Introdução ao

Bem vindos!

Quem é você?

Walmyr Carvalho

@walmyrcarvalho

Professor de Desenvolvimento para Plataformas Mobile

Instituto Infnet

Desenvolvedor Android

Hotel Urbano

O que vamos ver hoje?

- Introdução
- Breve histórico de criação do git
- Características
- Comandos básicos e seus conceitos
- Não gosta de Terminal? Apps com GUI!
- Links úteis: Livros, cursos e material de estudo

Histórico

Quem criou o git?

Linus Torvalds

Criador do kernel Linux

Criou o Git em 2005

Porquê?

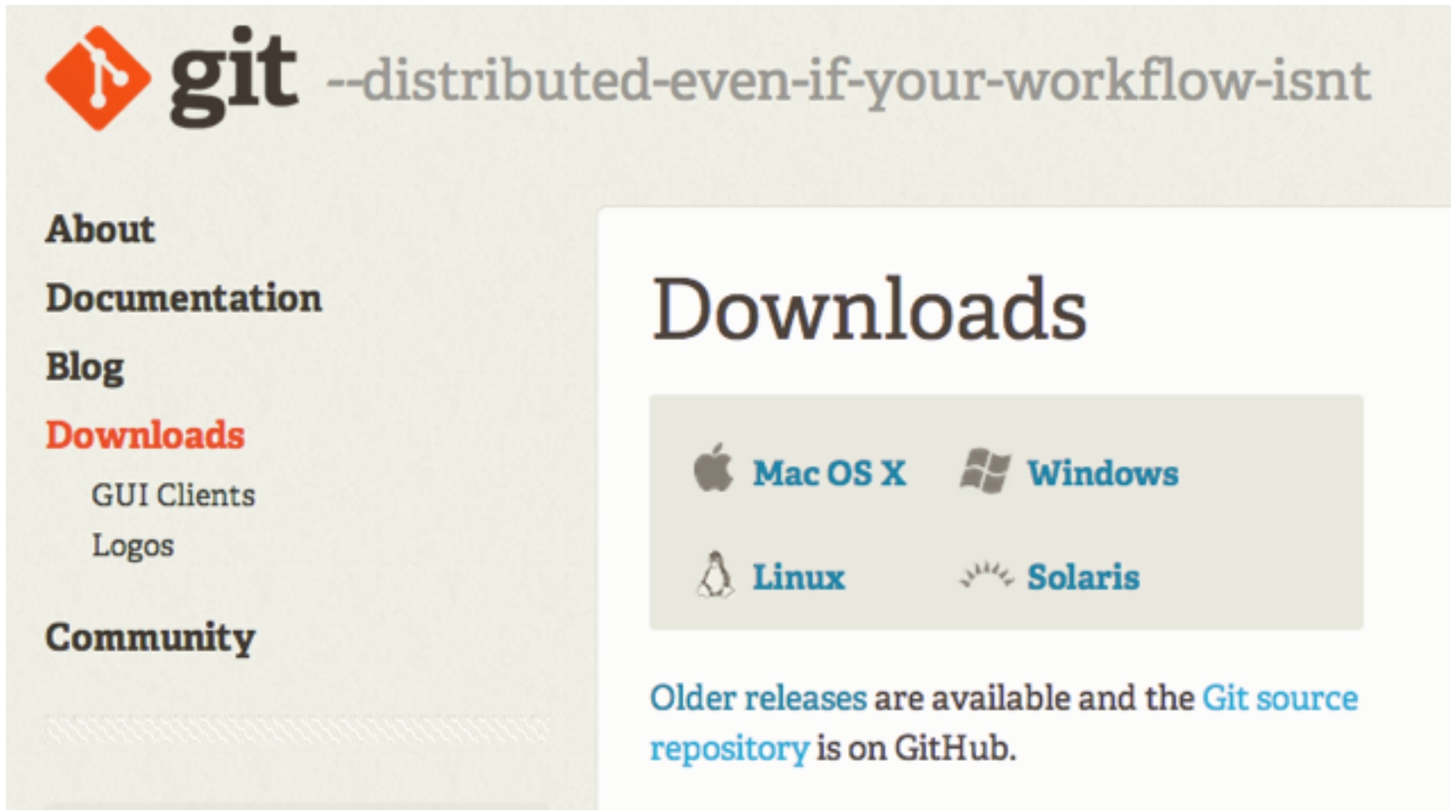
A criação do git começou durante o desenvolvimento do Linux, nenhum sistema de controle de versão tinha o desempenho ideal para lidar com o desenvolvimento não linear do projeto, com diversos desenvolvedores no mundo inteiro!

Por não encontrar nenhum outro sistema rápido, seguro e eficiente o suficiente para gerenciar um projeto grande como o Linux, Linus criou o próprio.

Características do git

- Sistema de controle distribuído
- Menos suscetível a erros
- Funcionamento offline
- Projetos menores que o SVN
- Área de preparo (staging)
- Muito rápido!
- Sistema de branches simples e eficiente

Instalação e comandos básicos



Antes de começar, é preciso instalar o git na sua máquina. Faça o download para o seu sistema operacional no link abaixo:
git-scm.com/downloads

Criando um repositório

Ok, agora temos o git instalado! Para iniciar um repositório git no seu projeto, acesse a raiz da pasta do projeto e execute o comando:

```
git init
```

Clonando um repositório

Caso você queira fazer uma cópia de um repositório local, execute o comando:

```
git clone caminho/do/repositorio
```

Se o projeto estiver em um repositório remoto, execute o comando:

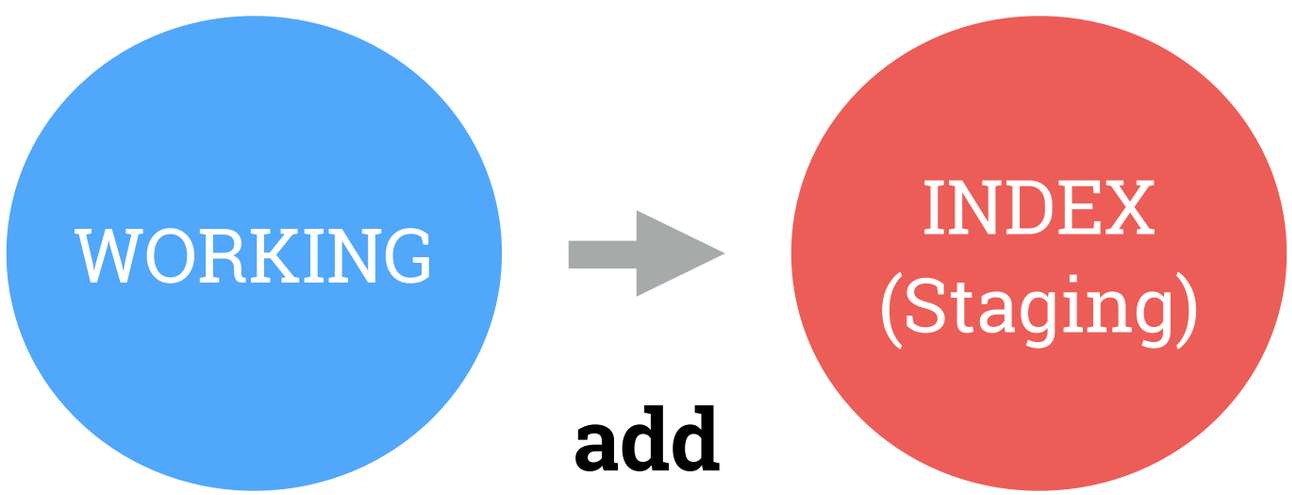
```
git clone
usuario@server:/caminho/do/repositorio
```

Workflow do Git

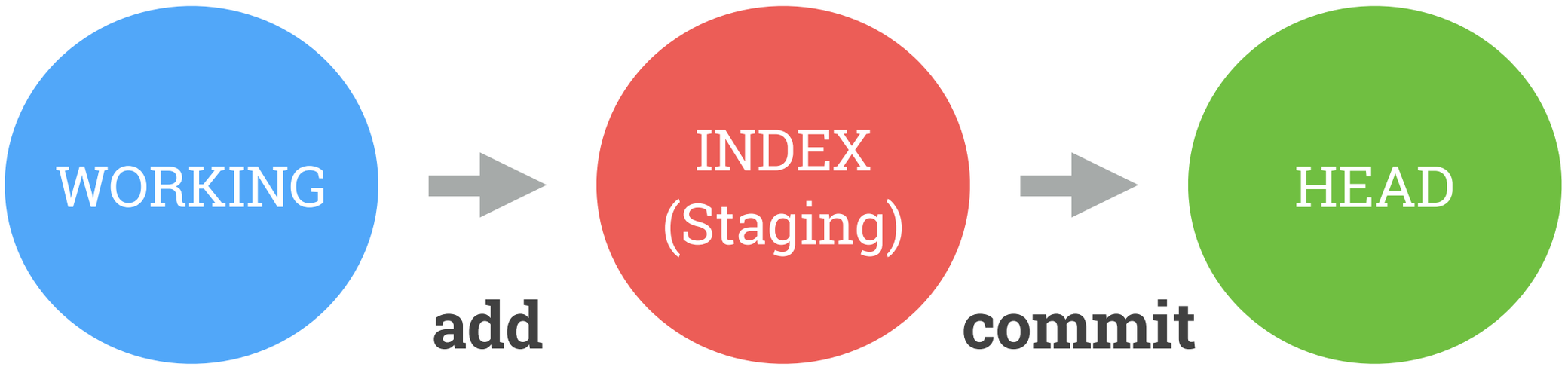
Quando trabalhamos com git, temos em mãos três “árvores" de trabalho no nosso fluxo local de operações.

A primeira delas é o diretório de trabalho (também chamado de **working tree**), que é onde ficam os arquivos do seu projeto.

Quando você adiciona algum arquivo alterado (**git add**), você envia as suas alterações para o **Index**, que é como uma área de preparação (**staging**) das suas alterações.



E por fim, quando você confirma uma nova alteração (**git commit**), você envia ela para a área de **HEAD**, que é onde está a última confirmação que você fez.



Adicionando e commitando arquivos

Para adicionar novas alterações no **Index**, execute o comando:

```
git add nome_do_arquivo
```

Caso queira adicionar todas os arquivos alterados do seu repositório:

```
git add *
```

O comando **add** é a primeira coisa que fazemos em um arquivo alterado, mas agora precisamos confirmar essa alteração. Para isso, usamos o comando abaixo:

```
git commit -m “comentário"
```

O **commit** é o que confirma a sua alteração e a envia para **HEAD**.

Enviando alterações

Agora que suas alterações já estão no **HEAD**, basta enviá-las para o seu repositório remoto. Para isso, execute o comando:

```
git push origin master
```

O **master** no caso é a sua **branch** atual, altere para a que desejar.

Caso você queira se conectar a um repositório remoto, basta adicioná-lo com o comando:

```
git remote add origin local_do_servidor
```

O que é uma branch?

Uma **branch** (ramo ou galho, em inglês) é uma ramificação utilizada quando se deseja criar uma nova funcionalidade isolada de outras partes do projeto.

Por exemplo: Esse é o **master**, é a branch padrão de um projeto git. Durante o projeto vamos precisar criar novas funcionalidades para ele.

Quando eu quero desenvolver uma nova funcionalidade, eu crio uma nova **branch** com o comando:

```
git checkout -b nova_branch
```

O comando acima é um atalho para executar dois comandos:

```
git branch nova_branch
git checkout nova_branch
```

Ou seja, o que você acabou de fazer foi criar uma **branch** paralela ao **master,** mas com sua própria timeline de alterações e commits, que deve ser mesclada ao **master** quando for concluída.

Ok, feita a nova funcionalidade, precisamos voltar para o **master** para mesclá-la com a nossa branch principal. Para isso, execute o comando:

```
git checkout master
```

Caso você queira deletar sua **branch**, utilize o comando:

```
git branch -d nova_branch
```

Lembrando que uma **branch** não está automaticamente disponível para todos que tem acesso ao seu repositório, é preciso adicioná-la usando o comando:

```
git push origin nova_branch
```

Ok, depois de criarmos uma **branch** nova e ter desenvolvido nossa nova funcionalidade, precisamos mesclá-la (**merge**) ao nosso **master.**

Mas antes de fazer o **merge** de uma nova **branch** ao **master**, precisamos atualizar o nosso repositório puxando possíveis novos **commits** do nosso repositório remoto. Para isso, utilize o comando:

```
git pull
```

Agora sim, temos o nosso repositório atualizado, pronto para receber a nossa **branch**. Para fazer o **merge** dela ao nosso **master**, utilize o comando:

```
git merge nova_branch
```

Lembrando que uma **branch** geralmente não vive por muito tempo, ao menos que a feature for muito grande. Nesses casos, o ideal é dividir a feature em **branches** menores.

Resolvendo conflitos

Infelizmente, durante o **merge** conflitos podem acontecer. É preciso editar os arquivos e resolvê-los manualmente.

Existem diversas ferramentas gráficas que facilitam essa resolução de conflitos, comumente chamadas de **merge tools** ou **diff tool.**

Caso você queira ver a diferença entre arquivos de **branches** diferentes, basta executar o comando:

```
git diff <branch_origem> <branch_destino>
```

Tags

Durante um projeto de software, é importante se criar **tags** para releases de lançamento.
Para criar uma **tag** no git, usamos o comando:

git tag 1.0.0 1b2e1d63ff

O 1b2e1d63ff é composto pelos 10 primeiros caracteres do id do commit que você quer referenciar na sua **tag**.

Você pode conseguir uma lista de ids de commits utilizando o comando:

```
git log
```

Ele lista todos os commits do projeto, como um histórico detalhado do que foi feito em cada um deles.

Links úteis

Serviços web

GitHub

github.com

BitBucket

bitbucket.com

GitLab

Sistema similar ao git, porém open source

www.gitlab.com

Aplicativos com GUI (Interface Gráfica)

GitHub

(OS X / Windows)

Gratuito!

mac.github.com/

windows.github.com/

SourceTree

(OS X / Windows)

Gratuito!

www.sourcetreeapp.com

Tower

(OS X)

US$ 60,00 :(

www.git-tower.com

Git Cola

(Linux)

Gratuito e open source!

git-cola.github.io/

RabbitVCS

(Linux)

Gratuito e open source!

rabbitvcs.org/

Projetos usando git

Node GH

GitHub no terminal, por Zeno Rocha

nodegh.io

Hub

Outro wrapper para GitHub no Terminal

hub.github.com

Livros e cursos gratuitos

Git - Guia Prático

por Roger Dudler, guia open source disponível em português, fonte principal de conteúdo da palestra!

rogerdudler.github.io/git-guide

git cheat sheet

learn more about git the simple way at *rogerdudler.github.com/git-guide/*
cheat sheet created by Nina Jaeschke of *ninagrafik.com*

create & clone

create new repository	*git init*
clone local repository	*git clone /path/to/repository*
clone remote repository	*git clone username@host:/path/to/repository*

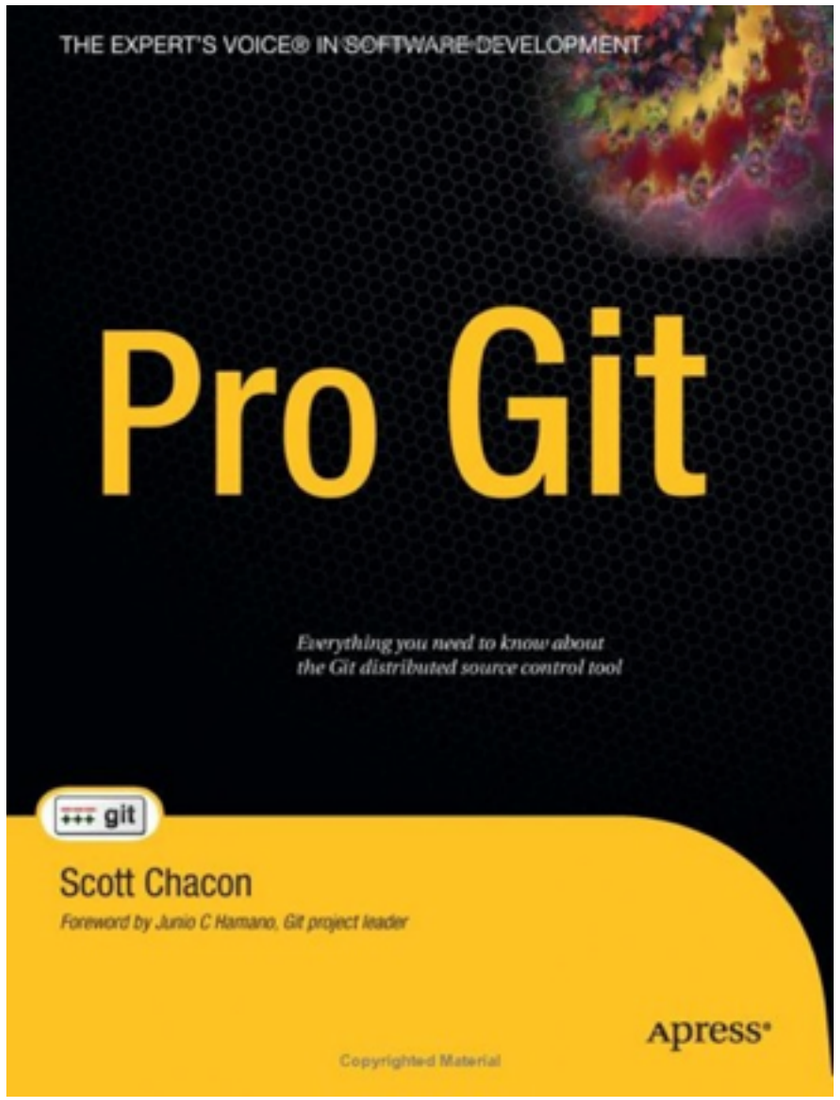
add & remove

add changes to INDEX	*git add <filename>*
add all changes to INDEX	*git add **
remove/delete	*git rm <filename>*

Git - Cheat Sheet

por Roger Dudler, lista de comandos mais comuns para referência rápida, em inglês

rogerdudler.github.io/git-guide/files/git_cheat_sheet.pdf



Pro Git

Livro oficial do projeto Git, disponível em português

git-scm.com/book/pt-br

Learn Version Control with Git

A step-by-step course for the complete beginner

Curso - Git Tower

Livro, vídeos e treinamento

www.git-tower.com/learn

Curso - Try Git | Code School

Curso disponibilizado online gratuitamente, parceria do Code School com o GitHub.

try.github.io

Dúvidas?

Eu ajudo, falem comigo!

Twitter: @walmyrcarvalho
Facebook: Walmyr Carvalho
Google+: +WalmyrCarvalho

Obrigado! :)