VERSIONAMENTO DE ARQUIVOS COM GIT

DIO CEZAR GO

AGENDA

- o que é controle de versão?
- ecossistemas para controle de versão;
- vantagens;
- git e GitHub;
- instalando git;
- configuração básica;
- iniciando um projeto no git;
- o básico do git;
- clonando repositórios;
- forks;
- branches;
- pull-request;
- hacks;
- materiais complementares;

O QUE É CONTROLE DE VERSÃO?

- cópias e mais cópias do mesmo projeto?
- com pequenas modificações?
- apagou um arquivo sem querer? E não dá para recuperar!
- solução: **Controle de Versão**
    - sistema com a finalidade de gerenciar diferentes versões do mesmo arquivo;
    - cada modificação (no projeto) gera uma nova versão;

ECOSSISTEMAS

- existem sistemas que fazem o controle desse ecossistema:
  - Git
  - Svn
  - Mercurial
  - Bazaar
- querem se aprofundar? vejam mais aqui: https://goo.gl/9TKjtj

VANTAGENS

Professor, não podemos apenas fazer um .zip e enviar pelo Moodle?

- não! no mundo real, você vai precisar saber utilizar um versionador de arquivos;
- mas para justificar:
  - você sempre vai ter um histórico de tudo o que for modificado em um arquivo;
  - programação colaborativa: diff;
  - vocês podem trabalhar em diferentes branchs e depois juntar tudo;
  - se perder alguma coisa, ou fizer algo errado... é só voltar!
  - vocês saberão o que cada um fez e em que parte o arquivo foi modificado;
  - podem clonar o repositório no ambiente de produção!

GIT É O GITHUB?

- não! git é a tecnologia para versionamento de arquivos;
- GitHub é uma plataforma que utiliza o git para armazenar os projetos;
- ou ainda... uma rede social para programadores?
  - mostre seus códigos para todos;
  - participe de projetos *OpenSource*;
  - pague para usar repositórios privados;

INSTALANDO O GIT

- tem tudo aqui: https://git-scm.com/download
- Mac → vem com o xcode;
- Linux → já vem instalado;
- Windows → deve-se instalar o pacote;

CONFIGURAÇÃO BÁSICA

- depois de instalado, podemos fazer algumas configurações básicas:
- em um terminal:
  - `git config --global user.name "Diogo Cezar"`
  - `git config --global user.email "diogo@diogocezar.com"`
- se algo der errado, será mostrado na tela;
- são essas informações que serão utilizadas para enviar o projeto para o repositório;

COMO INICIALIZAR UM PROJETO

- crie uma pasta:
  - `mkdir project`
- entre nessa pasta:
  - `cd project`
- para inicializar um repositório:
  - `git init`
- repare que um diretório é criado:
  - `ls -la`

O BÁSICO DO GIT

- quando se cria um novo arquivo ele não está sendo visto pelo Git; **(untracked)**
- para adicionar um arquivo: **(stage)**
  - `git add file.ext`
  - `git add --all`
- se houver uma edição nesse arquivo, ele passa para um outro estado **(modified)**
- pode-se observar os estados dos arquivos com o comando:
  - `git status`
- um **commit** só "leva" arquivos que estão no status stage;

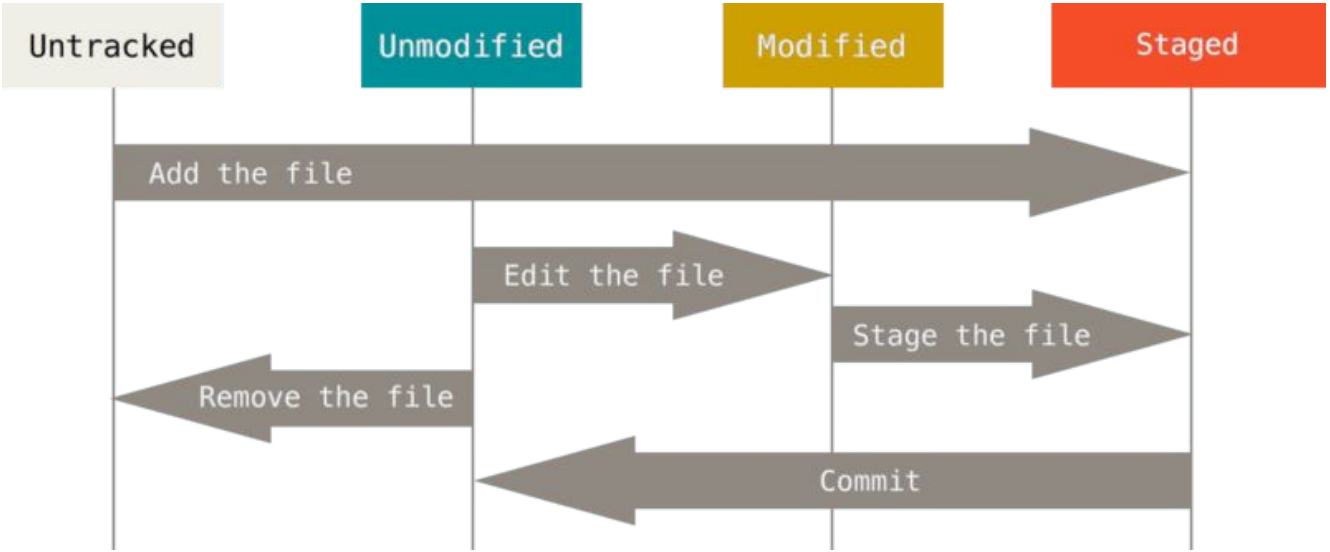
O BÁSICO DO GIT

- um **commit** só "leva" arquivos que estão no status **stage**;
- por isso, sempre que modificar um arquivo, tem que adicioná-los novamentel;
- nesse momento o git sabe de sua existência, mas não existe nenhum "commit" para essa versão.
- um **commit** é a criação de um *snapshot* do sistema, ou seja, um printscreen da situação atual do seu sistema.

O BÁSICO DO GIT

- para realizar um **commit**:
  - `git commit -m "Mensagem referente a sua modificação"`
- quando se faz um commit, tudo está ok, mas só na sua máquina!

O BÁSICO DO GIT



O BÁSICO DO GIT

- precisamos ligar o nosso git a um repositório na Internet;
  - `git remote add origin <endereço-do-repositório>`
- origin pode ser qualquer nome;
- um mesmo repositório pode ter diferentes locais remotos;
- para ver a lista de repositórios remotos:
  - `git remote`

O BÁSICO DO GIT

- já temos nosso commit, e um repositório linkado; precisamos enviar isso!
  - `git push -u origin master`
- `-u` "*trackeia*" o comando e possibilita que os próximos comandos sejam apenas `git push`;
- origin é o repositório destino;
- master é o branch que estamos;
  - *master* é sempre o *branch default*;

CLONANDO REPOSITÓRIOS

- podemos também, ao invés de criar um repositório e ligá-lo a uma origem, cloná-lo:
  - `git clone <endereço-do-repositório>`
- isso deixa nosso repositório pronto para realizar os *commit's* e *push's*;

RESUMINDO, ATÉ AGORA!

CRIANDO E LIGANDO A REPO

```
mkdir project
git init
touch myproject.txt
git add --all
git commit -m “Enviando arquivo”
git remote add origin …
git push -u origin master
```

CLONANDO UM REPO

```
git clone …
touch myproject.txt
git add --all
git commit -m “Enviando arquivo”
git push
```

FAZENDO FORKS

- o que é?
  - é uma cópia de um projeto para sua conta do GitHub;
- isso é ideal para realizar contribuição! por que?
  - você pode fazer um *pull-request* ao dono do repositório que quer contribuir;
- como fazer isso?
  - basta ir ao repositório no GitHub, estando logado, e clicar em fork;
- diferença do clone
  - clone → só consigo fazer para meus repositórios, até consigo clonar outros, mas nunca conseguirei enviar modificações;
  - fork → posso contribuir em repositórios de terceiros

BRANCHES

- o que é?
  - é um ponteiro para determinado *snapshot* do seu sistema de arquivos;
- por que usar?
  - pode-se modificar meus arquivos sem alterar o meu fluxo principal;
  - pode-se corrigir um bug em determinada funcionalidade sem interferir no fluxo original do projeto.
- e como juntar com o projeto principal?
- através do comando *merge*!
  - para mais informações: https://goo.gl/MQJGuw

PULL-REQUEST

- vamos a um passo a passo de como realizar um *pull-request*:
- baseado no material disponível em: https://goo.gl/ptmfSc

- fazer o *fork* do repositório em questão;
- clonar o repositório para a sua máquina:
  - `git clone <endereço-do-repositório>`

PULL-REQUEST

- criando um *branch*
- primeiro tenha certeza que você está no *master*:
  - `git branch`
- cria-se uma nova *branch*
  - `git branch name_of_branch`
- entra-se nesta branch
  - `git checkout name_of_branch`

PULL-REQUEST

- agora pode-se criar, editar e modificar os arquivos de acordo com sua necessidade;
- por que criar uma branch?
  - *branch master* é a *branch* com o código final do projeto, estável.
  - criando uma nova branch, ao submeter o *pull request* para o repositório original, caso não for aceito, as alterações não estarão na *branch master*.
  - desta forma, se você quiser manter sempre os dois repositórios atualizados e sincronizados, você só precisa apagar a branch que você criou e fez a *feature*.
  - as duas *master* vão continuar iguais;

PULL-REQUEST

- enviando para o seu fork:
  - `git add --all`
  - `git commit -m "Mensagem de commit"`
- enviando a branch
  - `git push origin nome_da_branch`
- acesse sua conta no GitHub
- basta clicar no botão verde: *Compare & Pull Request*;
- será direcionado para uma tela onde irá poder criar de fato o seu *pull-request*
- coloque um título e a descrição, e pronto!

HACKS

- você pode usar um gerenciador de git para obter vários recursos visuais em seu sistema operacional;
- https://tortoisegit.org/ → Windows
- uma lista com várias opções → https://goo.gl/hqWJT2

MÃOS NA MASSA!

- criando um repositório no GitHub;
- enviando um projeto web para o GitHub;
- clonando um repositório;
- realizando o fork de um repositório;
- enviando um pull-request;

MATERIAIS COMPLEMENTARES

- conteúdo oficial GIT:
  - https://goo.gl/Vxegfh
- curso git para iniciantes:
  - https://goo.gl/9G343o
- importância do GitHub para desenvolvedores:
  - https://goo.gl/eHDvik