UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DIO CEZAR GO

GIT E GITHUB

Versione seus projetos e participe da rede social dos programadores

APRESENTAÇÃO

quem sou eu?

APRESENTAÇÃO

- olá, sou o **Diogo Cezar**!
  - mestre em IA pela *UFPR* (2012);
  - tecnologia em Informática *UTFPR* (2007);
  - Full Stack Web Developer;
  - professor;

FUI PARAR NO BBB!

- mas... antes dos autógrafos!
- Hackaton Globo 2016
  - O que é?
  - Como funciona?
  - Como foi a experiência?

COISAS IMPORTANTES

- http://www.diogocezar.com - este é meu site;
- http://www.creativetroops.com.br - hub de profissionais de tecnologia;
- http://github.com/diogocezar - este é meu github;
- diogo@diogocezar.com - este é o meu e-mail;
- diogoc@utfpr.edu.br - este também (usem esse);
- sala: **K001**;

UM POUCO DO QUE JÁ FIZ

I’M BACK!

- estive na UTFPR em **2008** e **2009**;
- dei aulas na pós-graduação em **2013**;
- **2017** de volta a UTFPR;

MATERIAIS UTFWARE

https://github.com/diogocezar/dctb-utfware

DICAS GERAIS

o que eu já aprendi com a vida!

DICA 01

Se você é programador, deve passar boa parte do seu tempo atrás de uma tela de terminal!

Sério! aprendam e usem o terminal.

DICA 02

Só programar não é suficiente :(você precisa aprender a criar e configurar os seus ambientes de desenvolvimento e produção.

DICA 03

Aprendam Linux/Unix!

A maioria dos ambientes (reais) de desenvolvimento e produção estão abaixo de um ecossistema Linux/Unix.

Você irá precisar disso, mais cedo ou mais tarde.

Windows? *Máquina Virtual* ou *Bash*.

DICA 04

A melhor forma de aprender (com pessoas) é fazendo amizades

Arquivo confidencial: aprendi pela dor!

Você confia, ajuda e compartilha mais com seus amigos.

DICA 05

Hierarquia? Deixe para quando você for chefe.

Estou aqui para aprender (junto) com vocês!

Todos estamos aqui para aprender.

DICA 06

Se eu não souber, vou procurar.

Vocês, façam o mesmo!

DICA 07

A melhor skill que um programador pode ter não é saber de tudo, é saber se virar e fazer acontecer!

Atualmente, as coisas evoluem MUITO rápido! Acreditem, por experiência própria, vocês não vão conseguir dominar tudo!

Mas alivia saber que, se você precisar, vai saber como encontrar e como resolver.

Treine essa habilidade! Vai ser a sua principal durante sua vida de desenvolvedor;

DICA 08

As pessoas... bem elas são complicadas!

Tenha calma, trabalhe o seu psicológico para compreender que as pessoas são diferentes, nem sempre estão em um bom dia.

DICA 09

Aprenda um pouco sobre design!

Isso fez muita diferença durante todo o tempo que trabalhei.

Programadores nerds são ótimos, mas programadores que manjam de design conseguem entregar algo bonito, que é (as vezes) mais importante.

DICA 10

Seja responsável!

Nossa área está cheia de *sobrinhos*!

Clientes e empresas acabam sempre se decepcionando com entregas fora de prazo ou com escopo diferente do combinado, seja **responsável** e cumpra com suas obrigações.

Assim será reconhecido rapidamente.

DICA 11

Arrisque!

Nunca mais pense: será que consigo fazer isso?

Yes, you can!

É importante aprender a aprender, hoje, tem tudo no Google, ou melhor, no stackoverflow!

DICA 12

Faça contatos.

Hey nerdão! Saia de casa!

Conheça pessoas, mostre seus trabalhos, faça amizades indique parceiros, aumente a sua rede de contatos, e sempre terá oportunidades de trabalho.

DICA 13

Crie o seu GitHub hoje!

Empresas estão sempre de olho no que vocês andam “codando”.

Mas ninguém vai abrir o seu Moodle para ver se vc entrou a atividade ou não! Exponha todos seus estudos! Crie repositório com links, tutoriais e coisas do tipo.

Sabe aquele problema que você demorou um monte para resolver? Faça um repositório com esse tutorial. Além de ajudar outras pessoas, pode ajudar você mesmo no futuro!

DICA 14

Contribua com a comunidade!

Trabalhar em projetos *OpenSource* é uma ótima maneira de mostrar o que você sabe.
Contribuia sempre que possível com qualquer tipo de projeto que achar que pode melhorar.

Seja traduzindo, seja refatorando, sei lá... melhore as coisas!

DICA 15

A melhor linguagem é o Inglês!

Nunca mais escreva uma variável em HUE, digo.. PT-BR.
Sério, isso é importante. Quando tiver que mostrar seu código, pode ser para uma empresa de fora... Sempre faça tudo em Inglês!

Aprenda a ler e escrever em inglês, falar também é bom!

A maioria do conteúdo relevante na internet está em Inglês.

DICA 16

Acompanhe as tendências!

Você não vai conseguir aprender a programar em todas as novas linguagens que saírem.

Entenda como elas funcionam, se prepare para estar pronto caso alguém pergunte você conhece “Cobol” (zuera) NodeJs, GoLang, Angular, Vue, React, Kotlin... essas coisas...

DICA 17

Estude, sempre que possível

Está na faculdade, beleza, é hora de se concentrar no estudo. Terminou e acabou? Se engana meu jovem gafanhoto...

Você terá que estudar todos os dias se quiser acompanhar a próxima geração! Eles estão cada vez mais fortes e rápidos, mal posso ver seus movimentos!

DICA 18

Ajude!

Pode parecer bobagem, mas... seja uma pessoa boa! Ajude sempre que possível.

Você nunca perderá nada com isso.

Não existem perguntas idiotas, apenas respostas idiotas

DICA 19

Seja um membro ativo em fóruns

Fóruns e Grupos de discussões contribuem muito para ficar ligado nas últimas tendências, e lá é um bom lugar para ajudar e ser ajudado também.

DICA 20

Academicamente

Leia artigos científicos, eles é que mudam o mundo!

Participe de eventos;

Participe de algum projeto de pesquisa;

Publique um artigo;

Aprenda a escrever! É sério, é difícil e você vai precisar!

Professores? Respeite-os.

É HORA DA REVISÃO

ah, e você não sabe nada de terminal...

POR QUE?

- eventualmente você irá precisar acessar um ambiente de desenvolvimento ou produção baseado em um sistema operacional Linux.
- isso será visto em detalhes durante o curso, mas...
    - vamos listar alguns comandos básicos que provavelmente você me verá usando durante demonstrações e código;

COMANDOS LINUX

`du -hs *`	mostra o espaço usado por cada arquivo/diretório de usuário
`ls`	lista conteúdo do diretório local
`ls -l`	mostra conteúdo detalhado
`ls -a`	mostra arquivos escondidos
`ls -la`	combinação de ambos
`mkdir`	cria diretório
`cp`	copia arquivo
`cp -r`	copia recursivamente (para copiar diretórios)

COMANDOS LINUX

`mv`	mover ou renomear arquivo/diretório
`rm`	apaga arquivo
`rm -r`	apaga recursivamente
`rm -rf`	apaga recursivamente sem confirmação (use com cuidado!)
`pwd`	mostra o diretório atual
`cat, more ou less`	mostram conteúdos de arquivo
`tail`	mostra final de arquivo
`head`	mostra começo de arquivo
`ssh`	acessa outra máquina Linux via protocolo seguro SSH
`nano`	abre o editor *nano*
`touch`	cria um arquivo em branco

Agora estamos prontos para começar... eu acho...

AGENDA

- o que é controle de versão?
- ecossistemas
- vantagens;
- git e GitHub;
- GitHub;
- basicão do MarkDown;
- Git;
- instalando git;
- configuração básica;
- iniciando um projeto no git;
- o básico do git;
- clonando repositórios;
- prática 1 - clonando;
- prática 2 - local;
- branches;
- merge e rebase;
- prática 3 - merge;
- prática 4 - rebase;
- forks e pull-request;
- prática 5 - pull-request;
- hospedando no github
- interfaces visuais;
- sites legais;
- materiais complementares;

CONTROLE DE VERSÃO

o que é isso? para que vou precisar?

O QUE É CONTROLE DE VERSÃO?

- já se deparou com cópias e mais cópias do mesmo projeto?
- cada cópia com pequenas modificações?
- já apagou um arquivo sem querer? e depois descobriu que não dava para recuperar?
- já teve que trabalhar em equipe e trocar o projeto por e-mail ou facebook?
- já desenvolveu algo que ficou pior que a versão anterior?

O QUE É CONTROLE DE VERSÃO?

- solução: **Controle de Versão!**
- o que é? → é um sistema com a finalidade de gerenciar diferentes versões do mesmo arquivo;
- cada modificação (no projeto) gera uma nova versão;
- podemos imaginar que a todo momento o versionador de arquivos tira uma foto de todo nosso sistema e guarda isso;
- caso seja preciso, podemos recuperar o estado no momento em que a foto foi tirada;

ECOSSISTEMAS

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

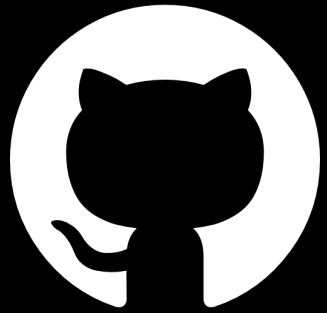
VANTAGENS

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 uma 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;



GITHUB

Uma rede social para programadores

GITHUB

- o GitHub foi adotado pela comunidade de desenvolvimento;
- é nele que a maioria dos projetos (inclusive os grandes) estão organizados;
- utiliza notações **MarkDown** para criação de sua documentação.
- segundo a fonte científica wikipedia → **Markdown** é uma linguagem simples de marcação originalmente criada por John Gruber e Aaron Swartz. Markdown converte seu **texto** em **XHTML válido**.

BASICÃO DO MARKDOWN

- # h1, ## h2, ### h3 ... → # define os níveis de títulos de seu texto;
- <!-- e --> → são comentários;
- *** → são linhas horizontais;
- textos são renderizados como parágrafos no HTML;
- ** negrito ** → textos em **negrito**;
- _ itálico _ → texto em *itálico*;
- ~~ texto tachado ~~ → texto ~~tachado~~;
- > texto → citação;

BASICÃO DO MARKDOWN

- listas não ordenadas → podem ser formatadas utilizando bullets, que podem ser:
  - + item
  - - item
  - * item
- listas ordenadas → utiliza formatação utilizando um número e um ponto:
  - 1. item
  - 2. item
  - 3. item

BASICÃO DO MARKDOWN

- código inline → são códigos que podem ser colocados na mesma linha; são delimitados por: `<seu_código>`;
- bloco de códigos → são blocos de códigos indentados; são delimitados por:
  - ``` <seu_bloco_de_códigos> ```;
- também é possível especificar uma sintaxe para seu bloco de códigos utilizando a linguagem após o início do bloco, como por exemplo:
  - ```js <seu_codigo_js> ```

BASICÃO DO MARKDOWN

- ainda é possível inserir tabelas, links, e imagens;
- você pode encontrar mais exemplos e detalhes aqui → https://goo.gl/EPMMi9

GITHUB

- com o GitHub é possível criar um perfil pessoal, para organizar seus projetos pessoais;
- ou ainda, criar uma *organization* um grupo que pode ter vários colaboradores e também seus próprios projetos;

GITHUB

- o **GitHub** é uma rede social, por isso, é possível seguir e ser seguido dentro da plataforma;
- você consegue acompanhar em um *feed* com a atividade de cada um dos usuários que você segue;

diogocezar

PedroFelipe starred erikras/redux-form an hour ago

MarcoWorms starred jaystack/repatch 3 hours ago

willianjusten created repository willianjusten/boilerplate-shiz 17 hours ago

AndersonMSoares forked diogocezar/dctb-utfpr to AndersonMSoares/dctb-utfpr 18 hours ago

amiltonjr forked diogocezar/dctb-utfpr to amiltonjr/dctb-utfpr 18 hours ago

Guihgo starred michalsnik/aos 19 hours ago

ReginaSobral created repository ReginaSobral/teste a day ago

ViniciusRomano created repository ViniciusRomano/aps-progweb a day ago

TecnologiaHacker starred diogocezar/dctb-web-project a day ago

HigorRozan created repository HigorRozan/UTFWare 2 days ago

HigorRozan created repository HigorRozan/UTFWare 2 days ago

Guihgo created repository Guihgo/PIC18F-MotorDC-example 2 days ago

Guihgo created repository Guihgo/C-Builder---Encrypt-MD5 2 days ago

Guihgo created repository Guihgo/Android---Bluetooth-example 2 days ago

Guihgo created repository Guihgo/NodejsChatExample 2 days ago

HigorRozan created repository HigorRozan/UTFWare 2 days ago

HigorRozan starred luandryl/unect 2 days ago

HigorRozan created repository HigorRozan/lastHope 2 days ago

HigorRozan created repository HigorRozan/oldLaravel 2 days ago

Repositories you contribute to 4

- droid-lab/droid-web-mggran 0
- droid-lab/droid-web-site 0
- cookingscript/cs-braziljs 4
- droid-lab/droid-shared 0

Your repositories 98 New repository

Find a repository...

All Public Private Sources Forks

- utfware-forkme
- utfware-local
- utfware-hello-world
- dctb-utfpr
- dctb-files-equality
- droid-shared
- dctb-bash-windows
- dctb-commands
- rodrigorhas/TerminalGame
- dctb-json-server
- dctb-js-data-structures
- dctb-sublime
- dctb-php-mvc
- dctb-php-mvc-db
- dctb-sass
- dctb-links
- droid-lab/droid-web-site
- dctb-grid
- dctb-flex
- droid-lab/droid-web-mggran

GITHUB

- as estrelas valem ouro → no GitHub os “likes” são medidos em estrelas; quanto mais estrelas seu projeto tiver, mais relevante ele é visto pela comunidade do GitHub;
- em seu perfil, estão: seus repositório principais, suas organizações, um resumo de duas contribuições, e uma *timeline* de suas atividades;
- além disso, é possível navegar entre seus repositórios, as estrelas que você marcou, seus seguidores, e as pessoas que você segue;

Diogo Cezar Teixeira Batista

diogocezar

Just a guy that knows that knows nothing.

Paraná - Brazil
diogo@diogocezar.com
http://www.diogocezar.com

Organizations

Pinned repositories

Customize your pinned repositories

dctb-web-project
Repositório informativo com diretrizes empíricas para o desenvolvimento de um Projeto Web.
★ 37 8

dctb-links
My Personal Links
★ 19 10

dctb-digital-ocean-for-zombies
Um pequeno tutorial de como criar um servidor web com U$ 5,00 por mês
★ 8 1

dctb-play-this
It is a system aiming to amuse and entertain people. Inspired by the old jukebox, a machine that reproduces music to insert coins, this system enables the choice of music that will be played on the...
PHP ★ 7 2

dctb-utfpr
Repositório para organizar os códigos apresentados nas disciplinas da UTFPR.
JavaScript ★ 1 4

dctb-firebase-chat
Simple Chat System with Firebase
JavaScript

293 contributions in the last year

Contribution settings

Contribution activity

Jump to

2017
2016
2015
2014

September 1, 2017

Created 3 commits in 3 repositories

diogocezar/utfware-hello-world 1 commit

diogocezar/utfware-local 1 commit

GITHUB

- na página de um projeto, temos:
  - o nome do usuário/organização a que o projeto pertence;
  - a opções de acompanhar (watch), dar estrela, ou fazer fork;
    - ao marcar como watching você será notificado sobre todas as modificações do projeto;
    - ao marcar a estrela, você está apoiando o projeto;
    - ao fazer um fork, você está literalmente “copiando” este projeto para seu usuário (é dessa forma que você conseguirá contribuir neste projeto);
  - temos também a descrição do repositório (textualmente)
  - e na parte inferior temos a exibição do *MarkDown* do arquivo README.md
    - é padrão que toda pasta mostre uma seção com MarkDown formatado que se encontra no readme.md

GITHUB

- ainda na página de um projeto, temos:
  - possuímos algumas abas para o projeto que são:
    - Code → o código fonte e a sua estrutura de pastas e arquivos;
    - Issues → é como se organiza em um projeto o que precisa ser feito; essencialmente são funcionalidades que devem ser implementadas no projeto;
    - Pull requests → é onde se organiza o que os colaboradores do projeto estão enviando para o seu projeto;
    - Projects → é uma forma visual de organização das suas funcionalidades no estilo kanban;
    - Wiki → é um fórum para solicitações e reports de bugs;
    - Settings → são as configurações do seu projeto;

diogocezar / dctb-web-project

Unwatch 6 Star 37 Fork 8

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Repositório informativo com diretrizes empíricas para o desenvolvimento de um Projeto Web.

Edit

Add topics

19 commits 1 branch 0 releases 1 contributor Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

diogocezar Update README.md Latest commit 39ab608 on 19 Mar 2016

LICENSE	Initial commit	2 years ago
README.md	Update README.md	2 years ago

README.md

Projeto para Web

Vou procurar listar neste repositório, uma checklist de ítems a serem cumpridos em um projeto web.

Essas métricas se aplicam a projetos simples, com curto prazo de desenvolvimento, mas ajudam a garantir o mínimo de qualidade para um projeto funcional.

É importante que a cada etapa, sejam gerados artefatos a serem armazenados de acordo com algum padrão. Pode ser com a ajuda de algum *software* ou até mesmo uma *planilha*. A forma com que estes dados serão organizados fica a critério de cada equipe.

A seguir os passos para a criação de um **Projeto Web**:

1. Briefing

Esta é a etapa em que se deve entender o negócio do cliente, suas necessidades, e junto com ele identificar:

1. Como é seu negócio? Como é o seu produto? Como será o seu serviço?
2. Qual será o objetivo do *website*?
3. Qual será o público-alvo? Quando possível identificar:
4. Classe Social;
5. Sexo;

GIT

entendendo o versionador de arquivos

ANTES DE TUDO

- o git funcional localmente em sua máquina;
- você pode utilizar um ou mais repositórios remotos para manter seus arquivos;
- o git não funciona apenas com GitHub, existem outros serviços, e você pode até configurar o seu próprio servidor Git;
- alguns outros exemplos de repositórios git:
    - https://gitlab.com/
    - https://bitbucket.org/

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;
- possivelmente você pode configurar chaves SSH para permitir uma autenticação automática com seu ambiente remoto Git;

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 → ele se encontra no status **(untracked)**
- quando se adiciona um arquivo:
  - `git add file.ext` → `adiciona um único arquivo`
  - `git add --all` → `adiciona todos arquivos que foram modificados`
- nesse momento ele passa para o estágio → **(stage)**

O BÁSICO DO GIT

- se houver uma alteração em qualquer um dos arquivos depois que houver um `git add`, ele passa para um outro estado **(modified)**
- pode-se observar os estados dos arquivos com o comando:
  - `git status`

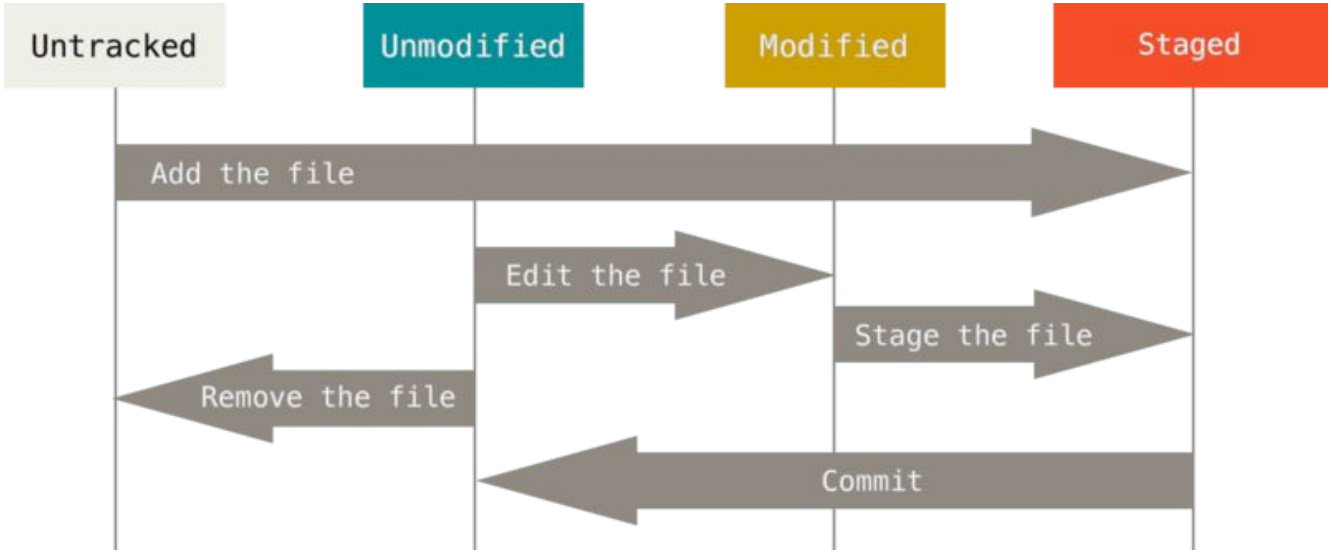
O BÁSICO DO GIT

- um **commit** só "leva" arquivos que estão no status **(stage)**;
- por isso, sempre que modificar um arquivo, deve-se adicioná-los novamente com o comando `git add`;
- nesse momento **(stage)** 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, mas é utilizado como convenção;
- 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
```

VAMOS PRATICAR

criando seus primeiros repositórios

PRÁTICA 1 - CLONANDO

- crie uma conta no **GitHub**;
- crie um novo repositório em sua conta, chamado “utfware-hello-world”;
  - marque “inicializar com readme”;
- na página de seu repositório clique no botão “Clone or download”
- copie o endereço;
- no seu terminal, com o git instalado e configurado, digite:

```
$ mkdir meus-repos
$ cd meus-repos
$ git clone git@github.com:diogocezar/utfware-hello-world.git
```

PRÁTICA 1 - CLONANDO

- os arquivos serão baixados para sua máquina;
- com seu editor de textos favorito, edite o arquivo README.md colocando um título → “Esse é meu primeiro repositório GIT”; e salve o arquivo;
- volte para o terminal e digite os comandos:

```
$ git add --all
$ git commit -m “Minha primeira contribuição”
$ git push
```

- volte em sua página do git, e veja as atualizações;

PRÁTICA 2 - LOCAL

- como segunda atividade, vamos criar um repositório localmente e depois ligá-lo ao GitHub;
- crie um segundo repositório chamado “utfware-local”

```
mkdir utfware-local
git init
```

PRÁTICA 2 - LOCAL

- crie um arquivo README.MD em seu diretório e inclua um texto livre, utilizando as formatações MarkDown;
- salve o arquivo e volte para o terminal com os seguintes comandos:

```
git add --all
git commit -m “Meu primeiro arquivo sem repositório definido”
git remote add origin git@github.com:diogocezar/utfware-local.git
git push -u origin master
```

- volte em sua página do git, e veja as atualizações;

OS BRANCHES

faça modificações fora do fluxo principal

BRANCHES

- o que é?
  - é um ponteiro para determinada *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 informações mais detalhadas: https://goo.gl/MQJGuw

BRANCHES

- mas... 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;

BRANCHES

- para criar um novo branch:
- primeiro tenha certeza que você está no *master*:
  - `git branch`
- cria-se uma nova *branch*
  - `git branch minha_branch`
- entra-se nesta branch
  - `git checkout minha_branch`

BRANCHES

- para navegar entre os branchs:
  - `git checkout minha_branch`
  - `git checkout master`
- para remover um branch
  - `git branch -D minha_branch`

RESETANDO

e agora... deu problema!

REVERTENDO OS SEUS ERROS

podem ocorrer situações na qual você precisará reverter o que já foi feito;

em um repositório qualquer podemos executar os comandos:

- `git log` ← para verificar os *commits*
- `git status` ← está vazio e eu não tenho nenhuma modificação
- `nano teste.txt` ← faz-se uma modificação que deverá ser revertida
- `git diff` ← para analisar as diferenças
- `git checkout teste.txt` ← retiras as modificações feitas (último commit)

REVERTENDO OS SEUS ERROS

e se eu já tiver adicionado?

- `git log` ← para verificar os *commits*
- `git status` ← está vazio e eu não tenho nenhuma modificação
- `nano teste.txt` ← faz-se uma modificação que deverá ser revertida
- `git add teste.txt` ← adiciona-se o arquivo ao git
- `git status` ← estado modified
- `git diff` ← não existem mais diferenças pq foi adicionado
- `git reset HEAD teste.txt` ← volta para opção editável
- `git checkout teste.txt` ← retiras as modificações feitas (último commit)

REVERTENDO OS SEUS ERROS

e como voltar caso isso já tenha sido commitado?

- `git log` ← para verificar os *commits*
- `git status` ← está vazio e eu não tenho nenhuma modificação
- `nano teste.txt` ← faz-se uma modificação que deverá ser revertida
- `git add teste.txt` ← adiciona-se o arquivo ao git
- `git commit -m "teste"` ← faço um commit
- `git log` ← mostra os commits e suas hashs

REVERTENDO OS SEUS ERROS

- três opções de reset:
  - --soft
    - mata o commit mas o arquivo estará em *staged*;
    - deixa o arquivo pronto para ser *commitado* novamente;
  - --mixed
    - mata o commit mas deixa os arquivos antes do *staged*;
    - será preciso dar o *add* novamente
  - --hard
    - mata o commit e todas as suas modificações

REVERTENDO OS SEUS ERROS

DICA → devemos escolher o commit anterior ao que queremos

- `git reset --soft <hash>`
- `git reset --mixed <hash>`
- `git reset --hard <hash>`

MERGE E REBASE

como juntamos as coisas

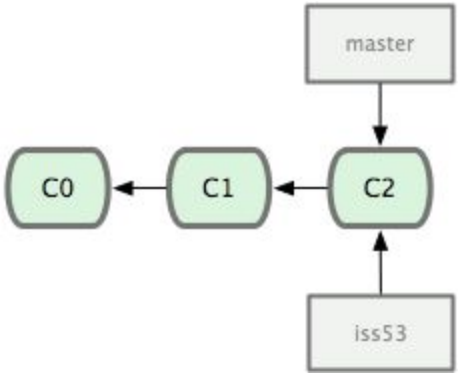

GIT MERGE

MERGE E REBASE

- depois que já criamos nossas funcionalidades em nosso branch separado, é hora de fazer a união entre os branchs;
- existem 2 métodos: merge e o rebase;
    - fazem basicamente a mesma coisa; de formas diferentes;
- o merge é interessante em *pull-requests* e novas *features*, pois sabemos exatamente de onde veio;
- o rebase é utilizado quando não é necessário saber quando a alteração foi feita (cronologicamente);
- ambos podem ser usados para juntar diversas funcionalidades na *timeline* principal de seu projeto;

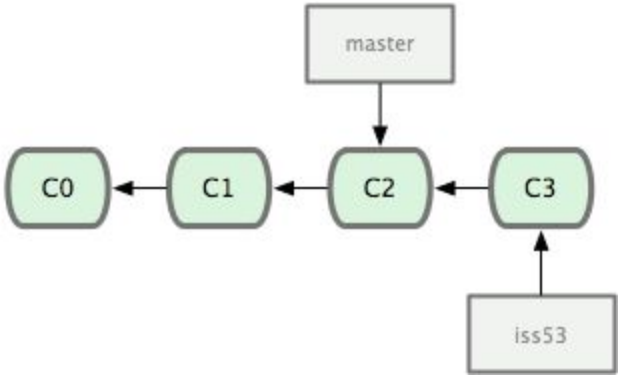
MERGE

MERGE



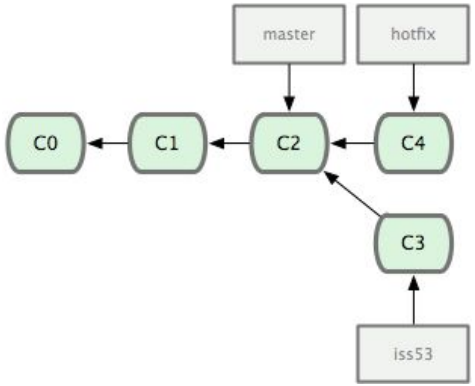
temos nossos *branches*: master e iss53 apontando para o mesmo *commit* (C2)

MERGE



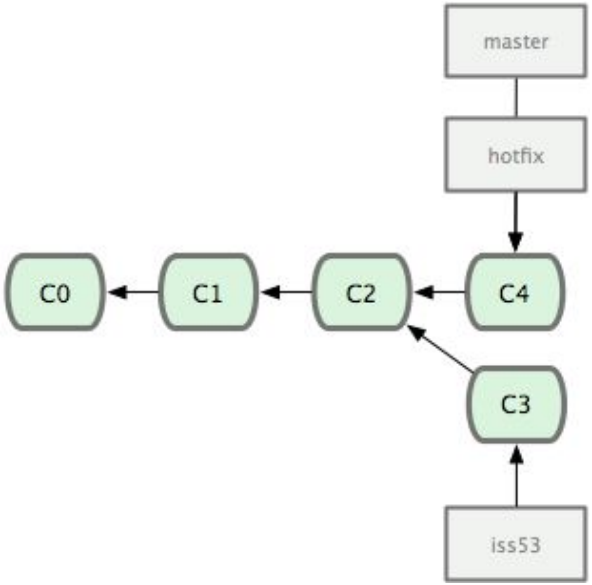
fizemos um commit na *branch* iss53 (C3)

MERGE



agora criamos uma outra *branch* chamada hotfix e um novo commit (C4).
neste momento... começamos a criar uma ramificação.

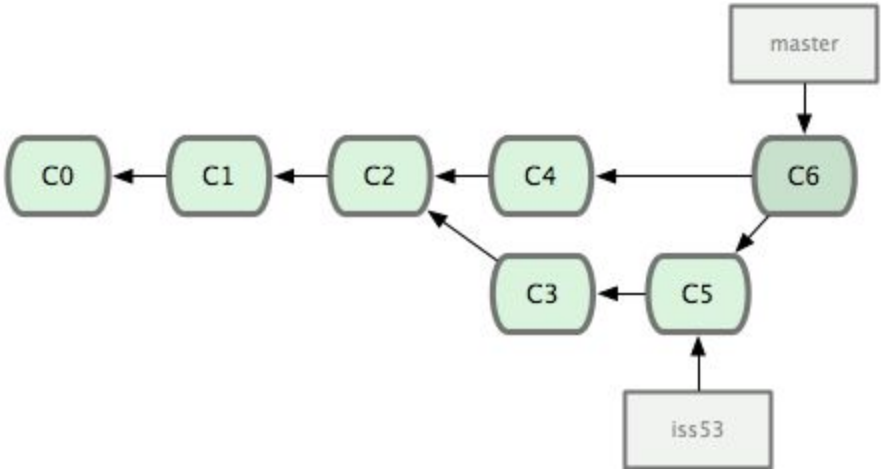
MERGE



utilizamos o comando **merge** para juntar o hotfix com o master

*um merge simples entre *commits* lineares que não causa problemas

MERGE

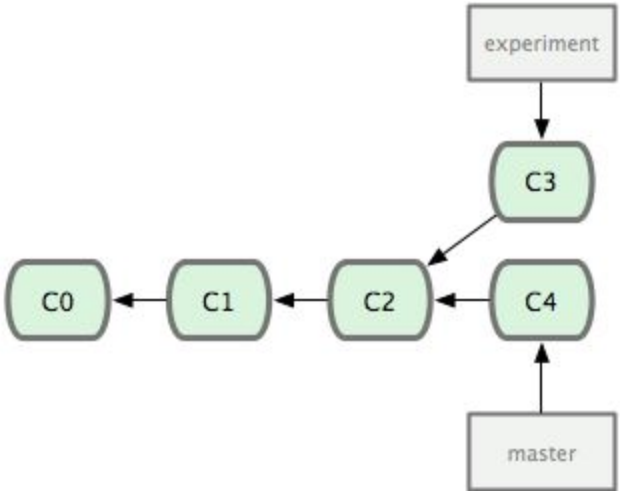


para juntar todo mundo:
com um **merge** eu crio um novo *commit* (C6) e é ele que junta as modificações de (C3) e (C5) com (C4) e deixa tudo linear novamente;

desvantagem → precisa-se criar um novo commit (C6);
vantagem → se mantém um histórico de qual branch a modificação veio

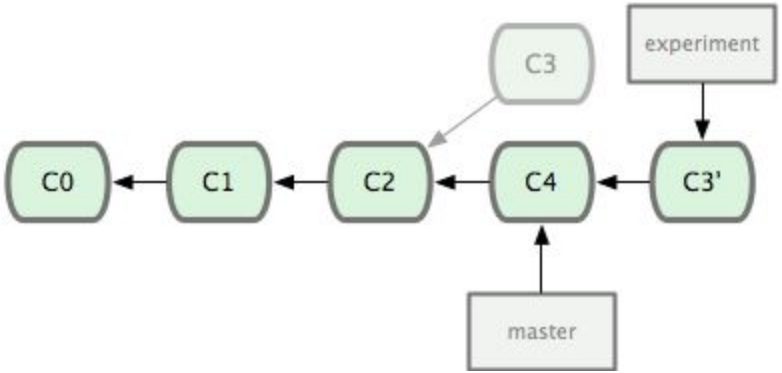
REBASE

REBASE



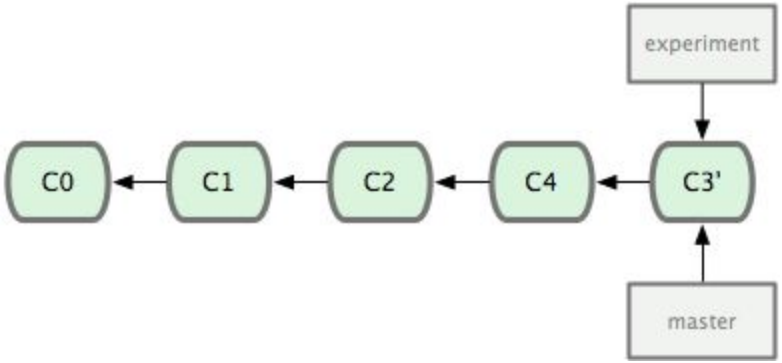
temos nossas *branches*: master e experiment apontando para diferentes *commits*

REBASE



o **rebase** move o *commit* (C3') para frente do *commit* a que o master aponta (deixando tudo linear);
pega tudo o que tiver na *branch* separada e coloca no início da fila

REBASE



tanto o master quanto experiment apontam para o novo commit criado (C3')

vantagem: evita os ciclos e mantenho a linearidade;
desvantagem: perde-se a ordem cronológica (não sei quem foi feito antes ou depois)

VAMOS PRATICAR

utilizando merge e rebase

PRÁTICA 3 - MERGE

CONSOLE	
`$ mkdir rebase-merge`	cria-se uma pasta rebase-merge
`$ cd rebase-merge`	entra na pasta criada
`$ git init`	inicia o repositório
`$ nano foo`	cria um arquivo com conteúdo “foo”
`$ git add foo`	adiciona o arquivo no git
`$ git commit -m “Add Foo”`	faz o commit de foo
`$ git checkout -b test`	crio e entro em um novo branch chamado test
`$ nano bar`	cria um arquivo com conteúdo “bar”
`$ git add bar`	adiciona o arquivo no git
`$ git commit -m “Add Bar”`	faz o commit de bar

PRÁTICA 3 - MERGE

CONSOLE	
`$ git log`	mostra os commits do branch test
`$ git checkout master`	volto para o branch master
`$ git log`	mostra os commits do master
`$ nano fizz`	cria um arquivo com conteúdo “fizz”
`$ git add fizz`	adiciona o arquivo no git
`$ git commit -m “Add Fizz”`	faz o commit de fizz
`$ git log`	mostra os commits de master (não tenho o bar)
`$ git merge test`	faz o merge (salve o arquivo que abrir)
`$ git log`	note que temos foo, bar e fizz e um commit para o merge
`$ git log --graph`	mostra o gráfico de commits

PRÁTICA 4 - REBASE

CONSOLE	
`$ nano buzz`	cria-se um arquivo com conteúdo buzz
`$ git add --all`	adiciona-se este arquivo no git
`$ git commit -m “Add Buzz”`	faz-se o commit do buzz
`$ git log`	veja que agora temos o commit buzz
`$ git checkout -b rebase-branch`	crio e entro em uma nova branch
`$ nano bla`	cria-se um novo arquivo chamado bla
`$ git add --all`	adiciona-se este arquivo no git
`$ git commit -m “Add Bla”`	faz-se um commit de bla
`$ git log`	temos buzz e bla
`$ git log --graph`	permaneço linetar no meu branch rebase-branch

PRÁTICA 4 - REBASE

CONSOLE

```
$ git checkout master
$ git log
$ nano poo
$ git add --all
$ git commit -m “Add Poo”
$ git log
$ git rebase rebase-branch
$ git log
$ git log --graph
```

volto para o meu branch master
vejo o log do master (não tem o bla)
cria-se um novo arquivo poo
adiciona-se poo no branch master
faz-se o commit de poo
temos buzz e poo
utiliza o rebase para juntar
veja que poo ficou no topo da lista
veja que se mantém linear

OS FORKS

uma forma de enviar suas contribuições

FAZENDO FORKS

- o que é?
  - é uma cópia de um projeto para sua conta do **GitHub**;
- isso é ideal para realizar contribuições! 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

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 https://github.com/diogocezar/utfware-forkme
- clonar o repositório para a sua máquina:
    - `git clone git@github.com:diogocezar/utfware-forkme.git`

PULL-REQUEST

- criando um *branch* → neste momento estamos criando um fluxo separado do principal, para que suas modificações não impactem diretamente no projeto;
- primeiro tenha certeza que você está no *master*:
  - `git branch`
- cria-se uma nova *branch*
  - `git branch add_my_name`
- entra-se nesta branch
  - `git checkout add_my_name`

PULL-REQUEST

- agora pode-se criar, editar e modificar os arquivos de acordo com sua necessidade;
- edite o arquivo README.MD e adicione o seu nome completo;

PULL-REQUEST

- enviando para o seu fork:
  - `git add --all`
  - `git commit -m "Adicionei o meu nome Diogo Cezar"`
- enviando a branch
  - `git push origin add_my_name`
- 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, uma descrição, e pronto!

PULL-REQUEST

- neste momento, seu pull-request foi enviado para o “dono” do repositório;
- cabe a ele analisar e aceitar ou não as suas modificações;
- se suas modificações forem aceitas, elas serão combinadas com os arquivos originais, e um merge poderá ser utilizado para deixar o arquivo consistente;

PRÁTICA 5 - PULL-REQUEST

- faça o fork do repositório → https://github.com/diogocezar/utfware-forkme
- clone o repositório que fez o fork em sua máquina;
- crie um novo branch chamado → meu_nome;
- edite o arquivo README.MD adicionando o seu nome na última linha;
- commite as modificações;
- suba para o repositório do git;
- abra seu painel de administração e envie o pull-request;

HOSPEDANDO SUAS PÁGINAS NO GITHUB

para que pagar servidor?

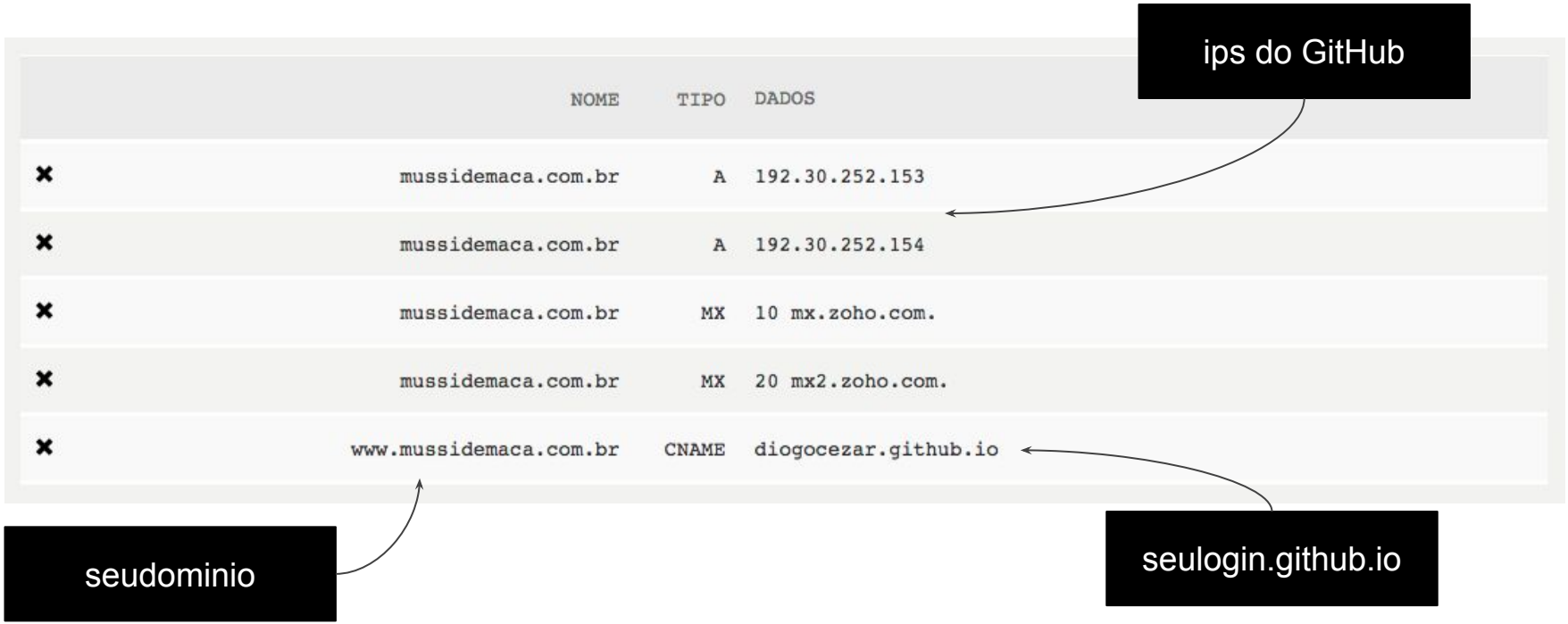
HOSPEDANDO SUAS PÁGINAS NO GITHUB

- bom, nem tudo são flores... só é possível hospedar arquivos estáticos: HTML, CSS e JS;
- vamos a uma passo a passo de como fazer isso!
- passo 1: crie um repositório (isso deve ser fácil nessa altura do campeonato)
- passo 2: vá até settings > GitHub Pages;
- passo 3: selecione um branch (você pode usar o master);
- feito isso sua página já pode ser vista em: seuuser.github.io/seu-repositorio;

HOSPEDANDO SUAS PÁGINAS NO GITHUB

- mas podemos ainda definir um domínio para apontar para o GitHub;
- para isso, devemos criar um arquivo chamado CNAME na raiz do seu repositório com o domínio em questão;
- depois disso, deve-se apontar para os servidores do GitHub no gerenciador do seu domínio;
- e pronto. temos um site grátis para sempre!

HOSPEDANDO SUAS PÁGINAS NO GITHUB



INTERFACES VISUAIS

chega de linha de comando


GAME DAY

INTERFACES VISUAIS

- entender os conceitos, os comandos e seus efeitos é essencial para um programador dominar o sistema de versionamentos GIT;
- com os comandos você é capaz de gerenciar todo o ambiente diretamente em um servidor remoto de produção, por exemplo;
- mas, existem ferramentas que facilitam a gerência do git no seu dia-a-dia:
  - https://git-scm.com/downloads/guis → recomendações de interfaces
  - https://tortoisegit.org/ → um que utilizei e recomendo (*free*);
  - https://desktop.github.com/ → o próprio github tem um aplicativo

SITES LEGAIS

que tal aprender git no próprio terminal

SITES LEGAIS

- com este link → https://goo.gl/uT3gLR
- você consegue cumprir uma série de tarefas aprendendo Git sem sair do seu navegador!

SITES LEGAIS

- neste outro link → https://goo.gl/daxaCV
- também temos tarefas gamificadas com comandos a serem digitados no próprio navegador para atingir os objetivos de cada tarefa;

MATERIAIS COMPLEMENTARES

seguem algumas outras fontes

MATERIAIS COMPLEMENTARES

- todo o material oficial do *git scm* → https://goo.gl/wkGwnh
- um ótimo curso gratúito na *Udemy* → https://goo.gl/7cUJLw
- curso básico de git no *YouTube* → https://goo.gl/qTu5RQ
- curso da Loiane sobre git e GitHub no YouTube → https://goo.gl/sy5s4g

THAT’S ALL FOLKS