Minicurso Git

Pedro Henrique Penna

pedro.penna@sga.pucminas.br

https://github.com/ppenna

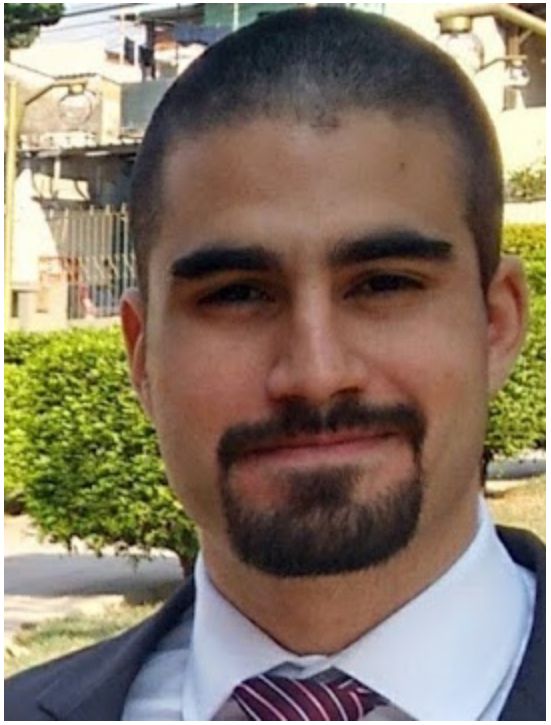
PUC Minas

XI Escola de Férias da Computação

Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

Sobre Mim – Formação

- **BSc em Ciência da Computação**
  - PUC Minas, 2015
  - Sistemas Operacionais



- **MSc em Ciência da Computação**
  - UFSC, 2017
  - Computação de Alto Desempenho
- **PhD *Candidate* em Ciência da Computação**
  - UGA (França) + PUC Minas
  - Sistemas Operacionais Distribuídos

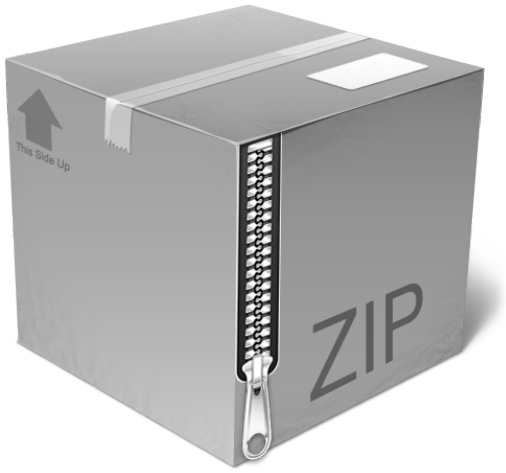
Sobre Mim – Experiência

- **CAP Benchmarks (2013 – 2017)**
  - https://github.com/cart-pucminas/CAPBenchmarks
  - Suíte de *benchmarks* para processadores manycore
  - Desenvolvido em C + OpenMP (12k LoC)
  - **O quê fiz:** projetei e desenvolvi 3 de 7 benchmarks
- **Nanvix (2011 – Hoje)**
  - https://github.com/nanvix
  - SO distribuído para processadores manycore
  - Desenvolvido em Assembly + C (150k LoC)
  - **O quê faço**: criador e principal engenheiro

Sumário

Sistemas de Controle de Versão #1

- **Como gerenciar o código fonte de um projeto?**
  - Fazer backup em arquivos compactados
  - Versionamento: usar um arquivo para cada versão
- **Como fazer o backup?**
  - Backup total: guardar todo o projeto
  - Backup incremental: guardar diferenças entre versões
- **Quais os problemas?**
  - Gerência de versões
  - Trabalho em equipe



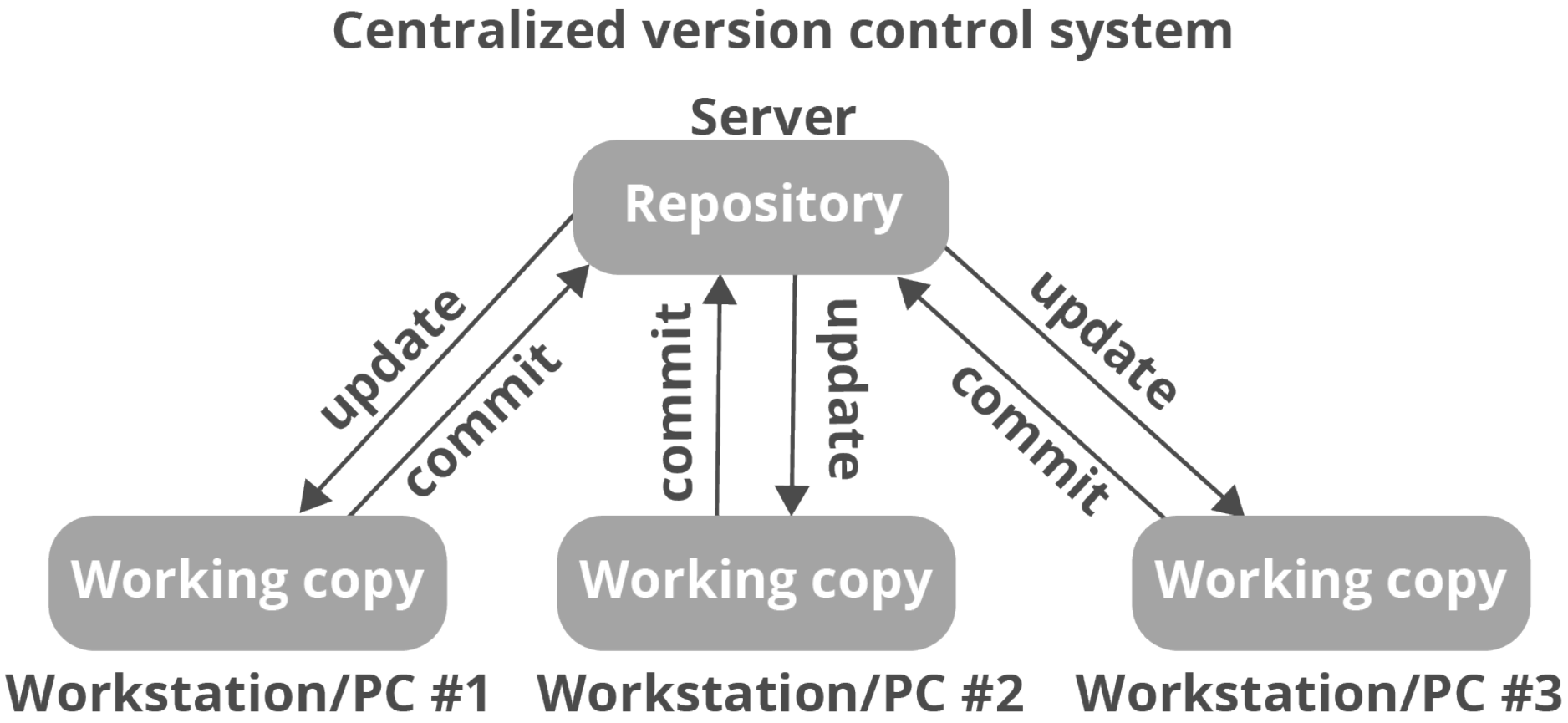
Sistemas de Controle de Versão #2

- **Sistemas de Controle de Versão**
  - Ex: CVS, Mercurial, SVN, Git...
  - Gerenciam o histórico do projeto
  - Permitem a navegabilidade entre versões
  - Facilitam o rastreio de bugs
  - Possibilitam o trabalho em equipe

- **Como esses sistemas funcionam?**
  - Gerenciam de forma transparente os backups (repositório)
  - Associam identificadores únicos a alterações no projeto
  - Expõem operações no repositório para vários clientes

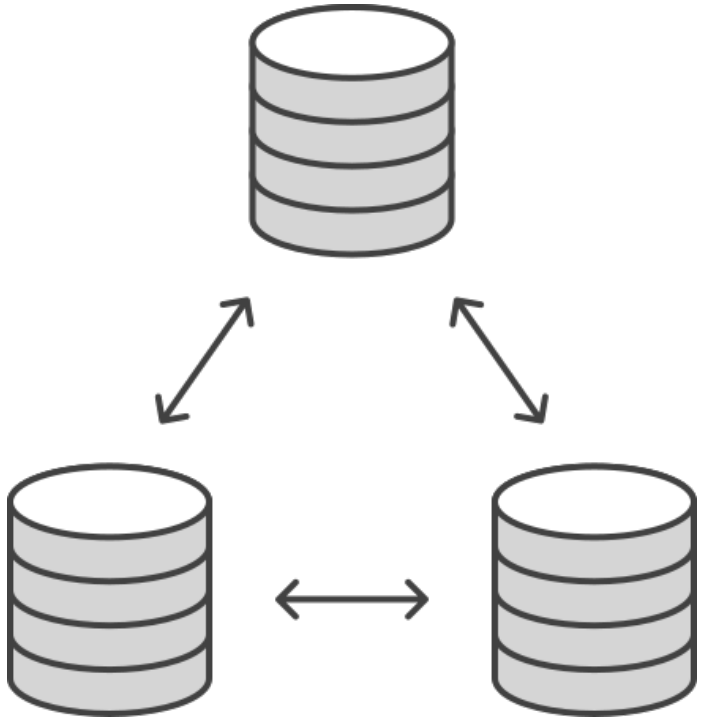
Sistemas de Controle de Versão #3

- **Commit:** envia alterações para o repositório
- **Update:** carrega alterações do repositório



Git: Controle de Versão Distribuído

- **Controle de Versão distribuído**
  - Uma cópia do repositório em cada cliente
  - Diminui erros de projeto e aumenta tolerância a falhas
- **Git**
  - Desenvolvido por Linus Torvalds (em 4 dias)
  - Criado para atender demandas técnicas do projeto Linux
  - Escalável e com alto desempenho



Trabalhando com Repositórios Locais #1

- **Inicializar Repositório Local**

```
git init
```

- **Clonar Repositório Remoto**

```
git clone <url>
```

- **Configurar Credenciais**

```
git config --local user.email <email>
```

  - `git config --local user.name <nome>`

Trabalhando com Repositórios Locais #2

- **Adicionar Arquivos a Área de *Staging***

  ```
  git add [--patch] <arquivo|diretório>
  ```

- **Criar um *Commit***

  ```
  git commit [-m <mensagem>]
  ```

- **Verificar Estado do Repositório**

  `git status` OU `git log`

- **Remover Arquivos do Repositório**

  ```
  git rm [-r] [-f] <arquivo|diretório>
  ```

Trabalhando com Repositórios Locais #3

- **Verificar Alterações**

  ```
  git diff
  ```

- **Salvar Temporariamente Alterações**

  ```
  git stash push [-m <mensagem>]
  ```

- **Restaurar Alterações Temporárias**

  ```
  git stash list OU git stash show [-v]
  ```

- **Restaurar Alterações Temporárias**

  ```
  git stash pop [stash@{id}]
  ```

Trabalhando com Repositórios Locais #4

- **Descartar Alterações**

```
git checkout [--patch] <arquivo|diretório>
```

- **Remover Arquivos Não Listados**

```
git clean [-f] [-d] <arquivo|diretório>
```

- **Remover Arquivos da Área de *Staging***

```
git reset [--patch] <arquivo|diretório>
```

Trabalhando com Repositórios Locais #5

- **Reverter *Commits***

```
git revert <commit>
```

- **Criar *Patches***

```
git format-patch <commit>
```

- **Aplicar *Patches***

```
git am [-3] <patch>
```

- **Remover Histórico de *Commits***

```
git reset [--hard] <commit>
```

Manipulando Branches #1

- **Criar uma *Branch***

```
git branch [commit] <branch>
```

- **Listar de *Branches***

```
git branch
```

- **Mudar de *Branch***

```
git checkout <branch>
```

- **Excluir uma *Branch***

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

Manipulando Branches #2

- **Mesclar *Branches***

```
git merge [--no-ff] <branch alvo>
```

- **Reposicionar *Branch***

```
git rebase [-i] <branch base>
```

Trabalhando em Equipe

- **Configurar Repositório Remoto**

```
git remote add origin <url>
```

- **Fazer Download de Alterações Remotas**

```
git pull
```

- **Fazer Upload de Alterações Locais**

```
git push [-u origin <branch>]
```

- **Excluir *Branches* Remotas**

```
git push --delete origin <branch>
```

Git Workflow

- **Nome de *Branches***
  - `feature-<nome>`
  - `enhancement-<nome>`
  - `bugfix-<nome>`
  - `stable, unstable, dev`

- **Mensagens de *Commits***
  - `<Módulo> Feature: <Mensagem>`
  - `<Módulo> Enhancement: <Mensagem>`
  - `<Módulo> Bug Fix: <Mensagem>`