Treinamento Organizacional

Git Hub

O que é Controle de Versão?

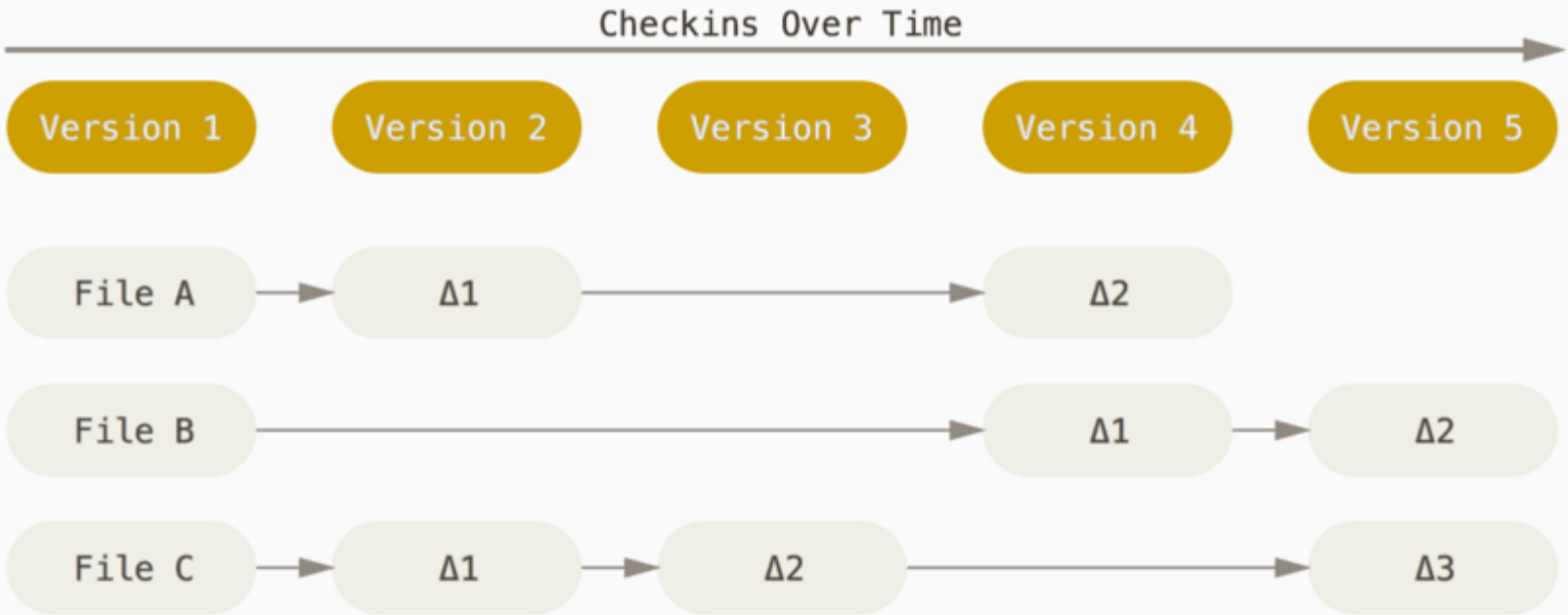
De acordo com o *git-scm*, *"Controle de Versão é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo..."*

O que é Controle de Versão?

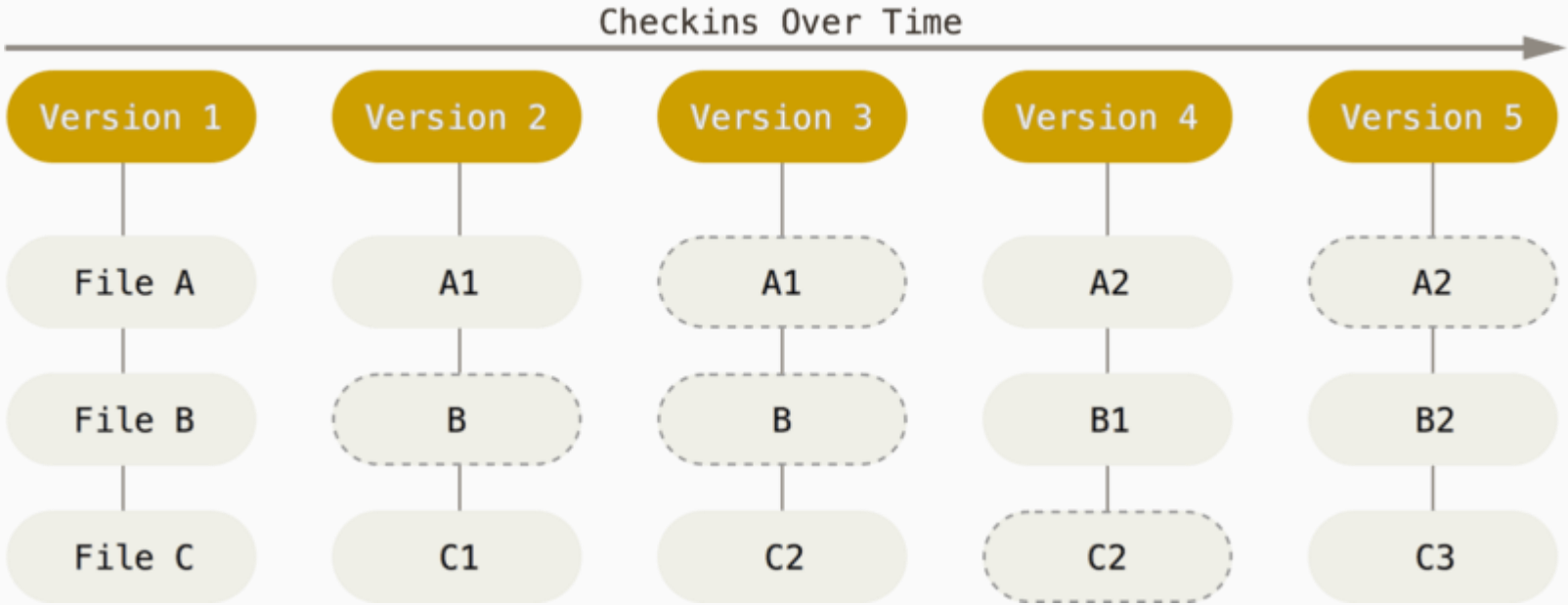
- *Acesso a toda e qualquer versão já feita do projeto*
- *Organiza e impulsiona trabalho colaborativo*
- *Estatísticas de desenvolvimento*
- *Reversão do estado de arquivos*
- *Comparar mudanças ao longo do tempo*
- *Facilidade para rastrear problemas no projeto*
- *Recuperação de arquivos*

Por que utilizá-lo?

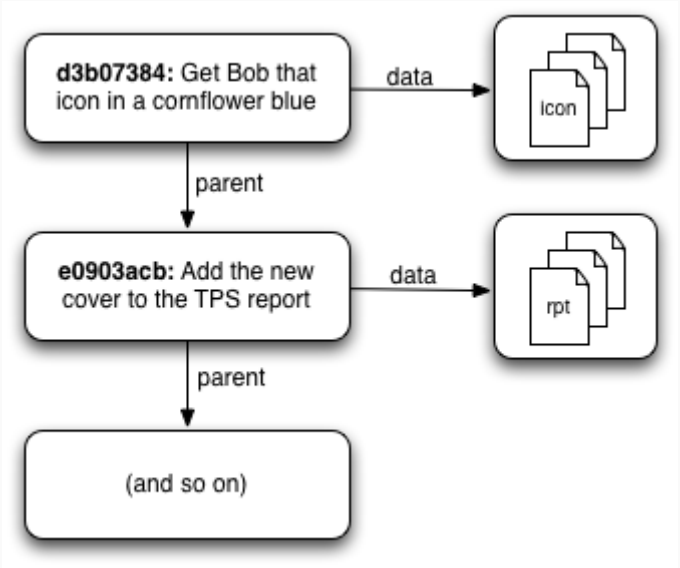
Segundo [git-scm](), *"most other systems store information as a list of file-based changes"*



Segundo *git-scm*, “Git thinks of its data more like a set of snapshots of a miniature filesystem. Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot”



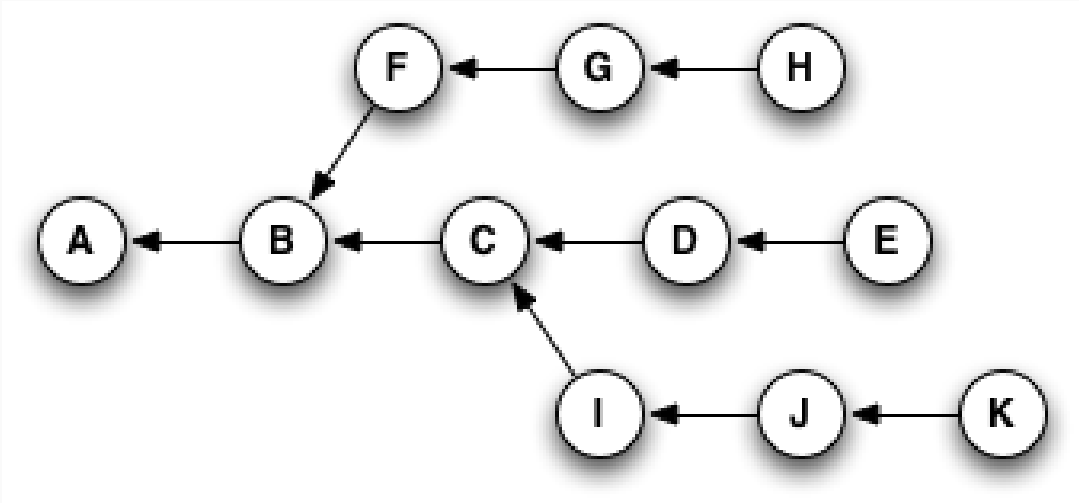
Git é um enorme Grafo!



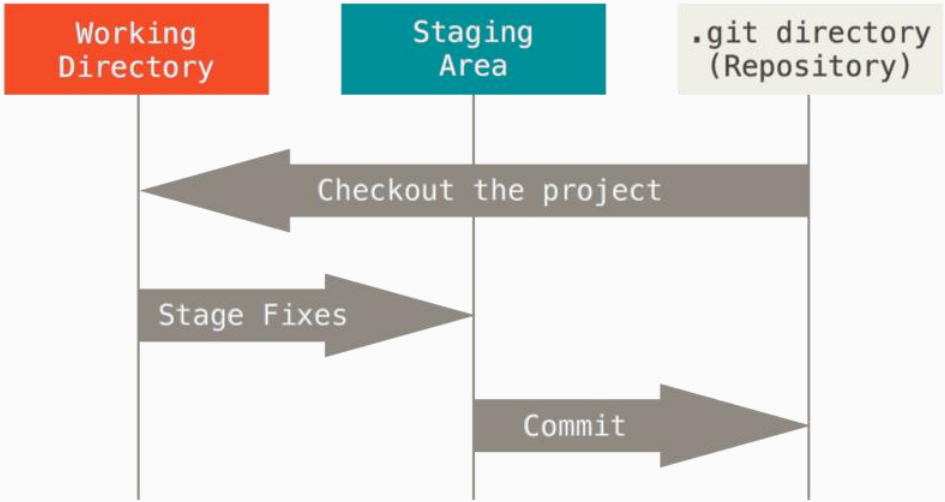
Falando num alto nível, Git é composto de duas coisas:

1. Ponteiro para o estado do código em algum determinado momento
2. Zero ou mais ponteiros para o(s) Commit(s) “Pai(s)”

Qual o segredo do Git?

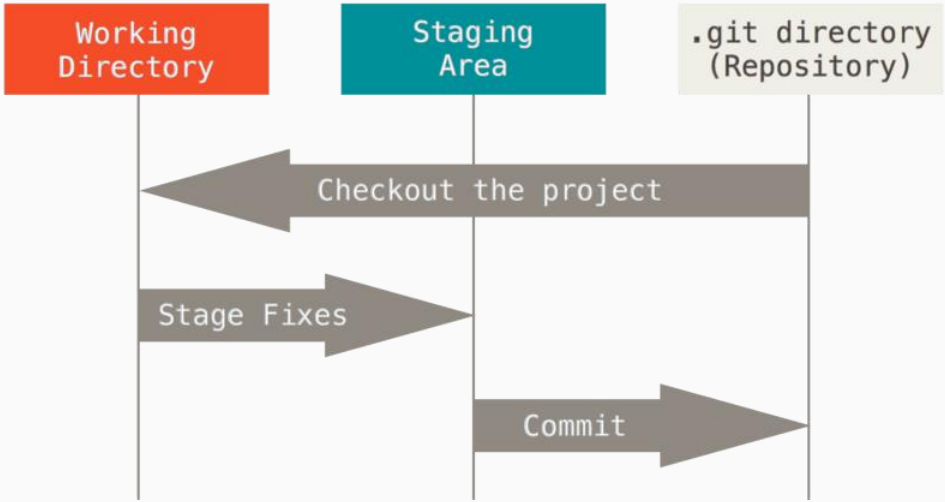


Qual o segredo do Git?



São eles:

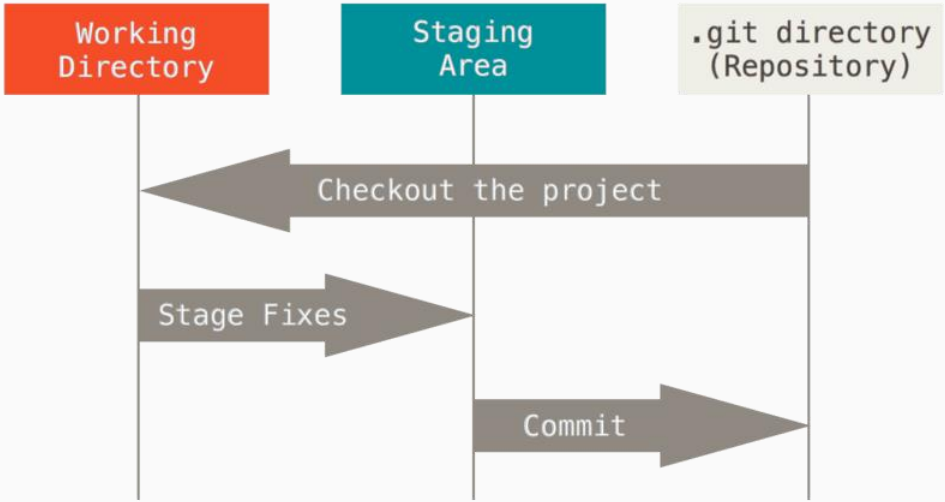
- Commited
- Modified
- Staged



São eles:

- Commited

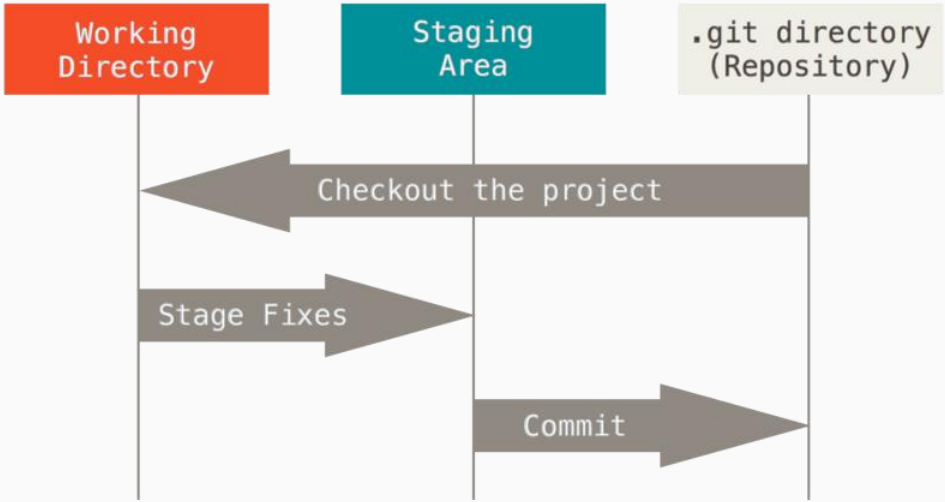
Dados estão armazenados no repositório



São eles:

- Modified

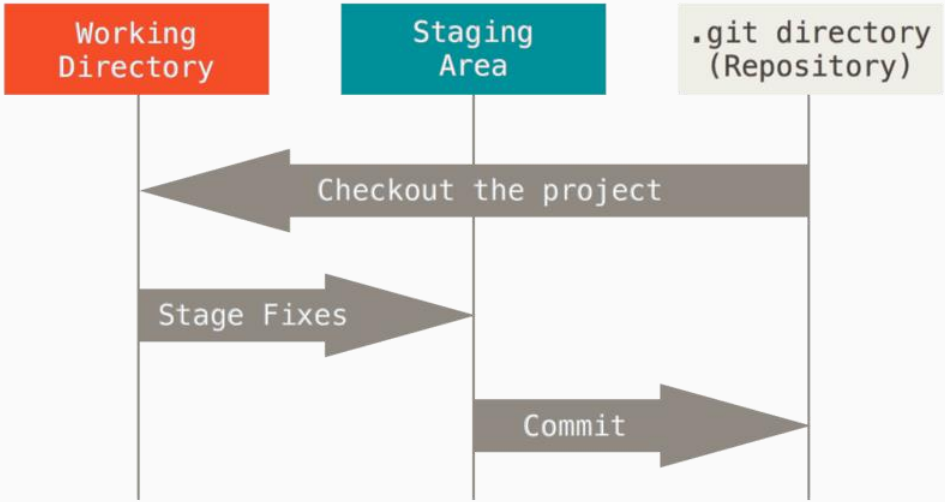
Mudanças foram feitas mas não foram comitadas



São eles:

- Staged

Atual versão dos arquivos modificados foram marcadas para serem comitadas



O fluxo de trabalho no Git, no geral, é:

- Arquivos são modificados no diretório de trabalho;
- Modificações são indexadas;
- Arquivos indexados são comitados

Antes de mais nada, vamos ignorar algumas coisas com o .gitignore

fluxo de trabalho

seus repositórios locais consistem em três "árvores" mantidas pelo git. a primeira delas é sua `Working Directory` que contém os arquivos vigentes. a segunda `Index` que funciona como uma área temporária e finalmente a `HEAD` que aponta para o último *commit* (confirmação) que você fez.

criando um novo repositório

crie uma nova pasta, abra-a e execute o comando

```
git init
```

para criar um novo repositório.

obtenha um repositório

crie uma cópia de trabalho em um repositório local executando o comando

```
git clone /caminho/para/o/repositório
```

quando usar um servidor remoto, seu comando será

```
git clone usuário@servidor:/caminho/para/o/repositório
```

- *git --help*
- *git status*
- *git remote –v*

Alguns comandos básicos bem legais

adicionar & confirmar

Você pode propor mudanças (adicioná-las ao **Index**) usando

```
git add <arquivo>
```

```
git add *
```

Este é o primeiro passo no fluxo de trabalho básico do git. Para realmente confirmar estas mudanças (isto é, fazer um *commit*), use

```
git commit -m "comentários das alterações"
```

Agora o arquivo é enviado para o **HEAD**, mas ainda não para o repositório remoto.

enviando alterações

Suas alterações agora estão no **HEAD** da sua cópia de trabalho local.

Para enviar estas alterações ao seu repositório remoto, execute

```
git push origin master
```

Altere *master* para qualquer ramo (*branch*) desejado, enviando suas alterações para ele.

Se você não clonou um repositório existente e quer conectar seu repositório a um servidor remoto, você deve adicioná-lo com

```
git remote add origin <servidor>
```

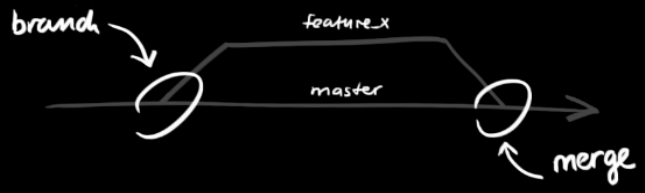
Agora você é capaz de enviar suas alterações para o servidor remoto


GIT PUSH MASTER
I TOO LIKE TO LIVE DANGEROUSLY
memegenerator.net

Git Hub

ramificando

Branches ("ramos") são utilizados para desenvolver funcionalidades isoladas umas das outras. O branch *master* é o branch "padrão" quando você cria um repositório. Use outros branches para desenvolver e mescle-os (*merge*) ao branch master após a conclusão.



crie um novo branch chamado "funcionalidade_x" e selecione-o usando

```
git checkout -b funcionalidade_x
```

retorne para o master usando

```
git checkout master
```

e remova o branch da seguinte forma

```
git branch -d funcionalidade_x
```

git push remote :branch

- git stash
- git stash list
- git stash apply [stash@{2}]
- git stash drop [stash@{2}]
- git stash pop

atualizar & mesclar

para atualizar seu repositório local com a mais nova versão, execute

```
git pull
```

na sua pasta de trabalho para *obter* e *fazer merge* (mesclar) alterações remotas.

para fazer merge de um outro branch ao seu branch ativo (ex. master), use

```
git merge <branch>
```

em ambos os casos o git tenta fazer o merge das alterações automaticamente. Infelizmente, isto nem sempre é possível e resulta em *conflitos*. Você é responsável por fazer o merge estes *conflitos* manualmente editando os arquivos exibidos pelo git. Depois de alterar, você precisa marcá-los como merged com

```
git add <arquivo>
```

antes de fazer o merge das alterações, você pode também pré-visualizá-as usando

```
git diff <branch origem> <branch destino>
```


BRACE YOURSELF
MERGE CONFLICTS ARE COMING
memegenerator.net

Git Hub

sobrescrever alterações locais

No caso de você ter feito algo errado (que seguramente nunca acontece ;)) você pode sobrescrever as alterações locais usando o commando

```
git checkout -- <arquivo>
```

isto substitui as alterações na sua árvore de trabalho com o conteúdo mais recente no HEAD. Alterações já adicionadas ao index, bem como novos arquivos serão mantidos.

Se ao invés disso você deseja remover todas as alterações e commits locais, recupere o histórico mais recente do servidor e aponte para seu branch master local desta forma

```
git fetch origin
```

```
git reset --hard origin/master
```

Let’s

Do

It