Controle de Versão de Software com Git

Sobre mim.

Paulo Silva

Contatos:
(96) 99142-4429
ptos@meta.edu.br

- Bacharel em sistemas de informação
- Especialista em Docência do Ensino Superior;
- MBA em Arquitetura de software;
- Desenvolvedor com 7 anos de experiência;
- Consultor na aŕea de desenvolvimento web e aprendizado de máquina

Git

O Git é um sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte, com ênfase em velocidade.

O Git foi inicialmente projetado e desenvolvido por **Linus Torvalds** para o desenvolvimento do **kernel Linux**, mas foi adotado por muitos outros projetos.

Vantagens do Git

1. Velocidade;
2. Design simples;
3. Suporte robusto a desenvolvimento não linear ;
4. Totalmente distribuído;
5. Capaz de lidar eficientemente com grandes projetos como o kernel do Linux (velocidade e volume de dados).

Sistema de Controle de Versão

- Controle de histórico
- Trabalho em equipe
- Marcação e resgate de versões estáveis
- Ramificação do Projeto

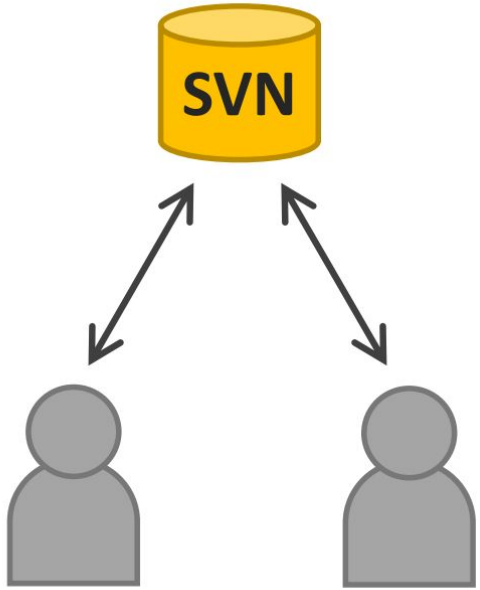
O controle de versão é um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo de forma que você possa recuperar versões específicas. (Git Book)

O por que você deve se importar com controle de versão?

1. Alguém já sobrescreveu o código de outra pessoa por acidente e acabou perdendo as alterações?
2. Tem dificuldades em saber quais as alterações efetuadas em um sistema, quando foram feitas e quem fez?
3. Tem dificuldade em recuperar o código de uma versão anterior que está em produção?
4. Tem problemas em manter variações do sistema ao mesmo tempo?

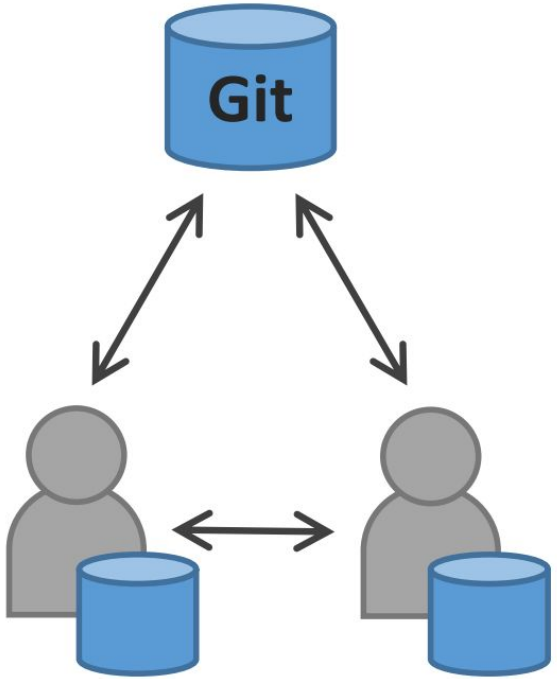
Sistema de Controle de Versão Centralizado

- Pouca autonomia
  - Ações necessitam de acesso ao servidor.
- Trabalho privado limitado
  - Versiona apenas arquivos no repositório.
- Risco de perda de dados
  - Tudo em um único repositório.



Sistema de Controle de Versão Distribuído

- Autonomia
  - Ações básicas “offline”.
- Rapidez
  - Processos são locais.
- Trabalho privado
  - Trabalho local não afeta os demais.
- Confiabilidade
  - Todo repositório é um backup, ou seja, uma cópia completa do repositório, incluindo versões anteriores e histórico.



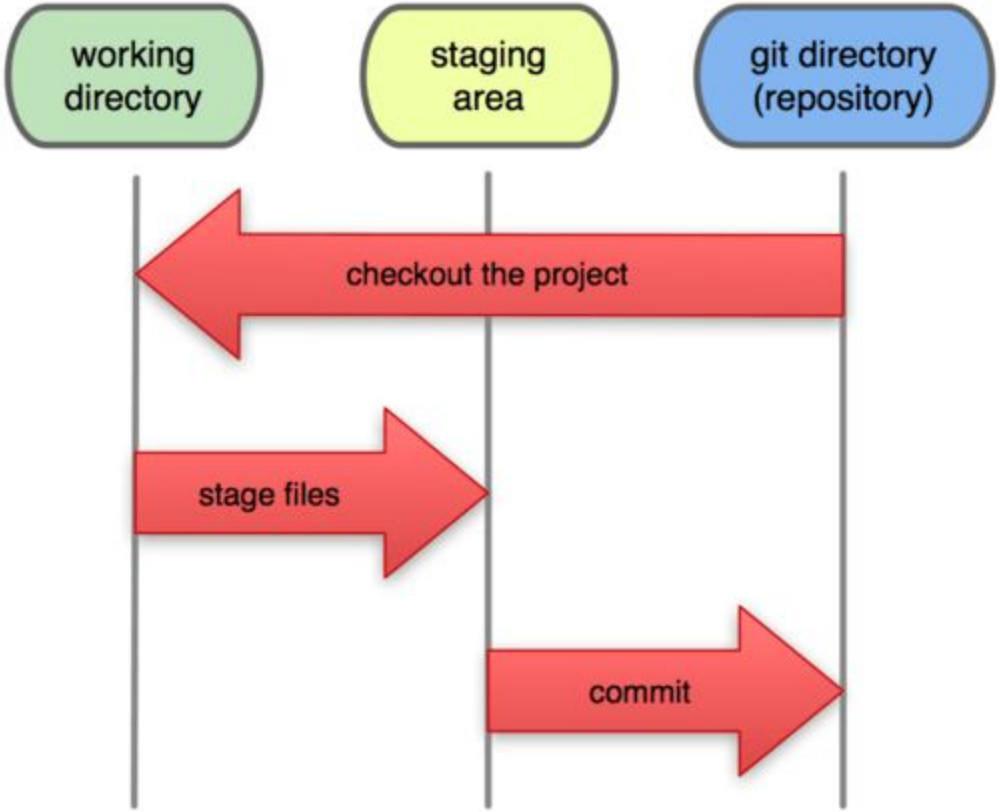
Alguns conceitos ...

- Área de trabalho
  - Local onde está o diretório git
- Repositório
  - Onde o git trabalha as modificações
- Repositório remoto
  - Repositório git armazenado em um local compartilhado

Fluxo de Trabalho

1. Você modifica arquivos no seu diretório de trabalho;
2. Você seleciona os arquivos, adicionando snapshots deles para sua área de preparação;
3. Você faz um **commit**, que leva os arquivos como eles estão na sua área de preparação e os armazena permanentemente no seu diretório **Git**.

Local Operations



Instalando o Git

- Instalando no Windows
  - Acesse a seguinte URL, faça o download e instale a última versão disponível: http://msysgit.github.io/
- Instalando no Linux
  - Para instalar o Git no Ubuntu, ou em uma outra distribuição baseada em Debian, execute em um terminal:
    - $ sudo apt-get install git
  - No Fedora, utilize:
    - $ sudo yum install git
  - Para as demais distribuições do Linux, veja o comando em: http://git-scm.com/download/linux

Configurações básicas

É importante nos identificarmos para o Git, informando nosso nome e e-mail.

Em um terminal, podemos executar os comandos a seguir:

```
$ git config --global user.name "Paulo Silva"

$ git config --global user.email 05.paulotarso@gmail.com
```

Criando um Repositório

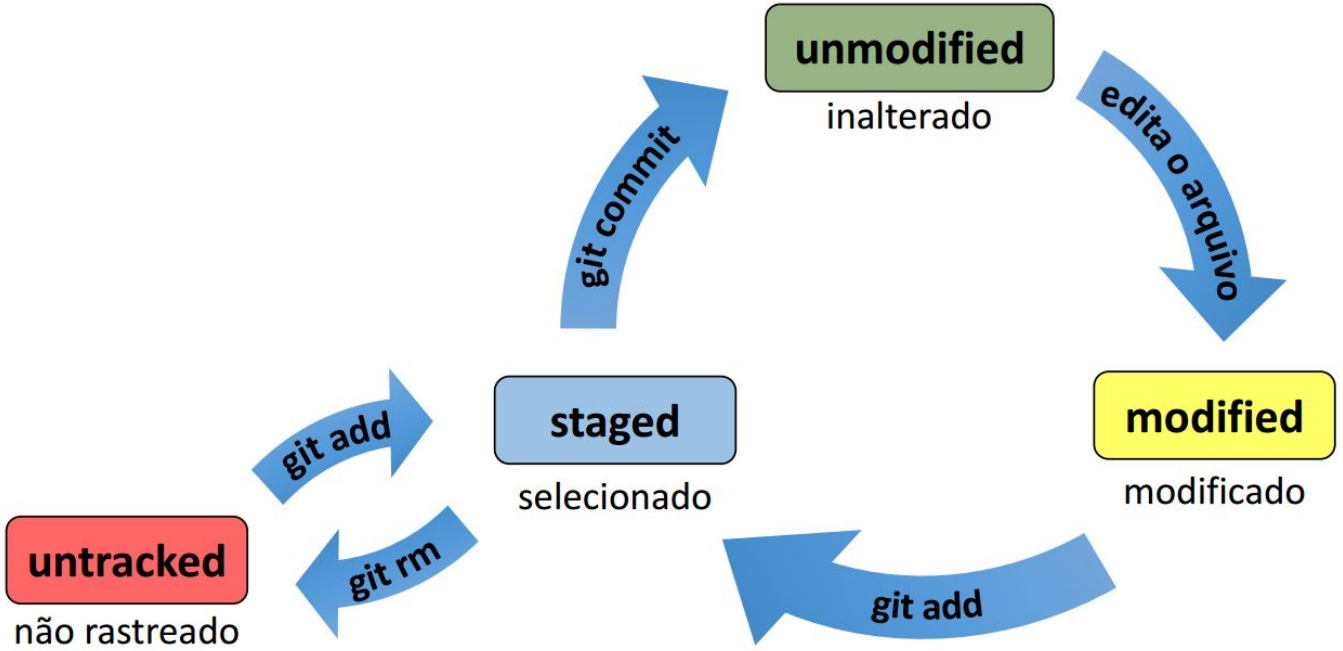
```
$ git init
```

Transforma a diretório atual em um repositório git, criando o subdiretório ".git".

```
$ git init <dir>
```

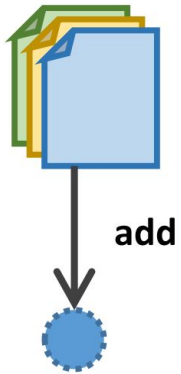
Cria o diretório <dir> e transforma em um repositório git.

Tipos de Estado de um Arquivo



Preparando Para Salvar Alterações

```
$ git add <arquivo | dir>
```



Stage Area (Index)

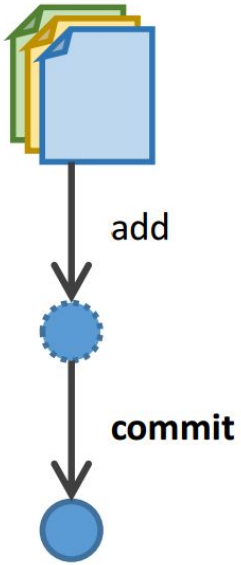
Adiciona as mudanças do arquivo ou do diretório para o próximo **commit**. O arquivo passa a ser rastreado.

Analisando os Arquivos na Área Transitória

```
$ git status
```

Lista os arquivos que estão e que não estão na área transitória, e os arquivos que não estão sendo rastreados.

Salvando Alterações




```
$ git commit
```

Realiza o commit e abre o editor para inserir add uma

```
$ git commit –m “<mensagem>”
```

Realiza o commit, com a mensagem “<mensagem>”

.gitignore

Arquivo que contém os arquivos que não serão visíveis pelo git.

.gitignore (exemplo)	
Thumbs.db	#Arquivo específico
*.html	#Arquivos que terminam com “.html”
!index.html	#Exceção, esse arquivo será visível ao git
log/	#Diretório específico
**/tmp	#Qualquer diretório nomeado de “tmp”

- Arquivos que já estavam sendo rastreados não são afetados.

Verificando mudanças nos arquivos

```
$ git diff
```

Lista as mudanças de todos arquivos rastreados.

```
$ git diff <arquivo>
```

Lista as mudanças do arquivo selecionado

```
$ git log
```

Exibe o log de commits.

DESFAZENDO AS COISAS

```
$ git  commit --amend
```

Modificando o último commit

```
$ git checkout <arquivo>
```

Desfazendo um arquivo modificado

```
$ git reset HEAD <arquivo>
```

Remove o arquivo da área de validação

Tagging

Assim como a maioria dos VCS's, Git tem a habilidade de criar tags em pontos específicos na história do código como pontos importantes.

Geralmente as pessoas usam esta funcionalidade para marcar pontos de release (v1.0, e por aí vai).

Criando uma tag

```
$ git tag –a <tag> -m “<msg>”
```

Cria a tag <tag> completa para o último commit com a mensagem <msg>.

Exemplo:

```
$ git tag -a v1.4.4 -m “versão 1.4 do aplicativo”
```

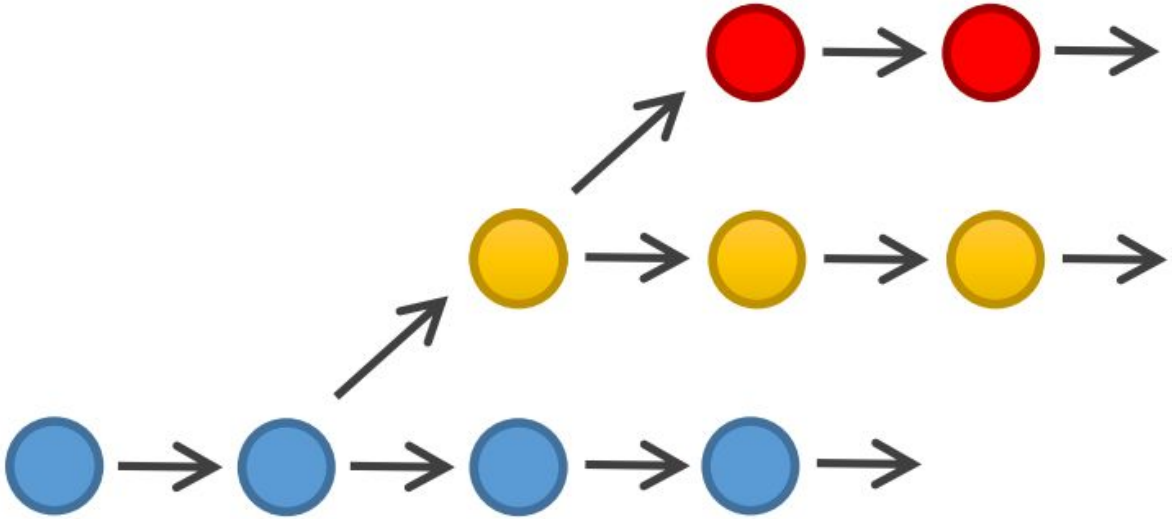
Versionamento

v[**major**].[**minor**].[**patch**]

- [**patch**]:
  - Correção de bugs.
- [**minor**]:
  - Incrementos de funcionalidades compatíveis com versões anteriores.
- [**major**]:
  - Incrementos de funcionalidades incompatíveis com versões anteriores. Versões teste: alpha (a), beta (b)

Ex: v0.1.9, v0.1.10, v0.2.0a, v0.2.0b, v0.2.0

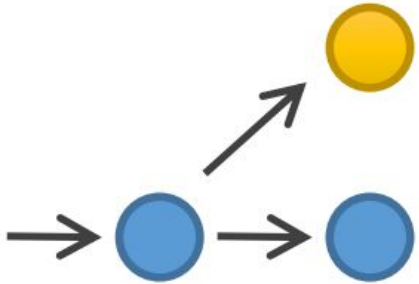
Branches



Branches

```
$ git branch <branch>
```

Cria o branch a partir do commit .



Alternando em Ramificações

```
$ git checkout <branch>
```

Altera para o branch <branch>

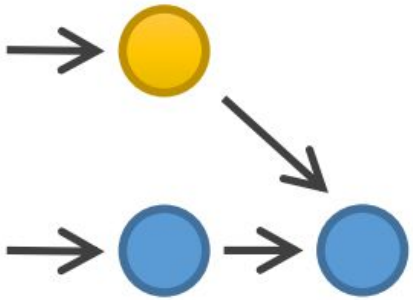
```
$ git checkout -f <branch>
```

Altera para o branch <branch> “na força”, perdendo-se as informações não “commitadas”.

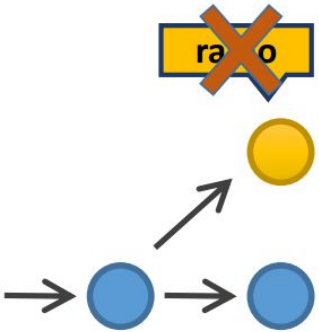
Mesclando Commits

```
$ git merge <branch>
```

Mescla os commits do branch <branch> para o branch atual.



Excluindo Ramificações



```
$ git branch -d <branch>
```

Exclui o branch <branch>. O branch já deve ter sido mesclado.

```
$ git branch -D <branch>
```

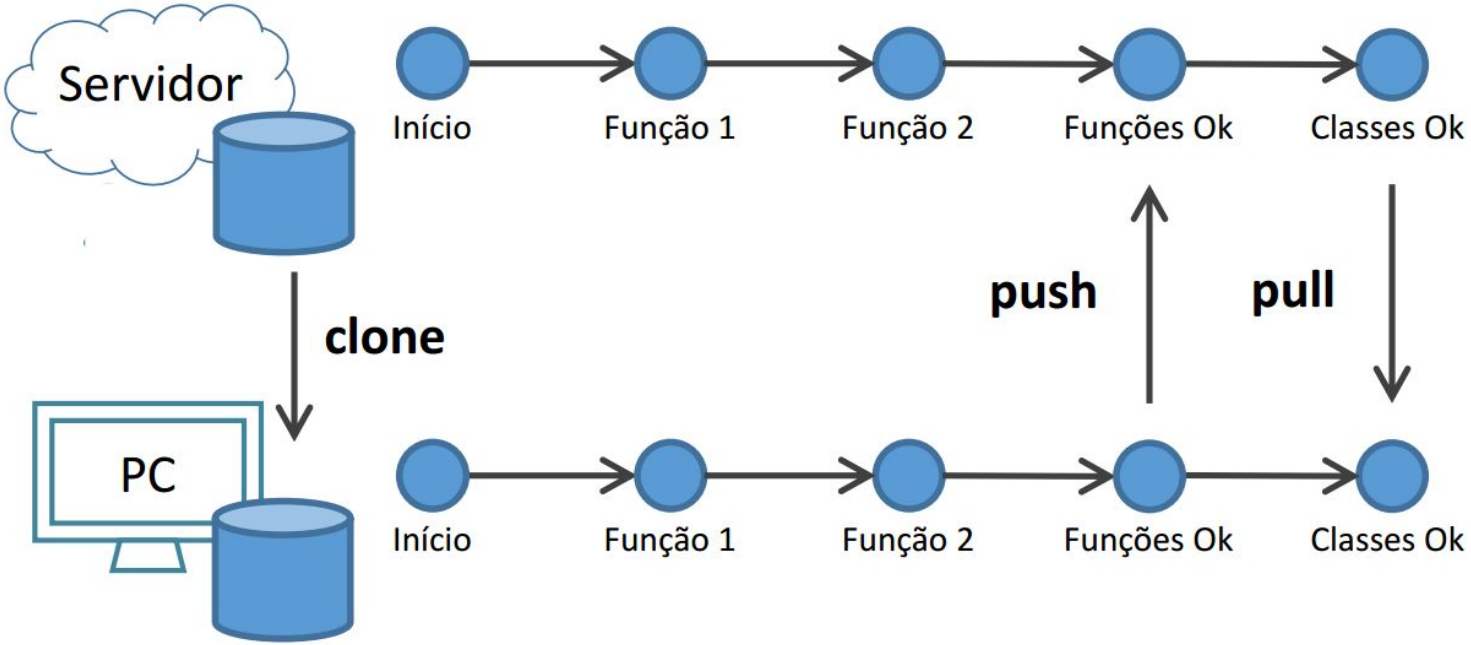
Exclui o branch <branch> mesmo não tendo sido mesclado.

Compartilhando seu código através do GitHub

Para que o mundo possa descobrir nosso incrível projeto, temos que compartilhá-lo na internet. Para isso, podemos utilizar uma aplicação web chamada GitHub.

O primeiro passo é criar uma conta no GitHub. Para projetos de código aberto, não há custo nenhum! Com um navegador, acesse: https://github.com/ Preencha seu nome, e-mail e escolha uma senha.

Entendendo a base do funcionamento com repositório remoto



Servidores Para Hospedagem

Obrigado!

Fontes e referências

- http://gitimmersion.com/index.html
- https://git-scm.com/book/pt-br/v1/Primeiros-passos