

- `git init`: Pede para o git começar a acompanhar as mudanças que forem feitas na pasta atual.
- `git add`: Atualiza a staging index com as mudanças que foram feitas no working.
- `git commit`: Atualiza o repositório com as mudanças que foram feitas na staging index.

- `git status`: Mostra o estado atual de cada uma das árvores do Git.
- `git log`: Mostra todos os commits que já foram feitos.
- `git diff`: Comparar diferenças entre as árvores ou entre commits.

4 Recuperando versões passadas

Comandos utilizados:

- `git checkout`: Utilizado para recuperar uma versão passada de um arquivo.
- `git reset`: Retorna à um commit especificado. Três variações:
  - `soft`: Muda somente o ponteiro HEAD de posição.
  - `mixed`: Muda o ponteiro HEAD e a staging index.
  - `hard`: Muda o ponteiro HEAD, a staging index e o diretório de trabalho!

5 Criando branches

Comandos utilizados:

- `git branch`: Cria um novo branch.
- `git checkout`: Troca de branches.
- `git merge`: Junta dois branches.

5.1 Ponteiros

Existe um ponteiro HEAD, como foi descrito anteriormente, e mais um ponteiro para cada um dos branches que você tiver em seu repositório. O ponteiro HEAD sempre aponta para o último commit que foi feito, enquanto o ponteiro de cada branch sempre aponta para o último commit naquele branch.

5.2 Conflitos em um merge

Ao dar um merge, pode ser que ambos os branches que estão sendo combinados tenham modificado as mesmas partes de um arquivo. Nessa situação, como o Git não tem por que escolher um dos branches em detrimento de outro, ele pede ao usuário para que resolva o conflito. Os conflitos podem ser resolvidos manualmente abrindo o arquivo conflitante e escolhendo qual das versões será mantida (as partes conflitantes estarão sinalizadas).

5.3 Como reduzir conflitos

- Commits pequenos e focados.
- Faça merges frequentemente.
- Mantenha-se informado do que está acontecendo no master (processo chamado de "tracking").

6 Repositórios remotos

Comandos utilizados:

- `git remote`: Utilizado para configurar quem é o servidor remoto.
- `git push`: Envio de informações do seu computador para o servidor remoto. É utilizado depois que foi feita alguma alteração no seu repositório Git que você quer que o repositório remoto também possua.
- `git fetch`: Envio de informações do servidor remoto para o seu computador. É utilizado para atualizar seu computador com possíveis novas mudanças que o repositório remoto possua.

Alguns provedores de servidores de Git populares:

- GitHub
- Bitbucket
- Codebase

7 Tópicos finais

- Algumas opções úteis do `git log`: `--oneline`, `--decorate`, `--graph`, `--all`, `--before`, `--after`, `--author`...
- Para ignorar arquivos, crie um arquivo chamado `.gitignore` na pasta raíz do projeto. Dentro dele, utilize expressões regulares para explicar para o Git o que você não quer que ele acompanhe mudanças

- Stash: Pilha para colocar mudanças que ainda não foram nem staged nem commited.
  - Colocar as mudanças atuais na stash: `git stash`
  - Ver o conteúdo atual da stash: `git stash list`
  - Tirar a mudança da stash e aplicar: `git stash pop stash@{n}` (onde `n` é a posição da mudança que você está se referindo na stash.
  - Só aplicar, sem tirar as mudanças da stash: `git stash apply stash@{n}`
  - Tirar da stash sem aplicar: `git stash drop stash@{n}`
  - Limpar a stash: `git stash clear` (cuidado, apaga tudo)
- Clonar um repositório já existente: `git clone linkgithub nomepasta`, onde `linkgithub` é o link que o GitHub mostra na página principal do projeto que você quer clonar e `nomepasta` é o nome que você quer dar para a pasta que será criada com o conteúdo do projeto no seu computador.
- Recursos para aprendizado:
  - Comando `git help`
  - Livro Pro Git
  - Stack Overflow