Minicurso de Gerenciamento e Compartilhamento de Software Utilizando Git/GitHub

I Semana da Informática

Charles Tim Batista Garrocho

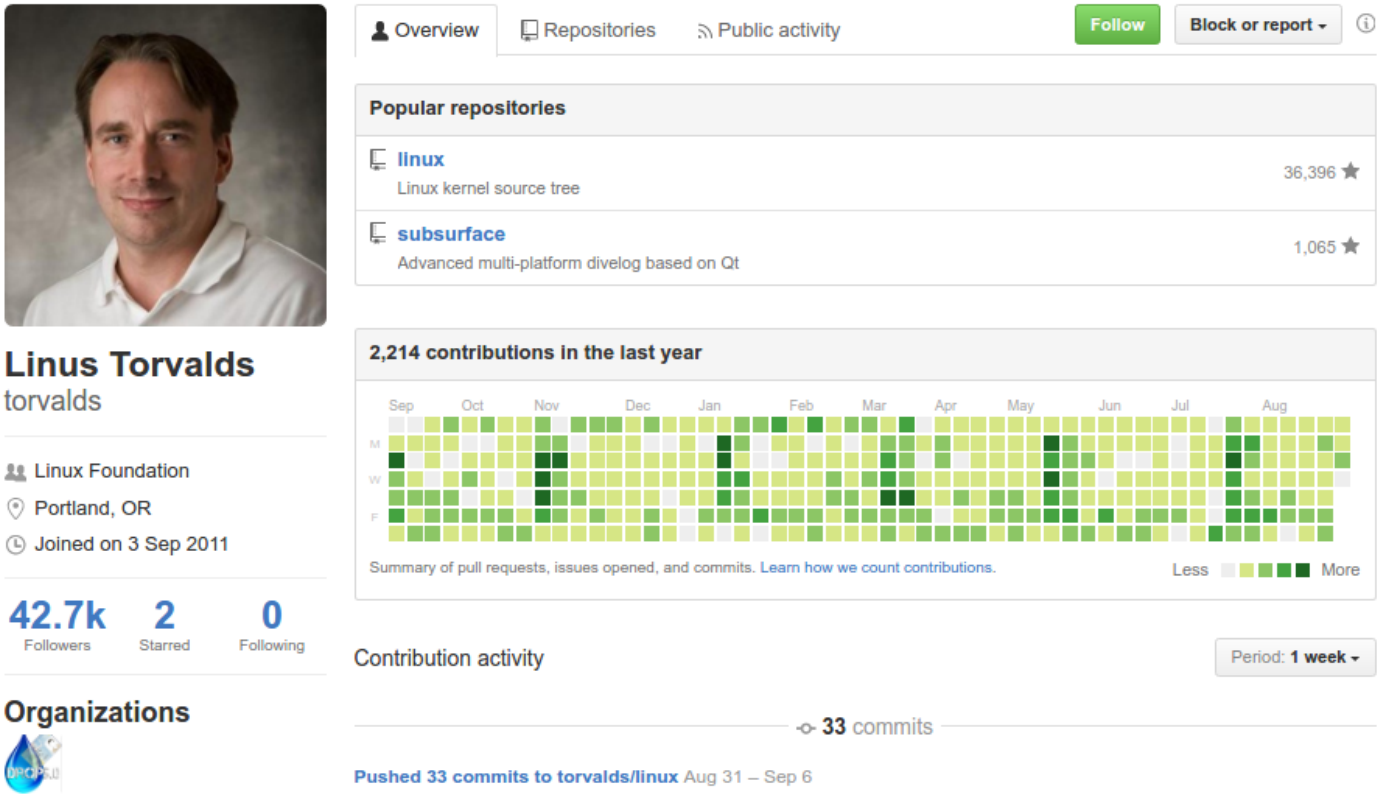
Instituto Federal do Paraná – IFPR
Campus Avançado Goioerê

charles.garrocho@ifpr.edu.br

13 de Setembro de 2016

Motivação

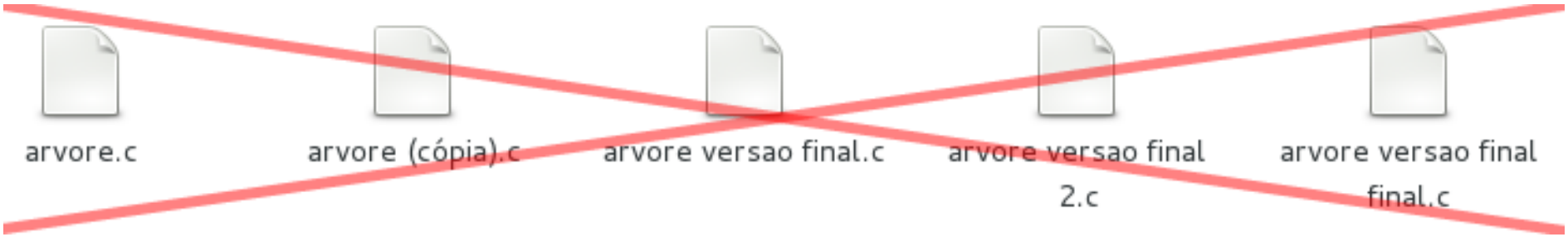
Linus Torvalds estava insatisfeito com o BitKeeper, ferramenta de controle de versão que ele utilizava para desenvolver o kernel do Linux.



github.com/torvalds

Por que utilizar um controle de versão?

Gerenciar manualmente as versões do seu software gera diversos problemas: perda de informações, cópia desnecessária de arquivos, falta de organização de estrutura.



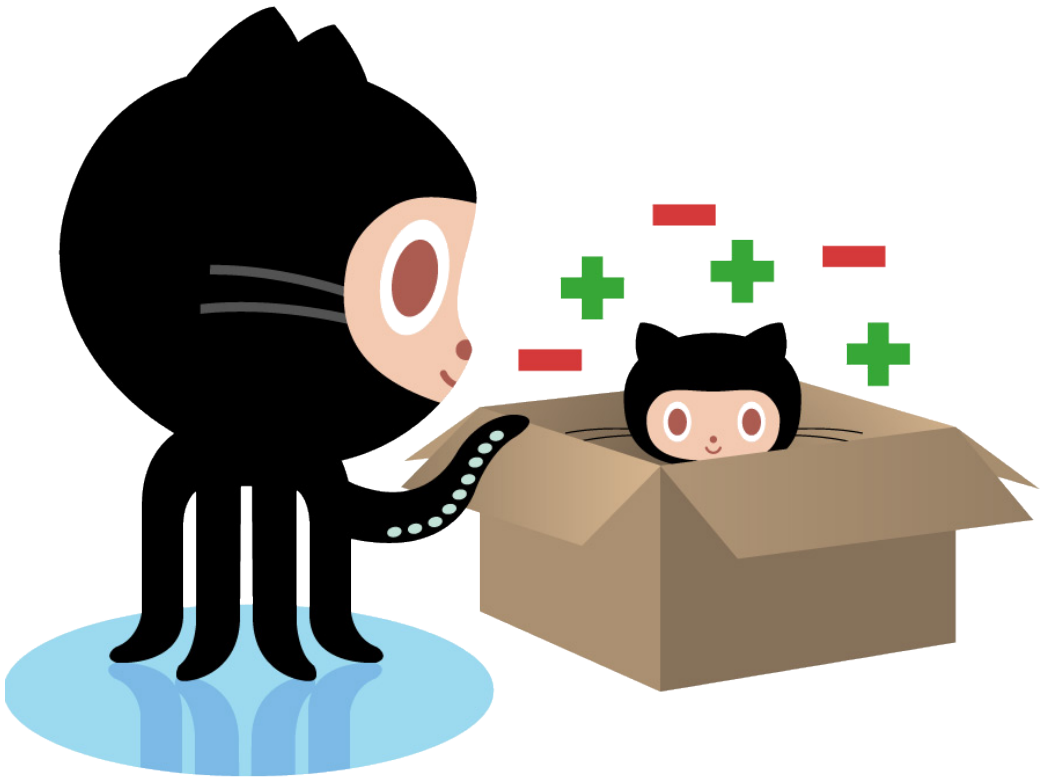
Utilizando um sistema de controle de versão você irá organizar o desenvolvimento do software.

Você terá um histórico de tudo que foi modificado em seu software, e poderá voltar em um estado a qualquer momento.

Organizar o desenvolvimento de software

O Git oferece controle total do projeto ao desenvolvedor para:

- Visualizar as mudanças ocorridas em cada arquivo;
- Visualizar o estado do projeto em etapas anteriores;
- Desfazer mudanças;
- Desenvolver funcionalidades em paralelo.

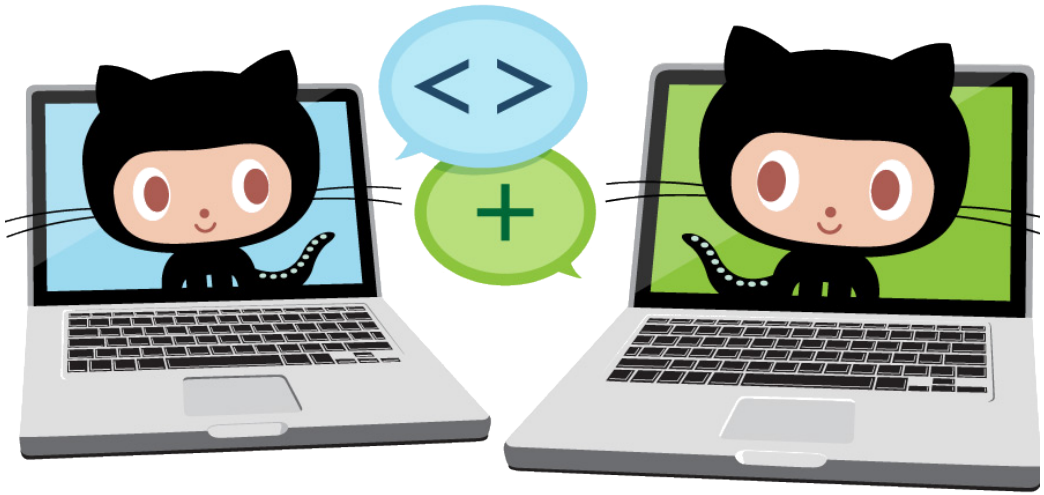


INSTITUTO FEDERAL

Compartilhar Projetos

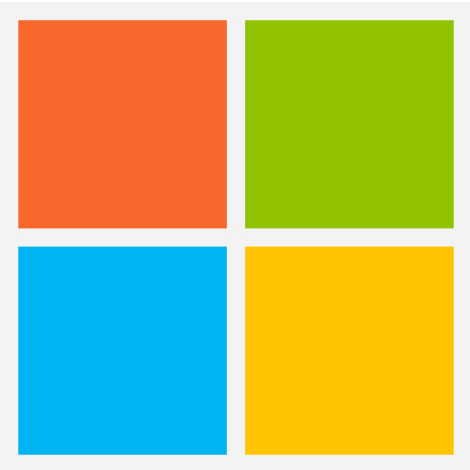
Desenvolver projetos colaborativos nem sempre é fácil.

Utilizar DropBox, pen drives ou afins para compartilhar código muitas vezes resulta em dor de cabeça.



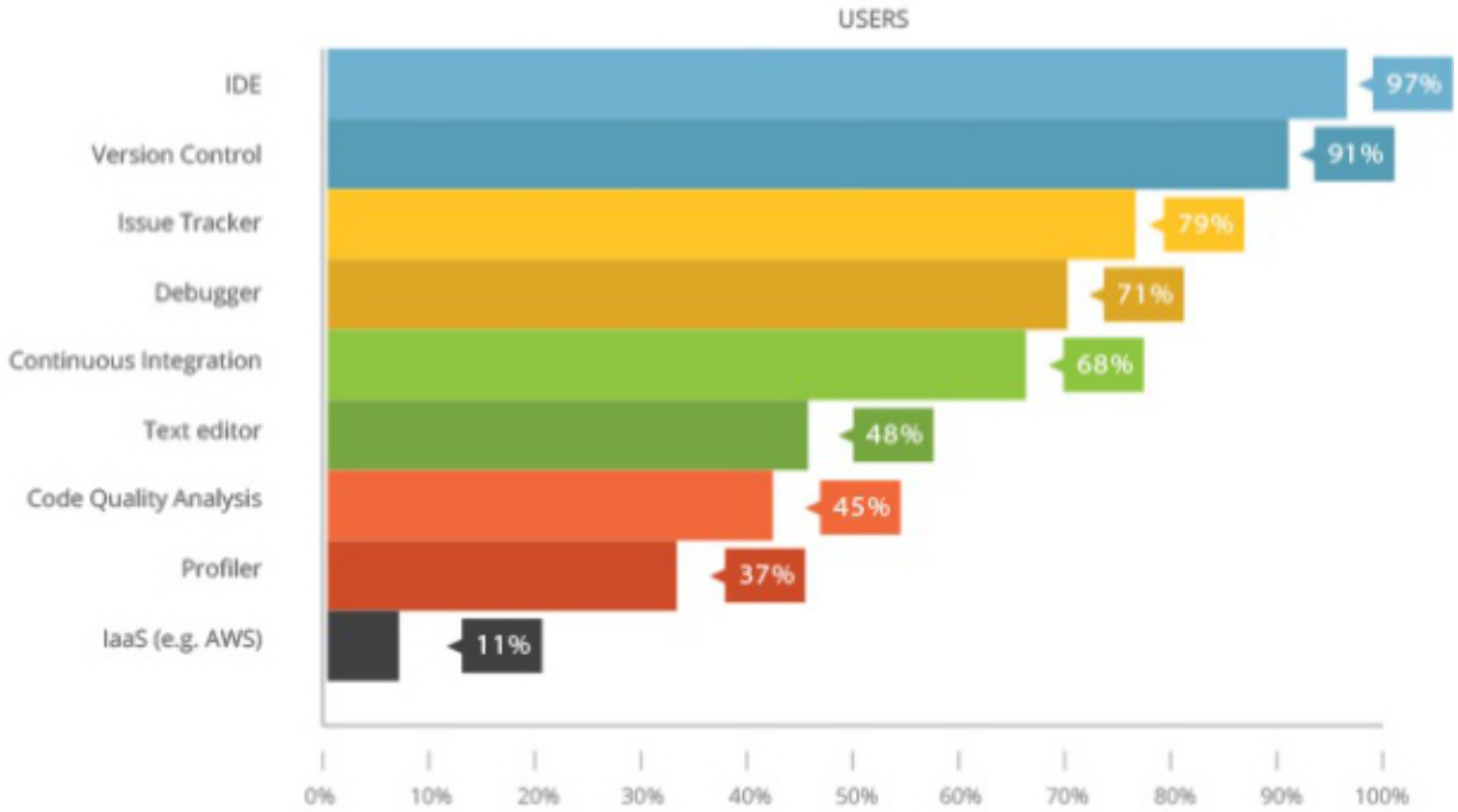
INSTITUTO FEDERAL

Independe da plataforma



O mercado utiliza!

Desenvolvimento profissional é feito assim :)



Rebellabs

INSTITUTO FEDERAL

Quem usa o Git?

Atualmente, a maioria das grandes empresas de desenvolvimento utilizam o Git!

`http://git-scm.com/`

Vagas de Emprego

Apple:

- Experience with subversion, GIT - branching, and merging

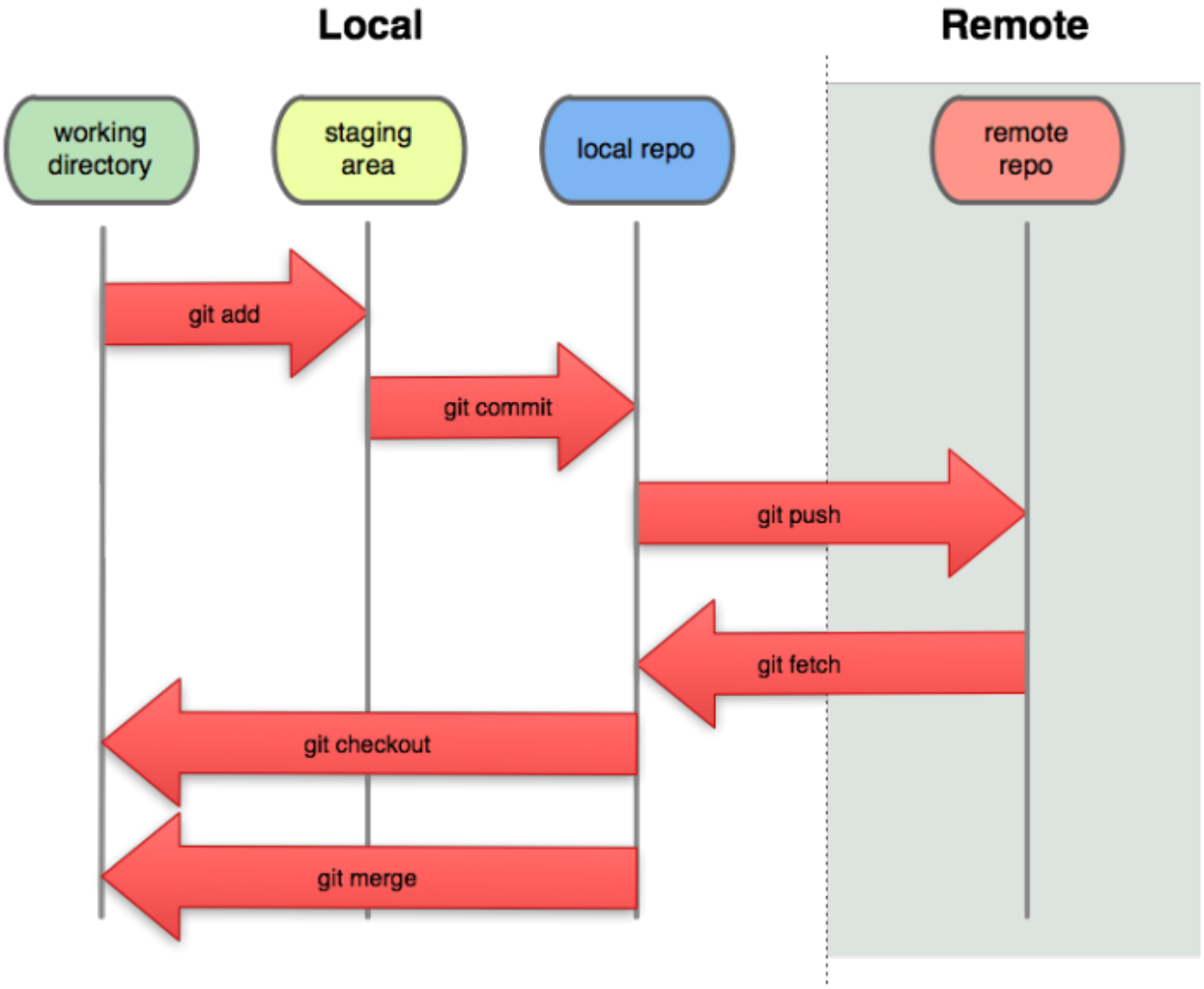
Blizzard:

- Experience with code/asset repositories such as SVN, Git,Perforce, TFS, and Alienbrain

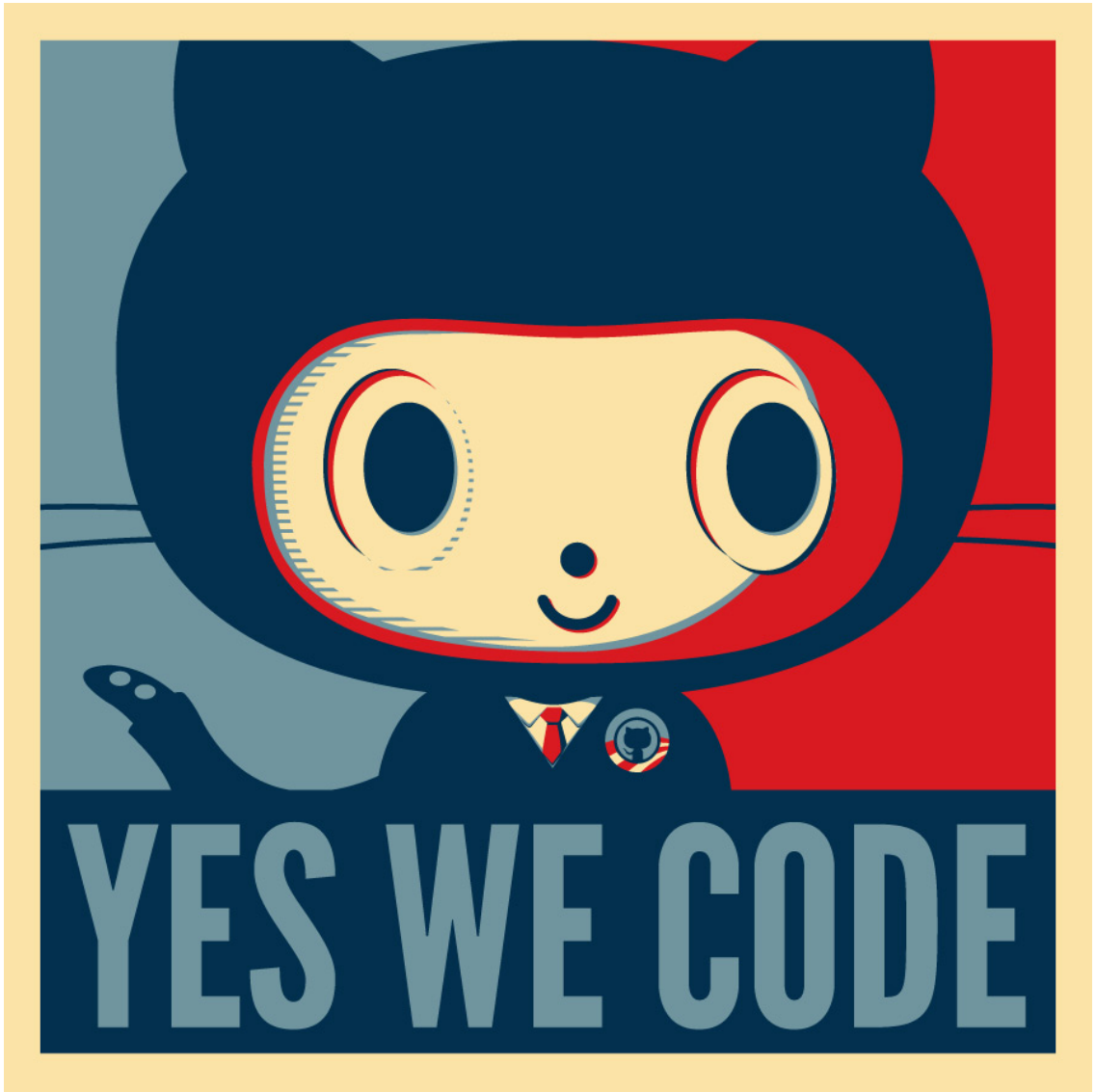
Android/Google:

. You will be responsible for everything from maintaining Google Docs and Sites aimed at Android partners to developing API references and managing source HTML in a Git repository.

Como Funciona o Git: Armazenamento



Vamos à prática!



INSTITUTO FEDERAL

Identificação

```
git config --global user.name "Nome"
git config --global user.email email
```

Listando as Configurações

```
git config --list
```

Para iniciar um repositório Git

```
git init
```

Este comando cria toda a estrutura que o Git necessita para funcionar. Os arquivos são criador na pasta oculta **.git/**. A partir de agora você pode desenvolver seu projeto sob controle do Git.

Verificando o estado do projeto

```
git status
```

O *git status* exibe as alterações ocorridas no repositório desde o último commit.

Adicionando arquivos

```
git add arquivo
git add .
git add diretório
```

O *git add* adiciona ou atualiza um arquivo da staging area. Ou seja, o comando informa ao Git para rastrear o referido arquivo. Caso o arquivo já esteja sob controle do Git, ele o atualiza.

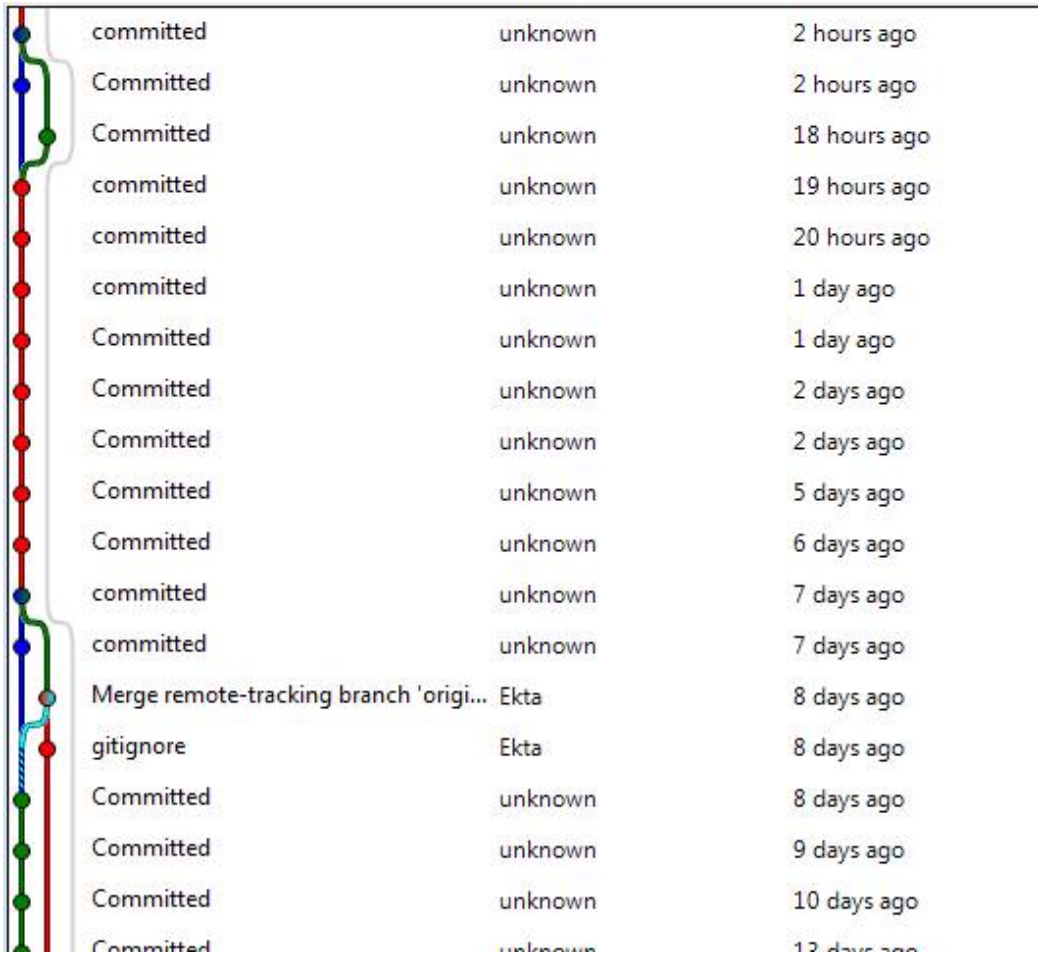
Confirmando mudanças

```
git commit -m "descrição do commit"
```

git commit transfere o estado do projeto salvo na staging area para o repositório do projeto. Simplificando, ele confirma as suas modificações.

Importância da mensagem do commit

Todo commit **deve** possuir uma mensagem de identificação, usada para descrever as alterações do commit. Favor, use-a para tal fim.



Exemplo de como não se fazer um commit.

Comparando alterações

```
git diff arquivo
git diff id_commit
```

O comando *git diff* compara o estado do repositório atualmente com o estado salvo na staging area

Histórico de alterações

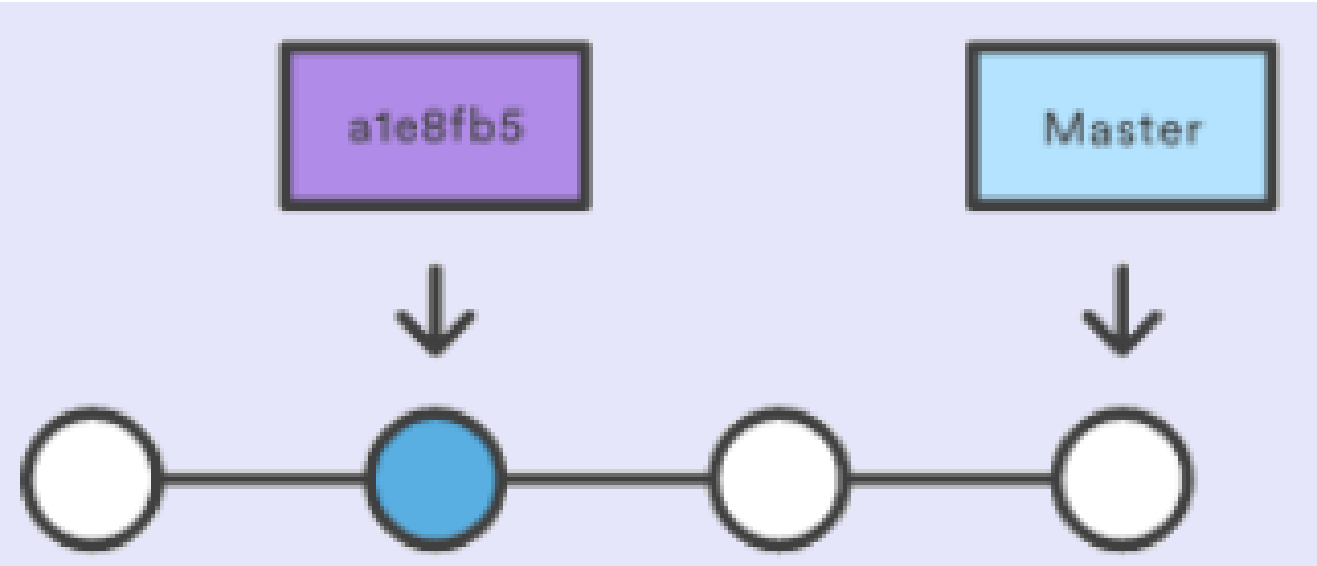
```
git log
```

O comando *git log* exibe o histórico de commits do projeto

'Passeando' pelo projeto

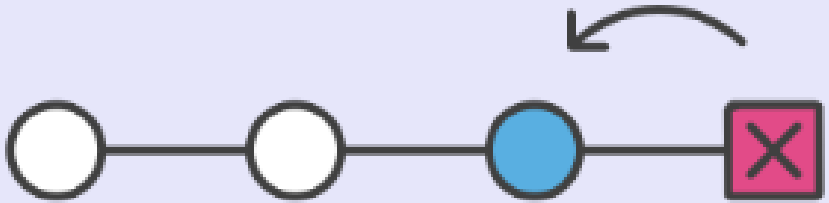
A partir do código associado aos commits você pode voltar para um determinado 'estado' do projeto.

```
git checkout id_commit
```



Desfazendo modificações

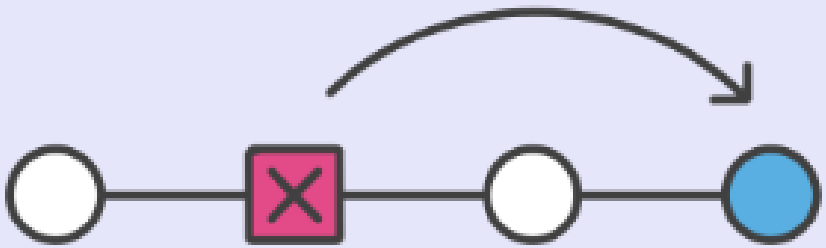
```
git reset id_commit
git reset arquivo
git reset --hard id_commit
```



O comando *git reset* faz o projeto/arquivo voltar para um estado anterior.

Revertendo modificações

```
git revert id_commit
```



O comando *git revert* não desfaz um commit, mas cria outro removendo a alteração anterior.

Removendo arquivos

O comando *git rm* remove um arquivo do seu projeto.

```
git rm arquivo
git rm --cached arquivo
```

A opção *–cached* remove o arquivo apenas do Git, sem esta opção o comando remove o arquivo do seu computador também.

Ignorando arquivos

Deixar todos os arquivos contidos na pasta do projeto sob controle do Git pode acabar dificultando o gerenciamento, e muitas vezes simplesmente não achamos necessário rastrear alterações de alguns arquivos (arquivos temporários, executáveis e etc...).

```
.gitignore
```

O arquivo oculto *.gitignore* é o responsável por dizer ao Git quais arquivos ele pode ignorar.

Trabalhando com repositórios remotos

Muitas empresas possuem servidores dedicados para suas equipes, mas há algumas soluções no mercados que cumprem a mesma função.

Mas daremos foco ao case mais bem sucedido.

INSTITUTO FEDERAL

Referenciando um repositório remoto

O comando *git remote add* é o responsável por referenciar um repositório remoto em um repositório local já existente.

```
git remote add origin URL
```

O comando *git remote rm* é utilizado para apagar uma referência à um repositório remoto. Util quando se deseja substituir o local onde o remoto está hospedado.

```
git remote rm origin
```

Clonando repositório

Quando se deseja iniciar/continuar um projeto já existente, utiliza-se do *git clone* para copiar todo um repositório remoto para uma máquina local.

```
git clone URL
```

Enviando conteúdo para o repositório remoto

```
git push
git push origin master
git push origin nome_branch
```

Utiliza-se o *git push* para enviar as modificações para o repositório remoto.

Baixando atualizações do remoto

É utilizando o comando *git pull* para baixar e incorporar as modificações no repositório local.

```
git pull origin master
```

Equivale à um *git fetch* seguido de uma *git merge*.

Criando versões

Conforme incluímos funcionalidades no sistema, podemos definir pontos relevantes no software, que geralmente, marcam uma versão de lançamento. O comando *git tag* inclui um rótulo à um determinado commit para que este possa ser referenciado mais facilmente.

```
git tag -a versao -m "descrição"
```

Para visualizar as versões ja criadas, utilize o comando *git tag* sem parâmetros.

```
git tag
```

Criando versões

Para visualizar o conteúdo de uma versão lançada, utilize o comando *git checkout*.

```
git checkout nome_tag
```

Para enviar os rótulos criados para o repositório remoto, utilize o parâmetro *–tags* no comando *git push*.

```
git push origin --tags
```

Atividade Prática

Primeiro Passo:

1. Crie seu perfil no GitHub e siga pelo menos um membro do minicurso.
2. Crie um repositório no seu perfil do GitHub. Adicione um título e uma descrição ao repositório. Gere o README.md e o .gitignore.
3. Adicione membros ao seu repositório através do email de seu colega.
4. Crie tarefas (issues) neste repositório e associe essas tarefas com os membros adicionados no repositório.

Segundo Passo:

1. Provavelmente alguém te adicionou em um repositório. Assim, crie uma pasta local, inicialize o git, e adicione como respositório nesta pasta o endereço do repositório que te adicionaram como membro.
2. Baixe as informações desse repositório. Crie uma pasta com seu nome, e adicione dentro dessa pasta um arquivo contendo seu nome e sobrenome. Submeta essas atualização adicionadas por você ao repositório que você é membro.