Curso de git

Aula 2

Pet Computação

24 de outubro de 2018

Revisão

- Conceitual
  - Ciclo de estados
- Comandos básicos
  - git init, status, add, commit
  - git log, branch, desfazendo modificações

Possíveis estados

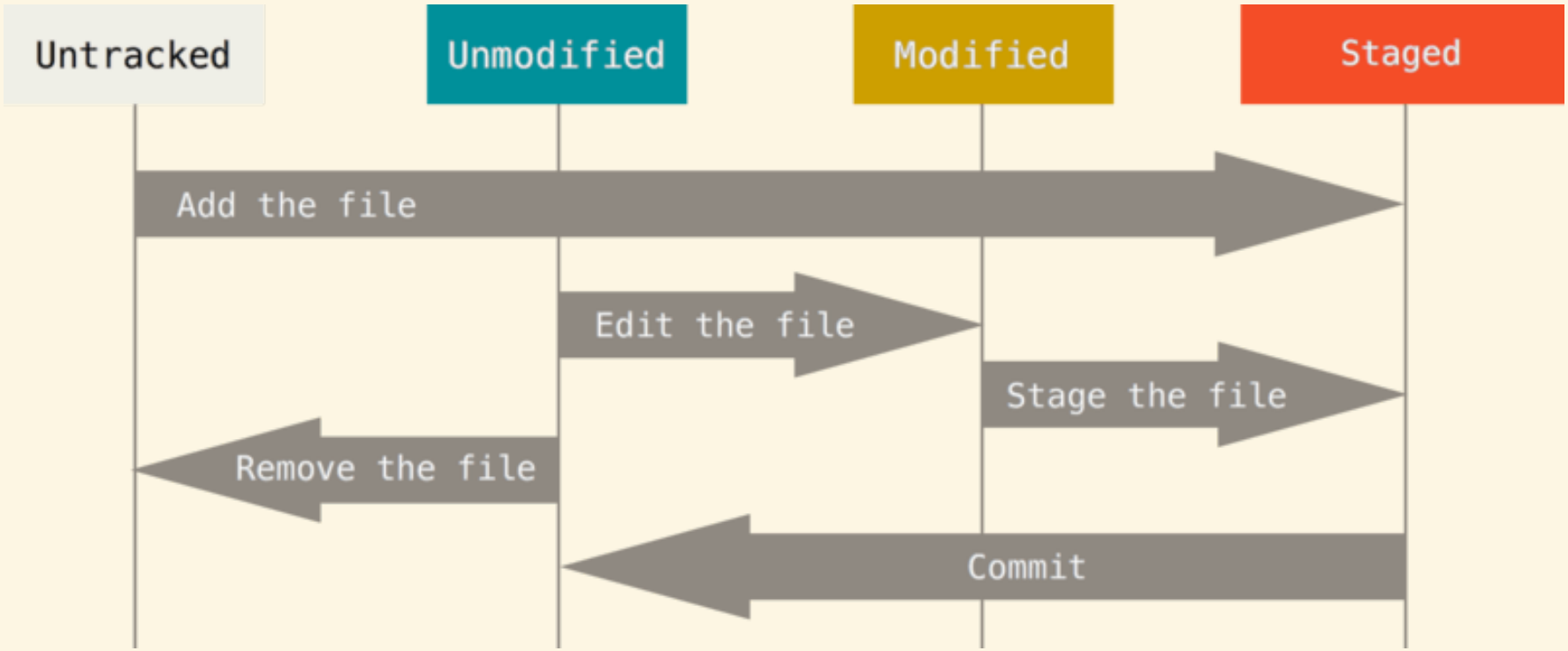

Figura: Ciclos de estados de um arquivo

git init, git status

git init

Para inicializar um repositório git utilize o comando `git init` dentro do diretório principal do projeto.

```
git init
```

git status

Mostra quais arquivos não estão sendo monitorado, foram modificados e quais estão na área de transição.

```
git status
```

git add, git commit

git add

Adiciona um arquivo a área de transição. Um arquivo só pode ser adicicionado a área de transição se: não estiver sendo monitorado, estiver sendo monitorado mas foi modificado ou se estiver sendo monitorado e foi deletado.

```
git add arquivo
```

git commit

Faz o commit das modificações dos arquivos que estão na área de transição.

```
git commit -m "uma␣mensagem"
```

git log, git branch

git log

Mostra o log de commits do repositório.

```
git log --oneline
```

git branch

Permite várias pessoas trabalharem em visões diferente do projeto e no final mesclar todo o trabalho produzido.

```
# cria uma nova branch
git branch nomeBranch
# muda para uma nova branch
git checkout nomeBranch
```

Desfazendo coisas I

Arquivo modificado

Desfazendo mudanças em arquivos modificados que não estão na área de transição.

```
git checkout -- nomeArquivo
```

Retirando arquivos da área de transição

Para retira um arquivo da área de transição:

```
git reset HEAD nomeArquivo
```

Desfazendo coisas II

Atualizando a mensagem do último commit

```
git commit --amend -m "nova␣mensagem"
```

Adicionando arquivos ao último commit

```
git add arquivo;
git commit --amend
```

Revertendo alterações até um commit

```
git revert <commit>
```

Exercício

1. crie uma pasta chamada atividade_git e inicialize um repositório git nela.
2. Crie um arquivo chamado nomes, e comece a monitorá-lo.
3. Insira 2 nomes no arquivo nomes, cada nome em uma linha e consolide as mudanças com um commit
4. Crie uma branch chamada mod1 e mude para ela
5. Insira 4 nomes no arquivo nomes e faça um commit com a mensagem " primeiro commit na branch mod1"
6. Ainda na branch mod1, insira mais um nome e atualize o ultimo commit (não deve fazer outro commit)
7. Volte para a master e crie outra branch chamada mod2, mude para ela.
8. Na branch mod2, edite o primeiro nome do arquivo nomes e adicione mais 3 nomes.

Exercício continuação

9. Faça o commit com a mensagem “primeiro commit da branch mod1”
10. Corrija a mensagem do ultimo commit da branch mod2 para “primeiro commit da branch mod2”
11. Volte para a master.
12. Dê merge da branch mod1, se houver conflitos o resolva.
13. Dê merge da branch mod2, se houver conflitos o resolva.
14. Em casos de conflitos, Resolva de modo que o arquivos nomes contenha o maior numero de nomes.
15. Faça um git revert para o commit “meu primeiro commit da branch mod1” se houver conflitos resolva.

Solução parte I

```
# item 1
mkdir atividade_git ; cd atividade_git;
git init;
# item 2
touch nomes; git add nomes;
git commit -m "adiciodado arquivo nomes"
# item 3
echo "Odr" >> nomes; echo "Mario" >> nomes;
git commit -am "Adiciona dois nomes em nomes"
# item 4
git branch mod1;
git checkout mod1;
# item 5
echo "Ditkun" >> nomes; echo "Junior";
echo "Joao" >> nomes; echo "Raul";
git commit -am "primeiro commit na branch mod1"
```

Solução parte II

```
# item 6
echo "Miguel" >> nomes;
git add nomes;
git commit --amend;
# item 7
git checkout master;
git checkout -b mod2;
# item 8
sed -i "1s/Odr/Odair/" nomes;
echo "Bruno" >> nomes; echo "Marcos" >> nomes;
echo "Eduardo" >> nomes;
# item 9
git add nomes;
git commit -m "primeiro␣commit␣na␣branch␣mod1"
# item 10
git commit --amend -m "primeiro␣commit␣na␣branch␣mod2"
```

Solução parte III

```
# item 11
git checkout master
git merge mod1;
# item 12
git merge mod2;
# item 13
git log --oneline; # pegar o commit mod1
git revert commitMod1;
```

Repositórios remotos

O git é software de controle de versão distribuído, o que dá possibilidade de várias pessoas trabalharem no mesmo projeto, cada um em sua estação de trabalho e depois sincronizar todo o progresso. Existem alguns servidores online que permitem hospedar o seu projeto e servir como servidor do sistema distribuído, são eles:

- Gitlab
- Github
- bitBucket

Gitlab

gitlab.c3sl.ufpr.br

- git do dinf
- versão *open source*

gitlab.com

- Algumas ferramentas extras
- Possui ferramentas pagas

Github

- Maior comunidade de software livres
- Ferramentas para contribuição comunitária
- Repositório privados somente se pagar

Obtendo um repositório remoto

Para obter um repositório remoto que está hospedado em alguma plataforma web como gitlab ou github utilize o comando `git clone url`. Existem duas formas de clonar um repositório, utilizando o protocolo https (se o projeto estiver privado é necessário se autenticar) ou por ssh que não é necessário se autenticar.

git clone por https

```
git clone https://gitlab.c3sl.ufpr.br/omdj17/curso_git.git
```

git clone por ssh

```
git clone git@gitlab.c3sl.ufpr.br:omdj17/curso_git.git
```

Atualizando repositório remoto I

Ao utilizar repositório remoto você pode obter atualizações do remoto para o local e depois enviar atualizações locais para o remoto.

git pull

Para obter todas as atualizações do repositório remoto para o local utilize o comando `git pull`. É importante lembrar que não deve haver arquivos monitorados e modificados ou arquivos na área de transição ao fazer git push.

git push

Em termos simplificados o comando `git push` tem como função enviar atualizações para o repositório remoto. Se voce estiver em uma branch que já exista no repositório remoto faça: `git push` caso contrário é necessário fazer:

```
git origin mod1
```

Atualizando repositório remoto II

git push

```
git push --set-upstream origin mod1;
git push;
```

Branch remota

Ao se trabalhar com repositório remoto, existe uma distinção de branchs, existe branchs e branchs remotas. Branchs remotas são referencias a uma branch do servidor é como se fosse uma subbranch da branch. As branchs remotas são precedida de onde ela está, normalmente 'origin'.

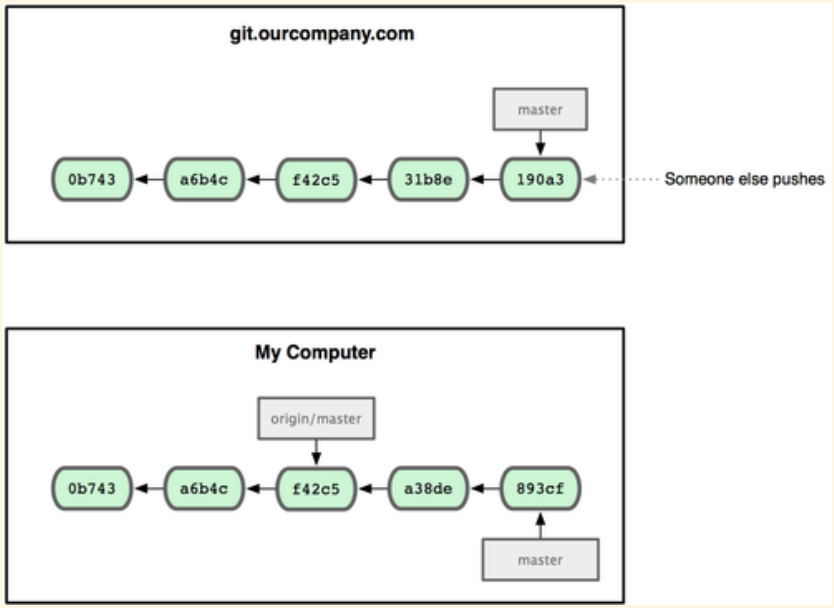

Figura: branch master local e remota sincronizada

Branch remota

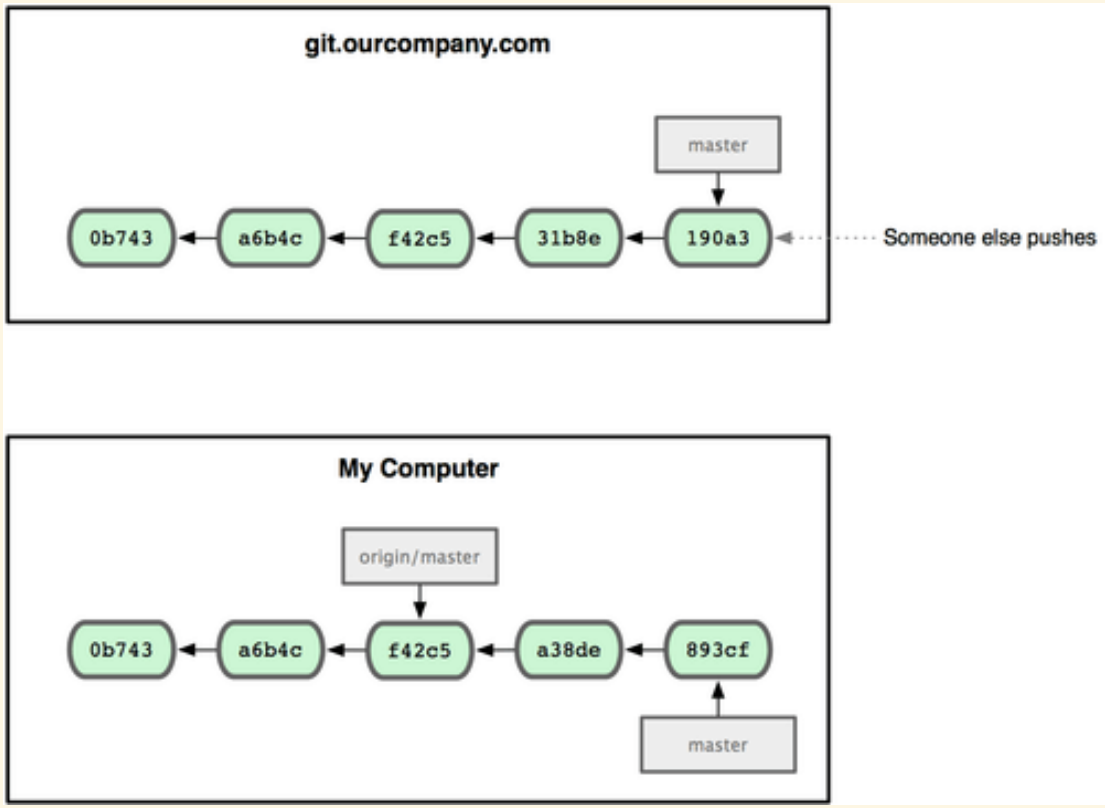

Figura: branch master local e master remota dessincronizada

git fetch

O comando `git fetch` atualiza a referência da branch remota para o commit mais atual. Em termos práticos ele verifica se houver novos commits e se sim atualiza a referencia das branchs dos commits

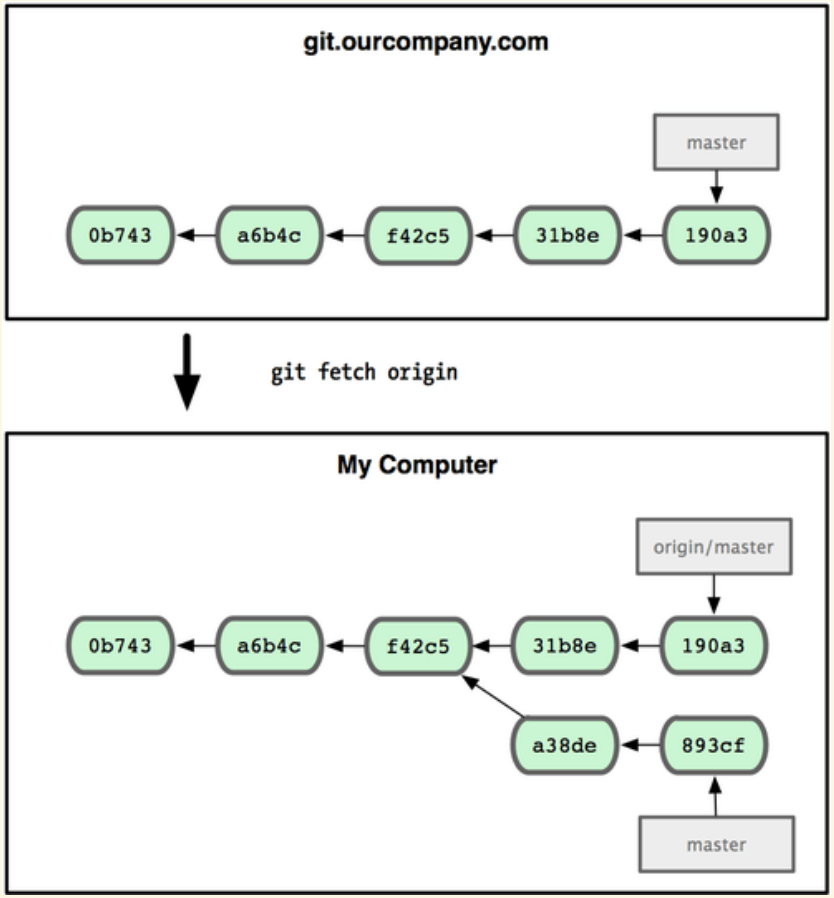

Figura: git fetch

git stash conceito

O comando `git stash` salva o estado dos arquivos monitorados e modificados e arquivos que estão na área de transição temporariamente, de modo a permiti operações como mudança de branchs e git push.

git stash prática

Fazendo git stash

```
git stash;
```

Voltando ao trabalho

Existem duas opções do `git stash` que permitem a volta do progresso a `git stash apply stash@ id` que voce precisa passar o id do stash e a opção `git stash pop` que aplica o último stash.

Removendo stashs

Remover um stash implica em perder o progresso armazenado no stash. O comando `git stash clear` remove todos os stashs do repositório. O comando `git stash drop stash\@ id` remove o stash especificado.

Chave ssh

Chaves ssh servem para evitar autenticação por login e senha a cada interação com o repositório remoto. O comando para gerar ssh key é `ssh-keygen`

Opções do ssh-keygen

-t: rsa tipo de criptografia -C: Adiciona um comentário a chave -b: quantidade de bits da chave

Criando chave ssh

As chaves ssh normalmente são guardada na pasta .ssh dentro da home.

```
cd ~/.ssh;
ssh-keygen -t rsa -C "your.email@example.com" -b 4096
```

ssh key sem nome padrao

Caso você criou a chave ssh sem o nome padrão é necessário configura o arquivo config na pasta ṡsh.

Arquivo config

```
Host gitlab.company.com # url do repositrio remoto
RSAAuthentication yes
IdentityFile ~/.ssh/config/private-key-filename
```

Gitlab e commits

O gitlab verifica as mensagens de commits em busca de determinados padrões. Na prática você consegue relacionar issues.

Relacionando issues a um commit

Para relacionar uma issue ao um commit basta utiliza # e o id da issue.

```
git commit -m "#id␣mensagem"
```

Algumas palavras podem alterar o comportamento das issues como fechar elas, são elas:

- Close, Closes, Closed, Closing, close, closes, closed, closing
- Fix, Fixes, Fixed, Fixing, fix, fixes, fixed, fixing
- Resolve, Resolves, Resolved, Resolving, resolve, resolves, resolved, resolving

Markdown

Markdown é uma linguagem de marcação utilizada em servidores webs como github e gitlab.

- readme
- wiki
- descrição de issues

Um bom tutorial de markdown: Tutorial Markdown

Readme

Readme é um arquivo que vai aparece na página principal do seu projeto em um servidor web, ele tem como objetivo introduzir novas pessoas ao seu trabalho. Um bom readme contêm os seguintes itens:

- Descrição do projeto
- Como obter o projeto
  - Pré-requisitos
  - tutorial de instalação
- Como rodar o projeto
- Como contribuir para o projeto
- Bugs e problemas conhecidos
- Créditos
- Licença do projeto

Wiki

A maioria de servidores online de git como gitlab e github implementa uma funcionalidade chamada wiki, que na prática é uma parte do projeto em que pode escrever textos do projetos, como por exemplo relatório de desenvolvimento, documentação de desenvolvimento e de utilização e outros. Normalmente projetos relativamente grande e complexos implementam uma seção de tutorial de como desenvolver ou utilizar o projeto.

Descrição de issues

Em projeto comunitário é essencial que esteja claro quais são os problema e o que precisa ser feito para o projeto finalizar. E descrever issues é um bom começo, um template genérico para issues pode ser:

- Descrição breve do problema ou do que deve ser implementado.
- lista de tarefas que devem ser feitas para completar a issues
- informações relevante