

Programação Orientada a Objetos

Sistema de Controle de Versão

Prof. Dra. Paloma Oliveira

Email: paloma.oliveira@ifmg.edu.br

INSTITUTO FEDERAL
Minas Gerais
Campus Formiga

Sistema de Controle de Versões

- O que é?
- Você já utilizou?

O que é?

- O Sistema de Controle de Versões (SCV) consiste, basicamente, em um **local para armazenamento de artefatos** gerados durante o desenvolvimento de sistemas de software (MASON, 2006).

Sistema de Controle de Versão

- É um software que irá cuidar da gestão das várias versões do seu programa, ou melhor, das várias versões dos arquivos que compõem o seu programa.
- Permite
  - reverter arquivos para um estado anterior;
  - reverter um projeto inteiro para um estado anterior;
  - comparar mudanças feitas ao decorrer do tempo;
  - ver quem foi o último a modificar algo que pode estar causando problemas;
  - quem introduziu um bug e quando, e muito mais.

Sistemas de Controle de Versão

Centralizado		Distribuído	
Livre	Comercial	Livre	Comercial
SCCS(1972)	CCC/Harvest(1977)	GNU arch(2001)	TeamWare(199?)
RCS(1982)	ClearCase(1992)	Darcs(2002)	Code co-op(1997)
CVS(1990)	Sourcesafe(1994)	DCVS(2002)	BitKeeper(1998)
CVSNT(1998)	Perforce(1995)	SVK(2003)	Plastic SCM(2006)
Subversion(2000)	TFS(2005)	Monotone(2003)	
		Codeville(2005)	
		Git(2005)	
		Mercurial(2005)	
		Bazaar(2005)	
		Fossil(2007)	

Conceitos

- Termos que são comuns a SCV:

- **repositório**: que é o local de armazenamento de todas as versões dos arquivos;
- **versão**: que representa o estado de um item de configuração que está sendo modificado. Toda versão deve possuir um identificador único, ou VID (Version Identifier);

Conceitos

- **espaço de trabalho:** espaço temporário para manter uma cópia local da versão a ser modificada. Ele isola as alterações feitas por um desenvolvedor de outras alterações paralelas, tornando essa versão privada;
- **check out (clone)**: que é o ato de criar uma cópia de trabalho local do repositório;
- **update:** o ato de enviar as modificações contidas no repositório para a área de trabalho;

Conceitos

- **commit:** ato de criar o artefato no repositório pela primeira vez ou criar uma nova versão do artefato quando este passar por uma modificação;

- **merge:** que é a mesclagem entre versões diferentes, objetivando gerar uma única versão que agregue todas as alterações realizadas.

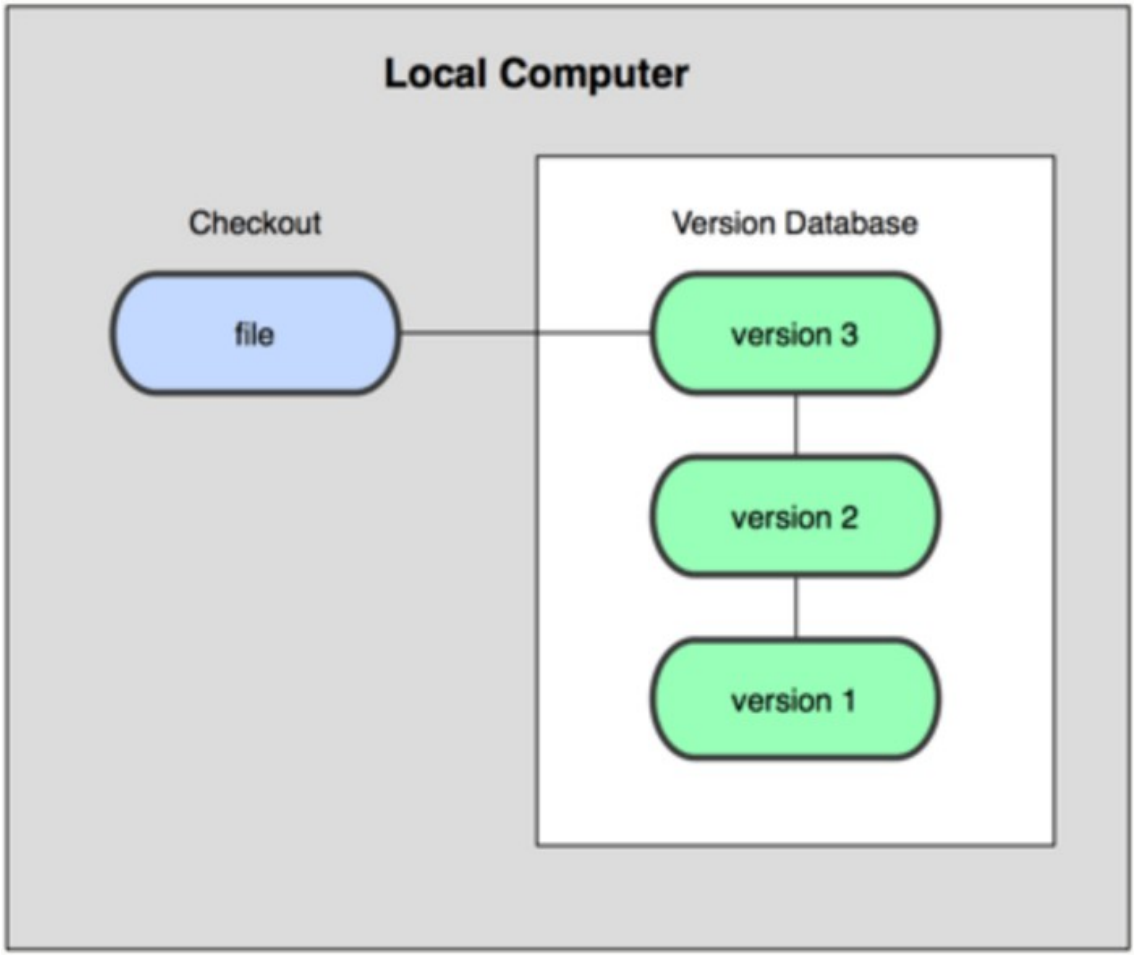
Tipos SCV

- Local: RCS
- Centralizado: CVS e Subversion
- Distribuído: Git, Mercurial, Bitkeeper

SCV Local

- O método preferido de controle de versão por muitas pessoas é copiar arquivos em outro diretório (talvez um diretório com data e hora, se forem espertos).

SCV Local


Local Computer
Checkout
file
Version Database
version 3
version 2
version 1

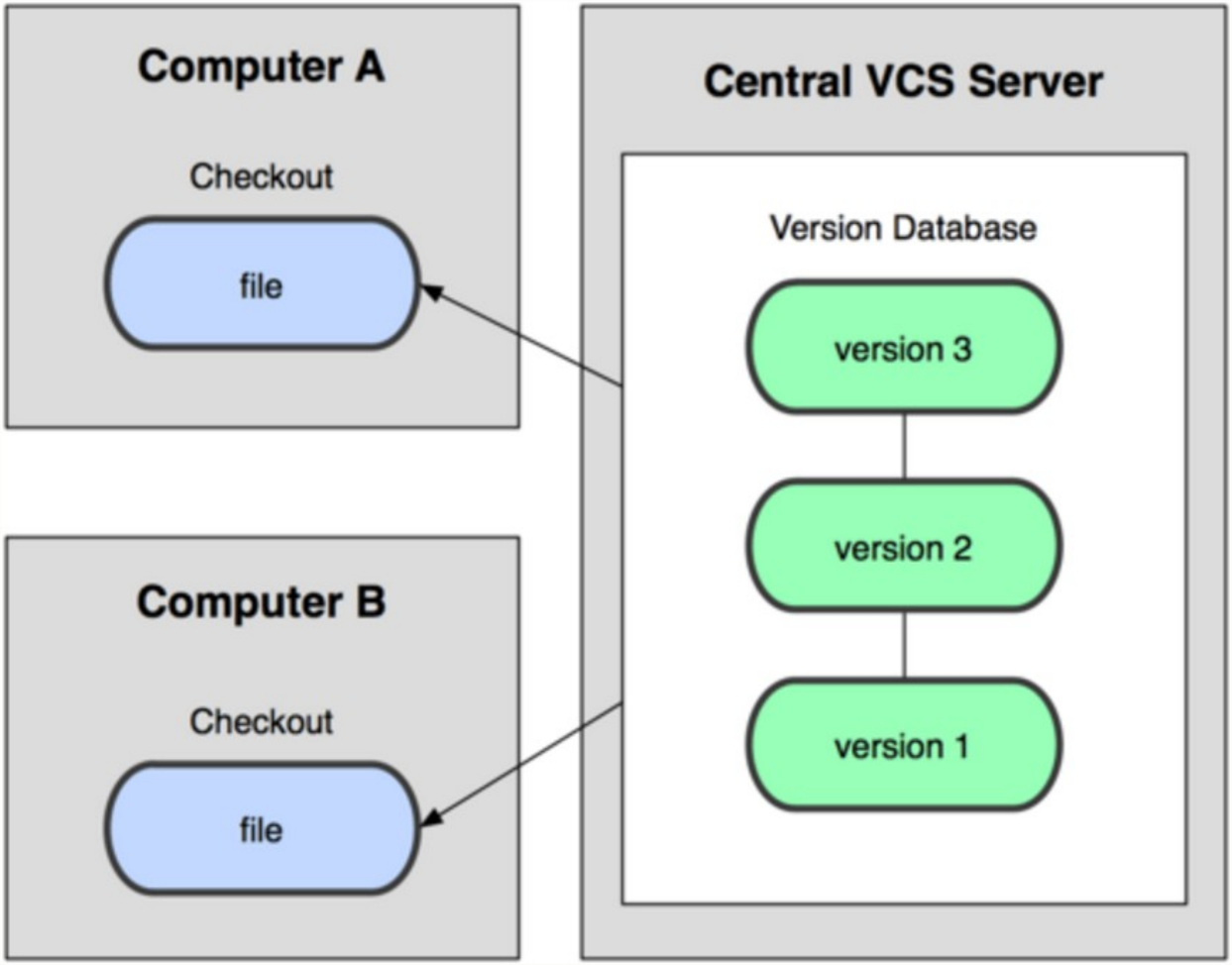
SCV Local - características

- **Suscetível a erros**: É fácil esquecer em qual diretório você está e gravar acidentalmente no arquivo errado ou sobrescrever arquivos sem querer;
- Outro grande problema: trabalhar em conjunto;
- Uma das ferramentas de SCV mais populares foi o chamado rcs, que ainda é distribuído em muitos computadores até hoje.
- **rcs:** Basicamente, mantém conjuntos de patches (ou seja, as diferenças entre os arquivos) entre cada mudança em um formato especial; a partir daí qualquer arquivo em qualquer ponto na linha do tempo pode ser recriado ao juntar-se todos os patches.

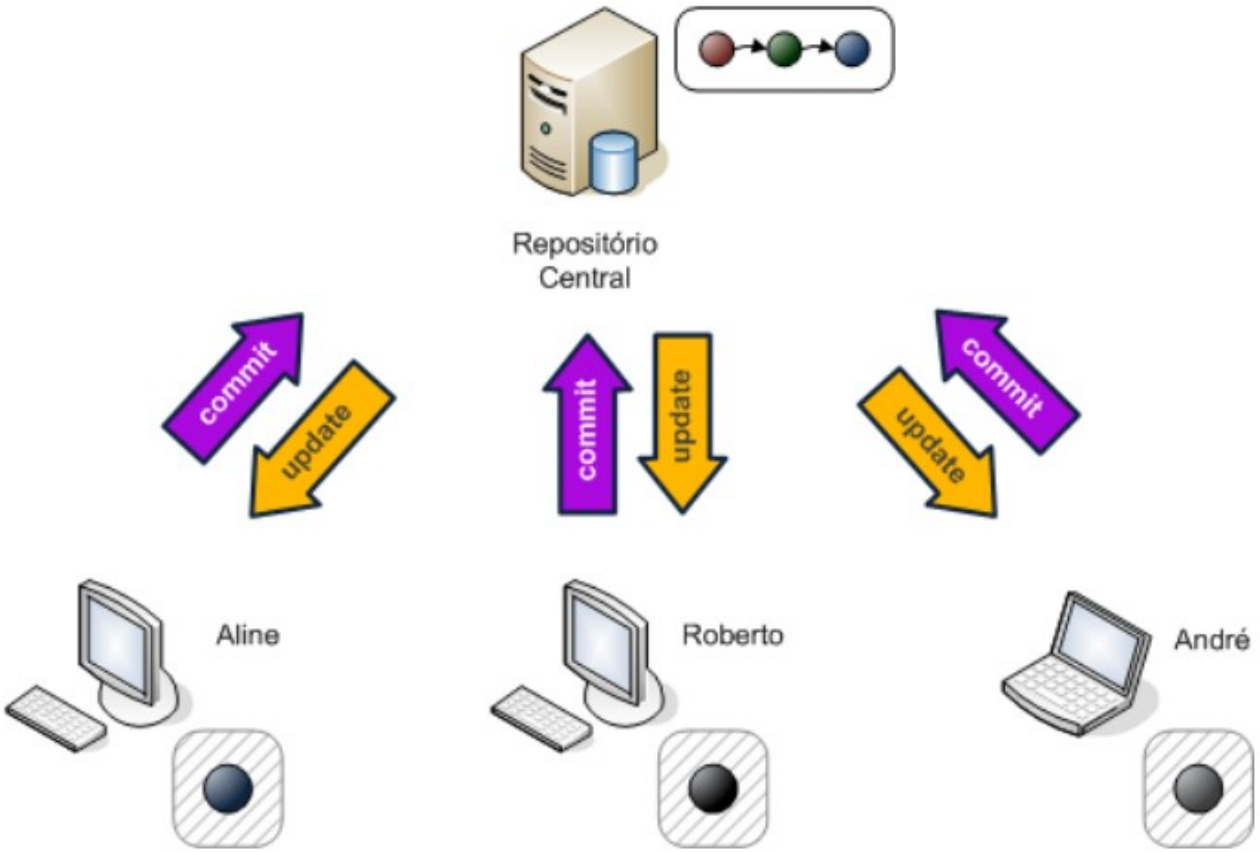
SCV Centralizado

- Controle de Versão Centralizados SCVC (Centralized Version Control System ou CVCS);
- Esses sistemas, como por exemplo o CVS, Subversion e Perforce, possuem um único servidor central que contém todos os arquivos versionados e vários clientes que podem resgatar (check out) os arquivos do servidor.
- Por muitos anos, esse foi o modelo padrão para controle de versão.

SCV Centralizado



SCV Centralizado



SCV Centralizado

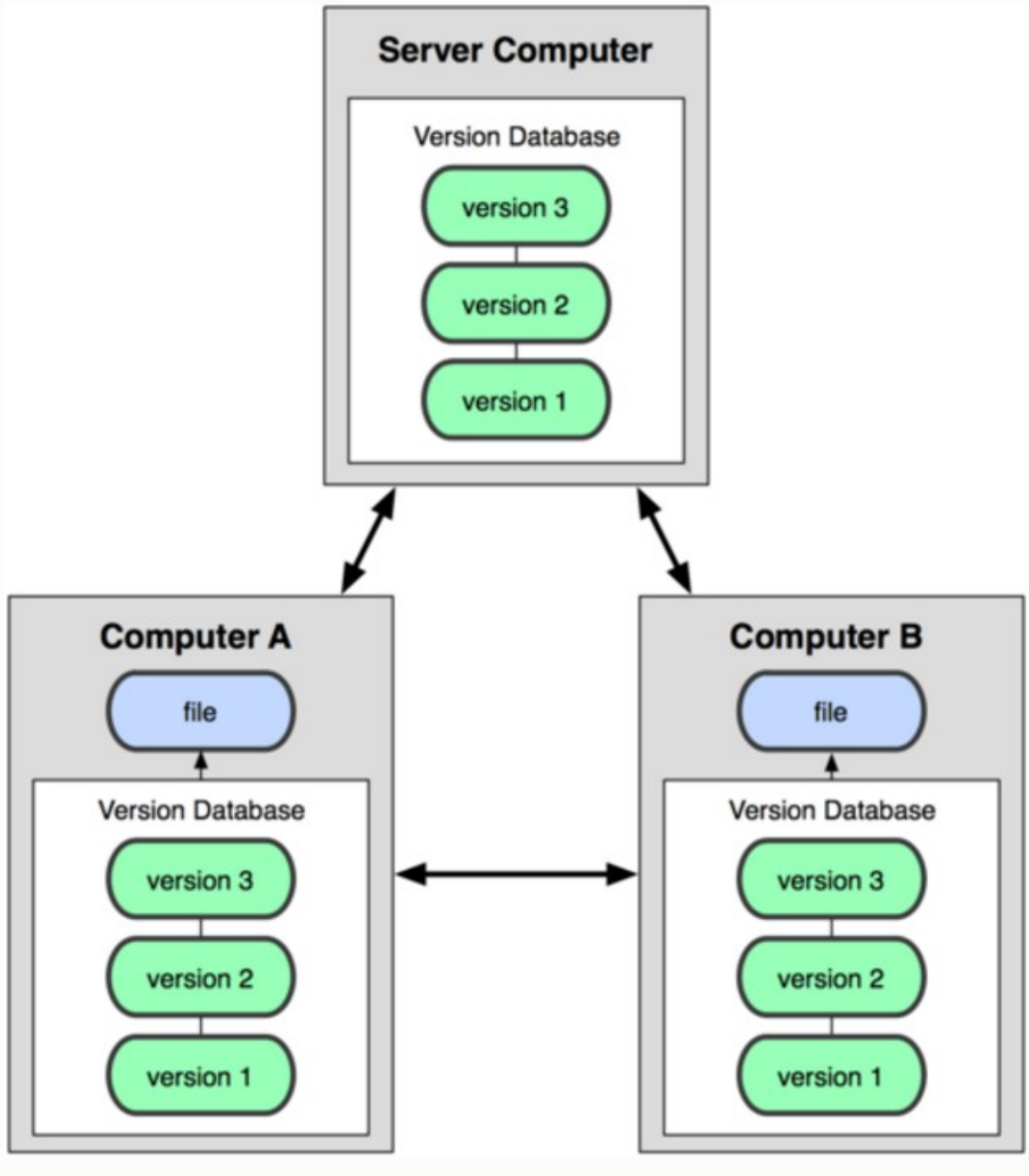
- Tal arranjo oferece muitas **vantagens**:
  - todo mundo pode ter conhecimento razoável sobre o que os outros desenvolvedores estão fazendo no projeto.
  - Gerente de projetos têm controle específico sobre quem faz o quê;
  - é bem mais fácil administrar um CVCS do que lidar com BD locais em cada cliente.

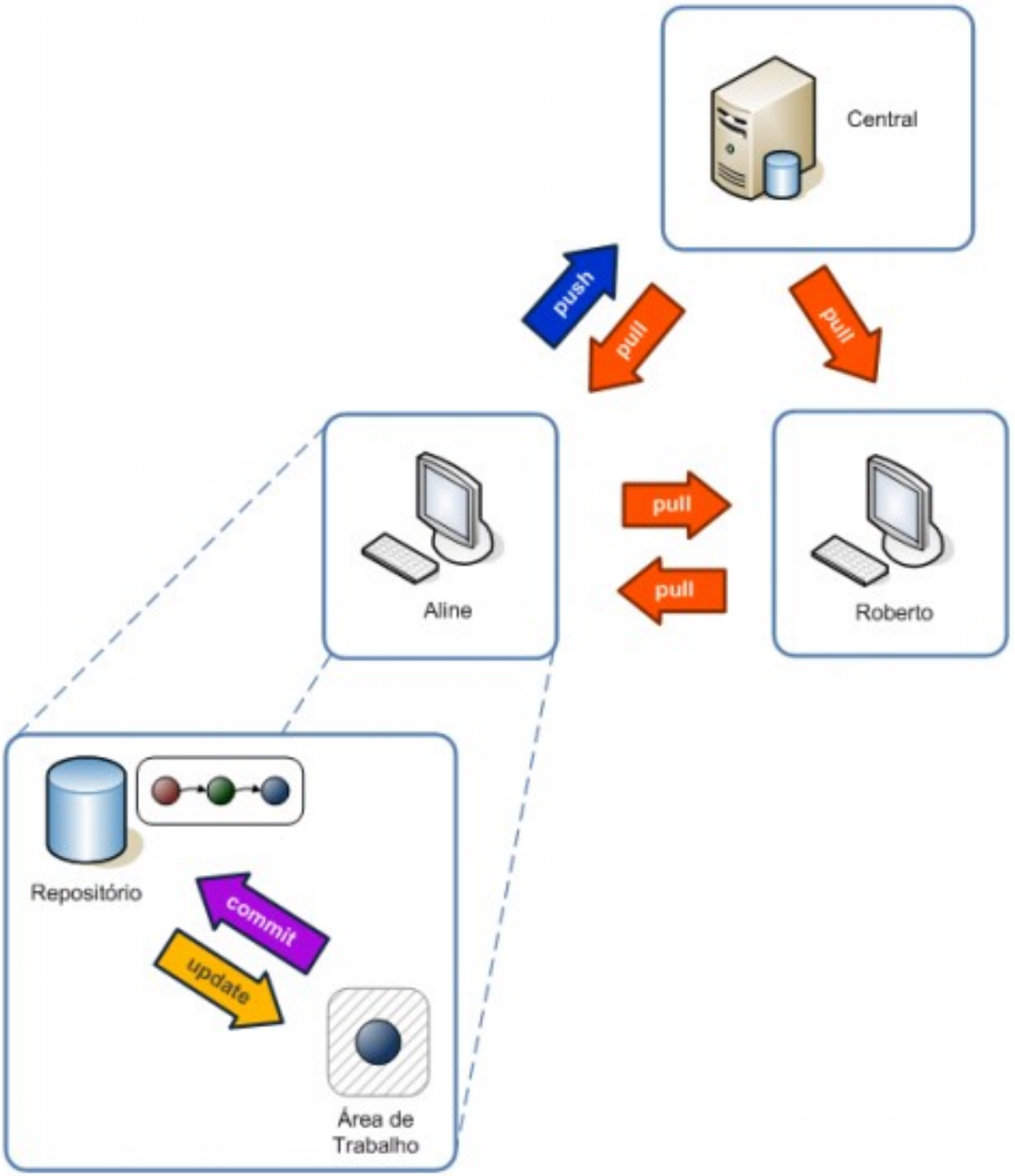
SCV Centralizado

- Desvantagens:
  - o servidor central é um ponto único de falha.
    - Se o servidor ficar fora do ar por uma hora, ninguém pode trabalhar em conjunto ou salvar novas versões dos arquivos durante esse período.
    - Se o disco do servidor do banco de dados for corrompido e não existir um backup adequado, perde-se tudo — todo o histórico de mudanças no projeto, exceto pelas únicas cópias que os desenvolvedores possuem em suas máquinas locais.

SCV Distribuído

- Sistemas de Controle de Versão Distribuídos SCVD (*Distributed Version Control System* ou DVCS).
- Em um SCVD (tais como Git, Mercurial, Bazaar ou Darcs), os clientes não apenas fazem cópias das últimas versões dos arquivos: eles **são cópias completas** do repositório.
- Assim, se um servidor falha, qualquer um dos repositórios dos clientes pode ser copiado de volta para o servidor para restaurá-lo.
- Cada checkout (clone) é na prática um backup completo de todos os dados


Server Computer
Version Database
version 3
version 2
version 1
Computer A
file
Version Database
version 3
version 2
version 1
Computer B
file
Version Database
version 3
version 2
version 1

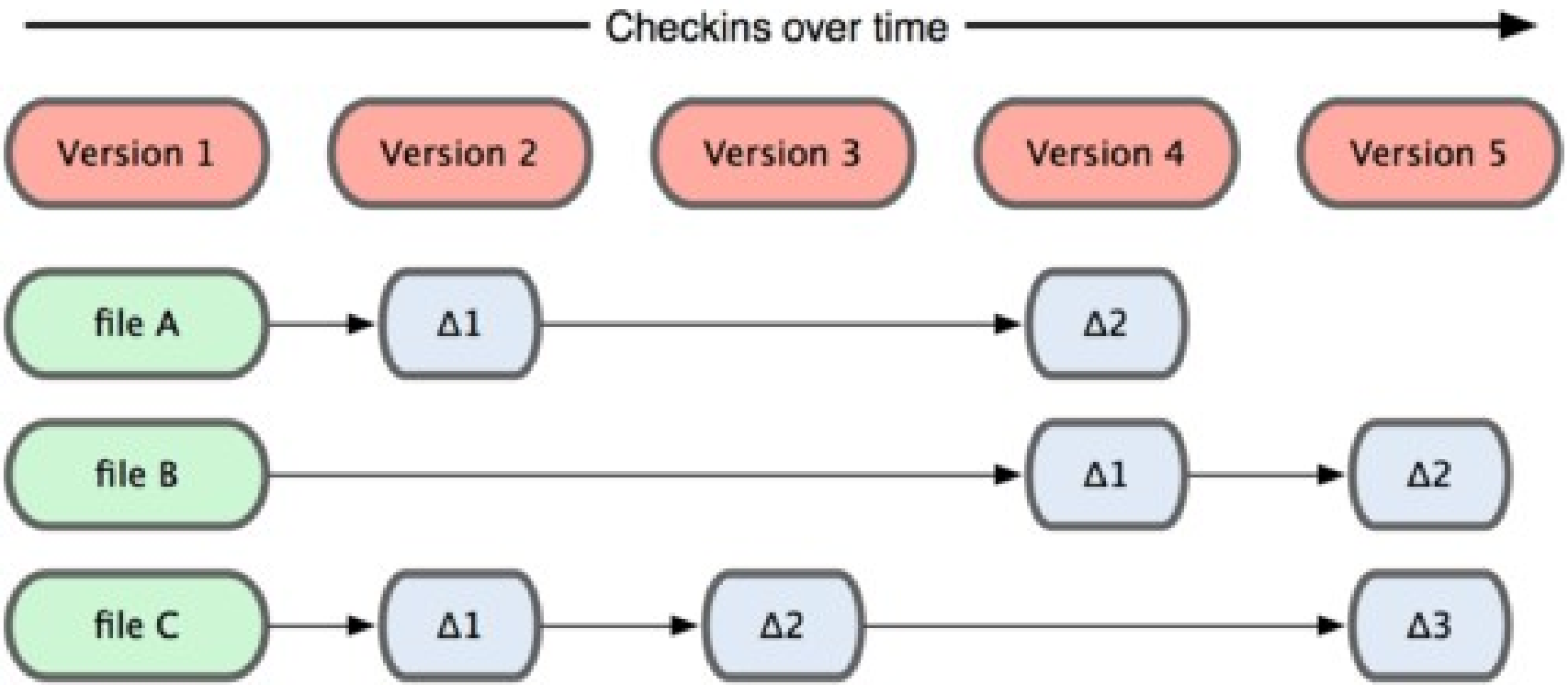

Central
push
pull
pull
Aline
pull
pull
Roberto
Repositório
commit
update
Área de
Trabalho

- Tornou um dos sistemas de controle de versão preferido dos desenvolvedores devido às suas características:
  - não depender de um servidor central;
  - potencializar o trabalho paralelo;
  - Praticamente todas as ferramentas de desenvolvimento dão suporte ao Git: Android Studio, Eclipse, NetBeans, Visual Studio, etc.

- A maior diferença entre Git e qualquer outro SCV (Subversion e similares inclusos) está na forma que o Git trata os dados;
- Outros sistemas armazena informação como uma lista de mudanças por arquivo;
- Esses sistemas tratam a informação que mantém como um conjunto de arquivos e as mudanças feitas a cada arquivo ao longo do tempo;

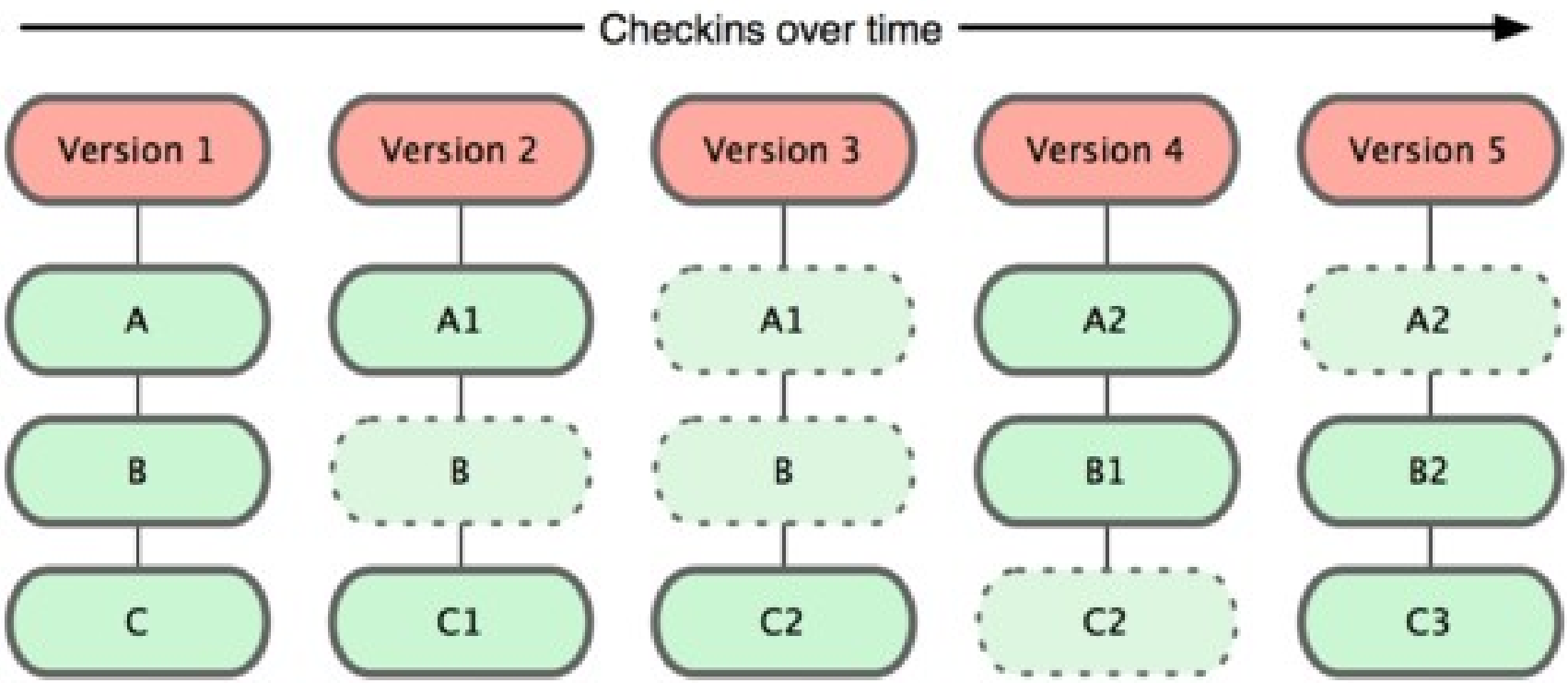
Sistema de Arquivos

- CVS, Subversion, Perforce, Bazaar, etc.



- o Git considera que os dados são como um conjunto de snapshots de um mini-sistema de arquivos;
- A cada commit é como se ele tirasse uma foto de todos os seus arquivos naquele momento e armazenasse uma referência para essa captura.
- Para ser eficiente, se nenhum arquivo foi alterado, a informação não é armazenada novamente - apenas um link para o arquivo idêntico anterior que já foi armazenado.

- Git armazena dados como snapshots do projeto ao longo do tempo

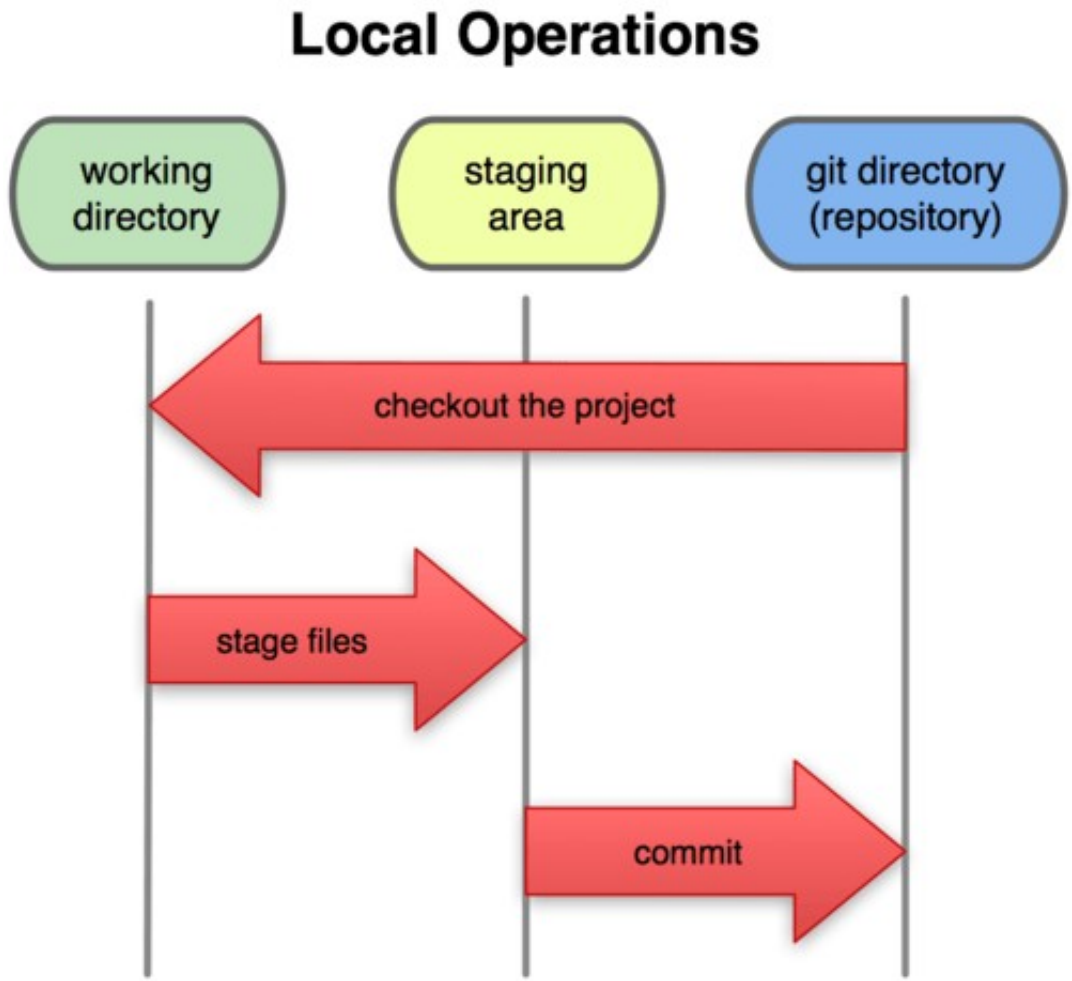


- Características:
  - Quase todas operações são locais: histórico, commits, etc.;
  - Git tem integridade (checksum em tudo);
  - Git geralmente só adiciona dados (não existe risco de perder informação);

Os Três Estados

- Git faz com que seus arquivos sempre estejam em um dos **três estados** fundamentais:
- **Consolidado (committed)**: Dados são ditos consolidados quando estão seguramente armazenados em sua base de dados local
- **Modificado (modified)**: Modificado trata de um arquivo que sofreu mudanças mas que ainda não foi consolidado na base de dados.
- **Preparado (staged)**: Um arquivo é tido como preparado quando você marca um arquivo modificado em sua versão corrente para que ele faça parte do snapshot do próximo commit (consolidação).

Workflow básico



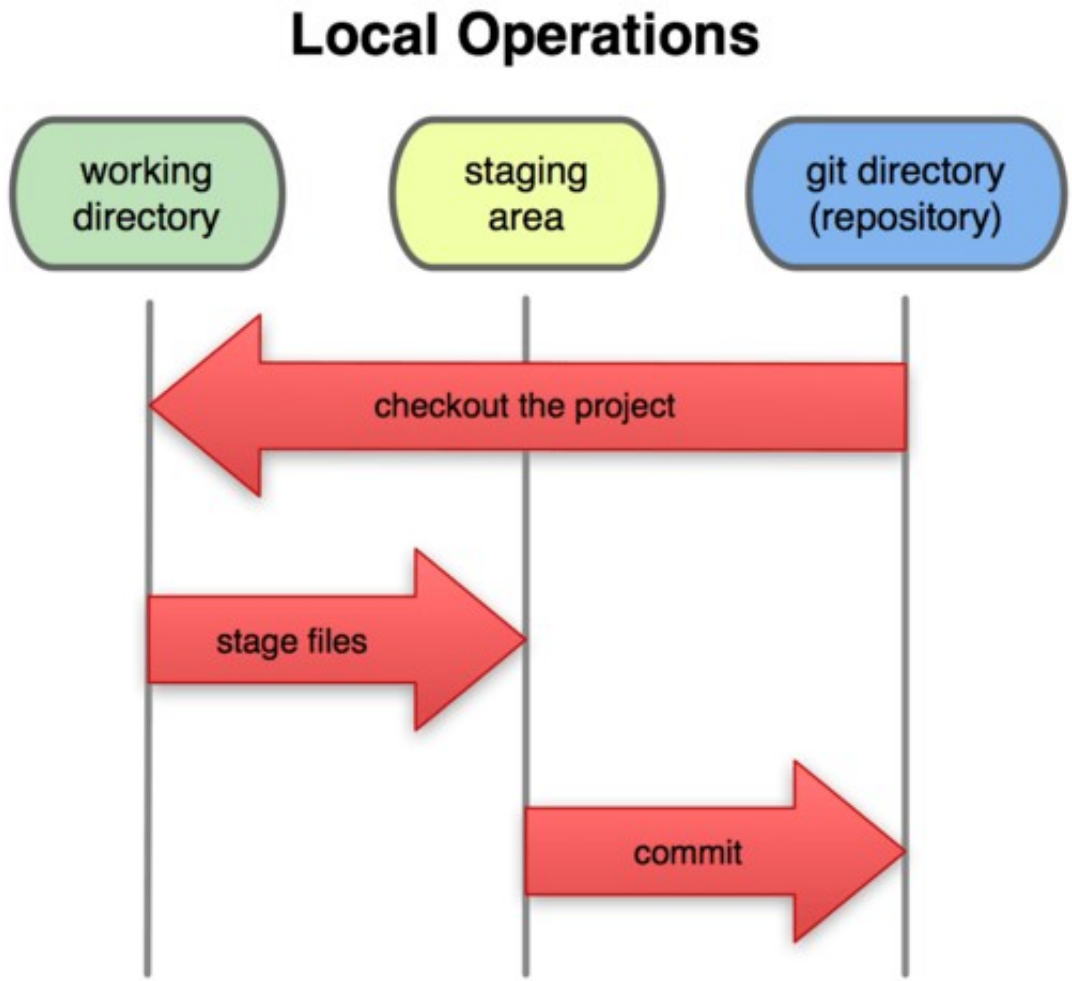
Workflow básico

- **O diretório do Git** é o local onde o Git armazena os metadados e o banco de objetos de seu projeto. Esta é a parte mais importante do Git, e é a parte copiada quando você clona um repositório de outro computador.

- **O diretório de trabalho** é um único checkout de uma versão do projeto.

- **A área de preparação** é um simples arquivo, geralmente contido no seu diretório Git, que armazena informações sobre o que irá em seu próximo commit. É bastante conhecido como índice (index), mas está se tornando padrão chamá-lo de área de preparação.

Workflow básico



Workflow básico

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.

Instalar o GIT

- Capitulo 1 do livro:
- https://git-scm.com/book/pt-br/v1/Primeiros-passos-Instalando-Git
- Após concluir a instalação, você terá tanto uma versão command line (linha de comando, incluindo um cliente SSH que será útil depois) e uma GUI padrão.
- Para checar se a instalação deu certo, digite git no prompt/terminal

git Criando uma identidade

- Definir o seu nome de usuário e endereço de e-mail.
- Importante porque todos os commits no Git utilizam essas informações

```
C:\>git config --global user.name "Paloma Oliveira"

C:\>git config --global user.email paloma.oliveira@ifmg.edu.br
```

- --global: Git sempre usará essa informação para qualquer coisa que você faça nesse sistema.

git Verificando suas configs

- Comando git config –list
- lista todas as configurações que o Git encontrar naquele momento:

```
C:\>git config --list
core.symlinks=false
core.autocrlf=true
user.name=Paloma Oliveira
user.email=paloma.oliveira@ifmg.edu.br
```

Obtendo Ajuda

- Caso você precise de ajuda usando o Git, exitem três formas de se obter ajuda das páginas de manual (manpage) para quaisquer comandos do Git:

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

- Por exemplo, você pode obter a manpage para o comando config executando
- git help config

Exercício

1) Crie um diretório com o nome RepoGit01

2) Crie um arquivo .txt dentro deste diretório com o nome README.txt

3) Digite a seguinte frase dentro do arquivo README.txt

meu primeiro repo git

- Pronto, esse diretório será seu primeiro repositório GIT

Iniciando um Repositório

- Git init – dentro do diretorio que deseja versionar
- Git add <nome do arquivo com extensão>

```
C:\Users\Admin\Documents\RepoGit01>git add README.txt
```

- Git commit -m “Mensagem do Commit”

```
C:\Users\Admin\Documents\RepoGit01>git commit -m "versao inicial"
[master (root-commit) 4d0eb1e] versao inicial
 1 file changed, 1 insertion(+)
 create mode 100644 README.txt
```

git Status de seus Arquivos

- A principal ferramenta utilizada para determinar quais arquivos estão em quais estados é o comando **git status**.
- Se você executar este comando diretamente após uma clonagem, você deverá ver algo similar a isso:

```
C:\Users\Admin\Documents\RepoGit01>git status
On branch master
nothing to commit, working directory clean
```

- o comando lhe mostra em qual branch você se encontra. Por enquanto, esse sempre é o master, que é o padrão;

Exercício

1) Testando o git status

a) Acrescente uma nova linhano arquivo README.txt: Hello GIT

b) Execute o comando git status, o que acontece?

"Changes not staged for commit" — que significa que um arquivo monitorado foi modificado no diretório de trabalho, mas ainda não foi selecionado (staged).

c) Adicione o arquivo README.txt novamente ao repositório: git add

git add é um comando com várias funções — você o utiliza para monitorar novos arquivos, selecionar arquivos, e para fazer outras coisas como marcar como resolvido aquivos com conflito

d) Execute o comando git status novamente, e agora o que acontece?

e) Execute o commit e logo em seguida git status

git Clonando um Repositório

- Caso você queira copiar um repositório Git já existente — por exemplo, um projeto que você queira contribuir
- o comando necessário é git clone <URL>.
- Exemplo: clonar a biblioteca Git do Ruby chamada Grit
- git clone git://github.com/schacon/grit.git

```
C:\Users\Admin\Documents\RepoGit01>git clone git://github.com/schacon/grit.git
Cloning into 'grit'...
remote: Counting objects: 4051, done.
R ceiving objects:  30% (1255/4051), 508.00 KiB | 182.00 KiB/s
```

Exercício

1) Execute o clone da biblioteca Grit

Isso cria um diretório chamado grit, inicializa um diretório .git dentro deste, obtém todos os dados do repositório e verifica a cópia atual da última versão.

2) Verifique o que acontece com seu repositorio – diretório RepoGit01

3) Execute o comando git status, o que acontece?

Exercício

"Untracked files"

- Você pode ver que o seu novo diretorio *grit* não está sendo monitorado, pois está listado sob o cabeçalho "Untracked files" na saída do comando status.
- Não monitorado significa basicamente que o Git está vendo um arquivo que não existia na última captura (commit);
- o Git não vai incluí-lo nas suas capturas de commit até que você o diga explicitamente que assim o faça (git add).

Exercício

4) Execute o comando git add grit em seguida veja o status

- você pode ver que o seu arquivo grit agora está sendo monitorado e está selecionado
- Você pode dizer que ele está selecionado pois está sob o cabeçalho “Changes to be committed”.

Exercício

5) Acrescente uma nova linha no arquivo README.txt: Verificando mudanças

6) Execute o comando git status

7)Adicione o arquivo README.txt novamente ao repositório: git add

8) Execute o comando git status novamente

9) Acrescente uma nova linha no arquivo README.txt: Mudando novamente

10) Execute o comando git status novamente, o que acontece?

Exercício

Agora o arquivo README.txt aparece listado como selecionado e não selecionado. Como isso é possível?

Acontece que o Git seleciona um arquivo exatamente como ele era quando o comando git add foi executado.

Se você fizer o commit agora, a versão do README.txt como estava na última vez que você rodou o comando git add é que será incluída no commit, não a versão do arquivo que estará no seu diretório de trabalho quando rodar o comando git commit.

Se você modificar um arquivo depois que rodou o comando git add, terá de rodar o git add de novo para selecionar a última versão do arquivo

Visualizando Mudanças

- Como saber o que exatamente você alterou?
- O comando git status mostra apenas quais arquivos foram alterados…
- Comando git diff
- Utilizado com frequência para responder estas duas perguntas:
- O que você alterou, mas ainda não selecionou (stage)?
- E o que você selecionou, que está para ser commitado?

Visualizando Mudanças

- Comando git diff
- Apesar do comando git status responder essas duas perguntas de maneira geral…
- O git diff mostra as linhas exatas que foram adicionadas e removidas — o patch, por assim dizer.
- Este comando compara o que está no seu diretório de trabalho com o que está na sua área de seleção (staging). O resultado te mostra as mudanças que você fez que ainda não foram selecionadas.

Visualizando Mudanças

- Se você quer ver o que selecionou que irá no seu próximo commit, pode utilizar git diff --cached ou git diff --staged
- Este comando compara as mudanças selecionadas com o seu último commit

Visualizando Mudanças

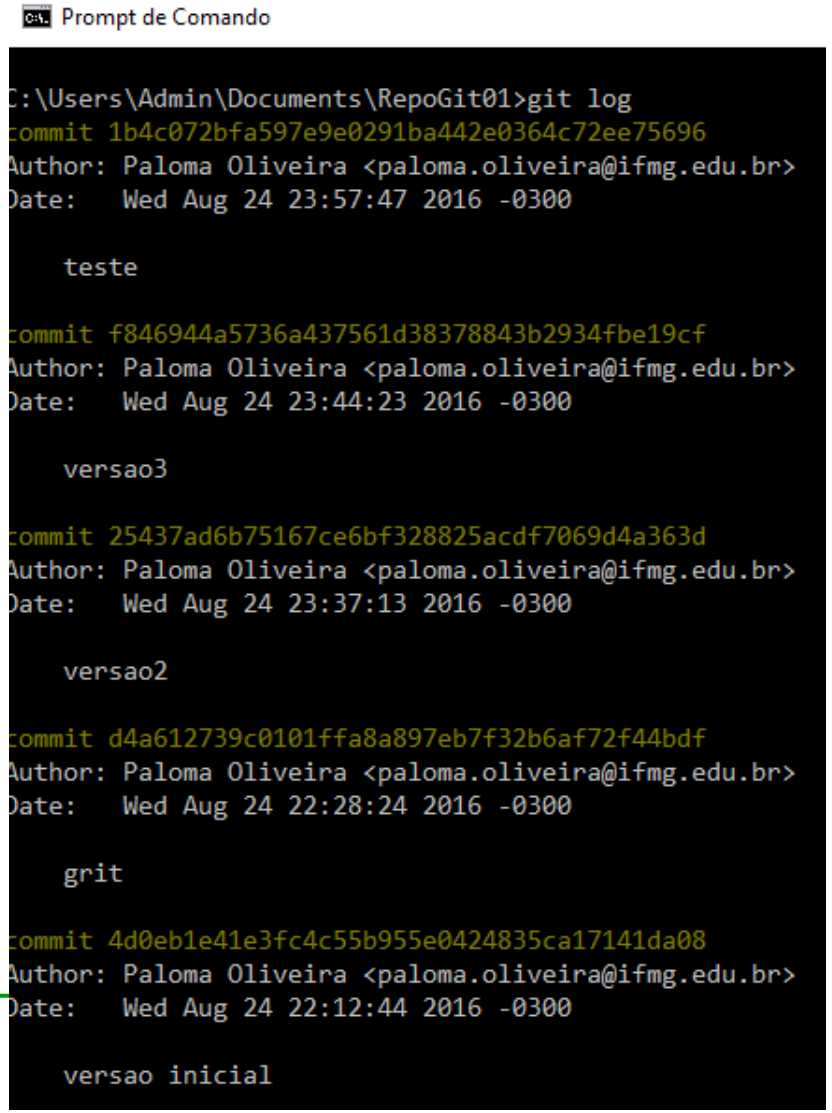
- É importante notar que o git diff por si só não mostra todas as mudanças desde o último commit
- apenas as mudanças que ainda não foram selecionadas.
- Isso pode ser confuso, pois se você selecionou todas as suas mudanças, git diff não te dará nenhum resultado.

Exercício

1) Selecionar o arquivo README.txt e então edite-o, use o git status para ver as mudanças no arquivo que estão selecionadas, e as mudanças que não estão;
2) Agora você pode utilizar o git diff para ver o que ainda não foi selecionado;
3) Execute git diff --cached para ver o que você já alterou para o estado staged até o momento

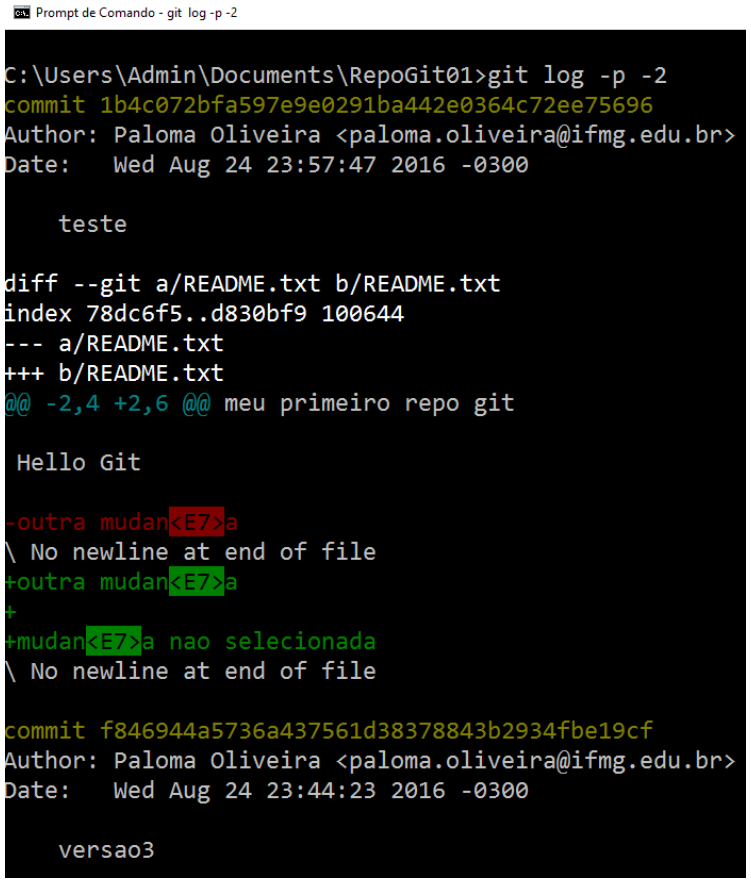
Histórico de Commit

- Após vários commits, você provavelmente vai querer ver o que aconteceu...
- Comando **git log** mostra o histórico de commits;
- commits mais recentes primeiro.
- comando lista cada commit com seu checksum SHA-1, o nome e e-mail do autor, a data e a mensagem do commit.



Histórico de Commit

- Um grande número e variedade de opções para o comando git log estão disponíveis;
- Opções mais usadas:
- -p: mostra o diff introduzido em cada commit.
- -2: limita a saída somente às duas últimas entradas.



Tagging

- Git tem a habilidade de criar tags em pontos específicos na história do código;
- Use esta funcionalidade para marcar pontos de release (v1.0, e por aí vai);
- Listar as tags disponíveis em Git: **git tag**
- Criando Tags: **git tag -a v1.4 -m “my version 1.4”**
  - **-a:** especifica a versão desejada
  - **-m:** define uma mensagem, que é armazenada com a tag.

Tagging

- ver os dados da tag junto com o commit que foi taggeado: **git show**
- Taggeando mais tarde
  - Você também pode taggear commits mais tarde.
    - Para criar a tag no commit, você especifica a chave de verificação (ou parte dela) no final do comando:

git tag -a v1.0 9fceb02

Compartilhando Tags

- Por padrão, o comando git push não transfere tags para os servidores remotos.
- Você deve enviar as tags explicitamente para um servidor compartilhado após tê-las criado.
- **git push origin [nome-tag]**
- Ex: git push origin v1.5
- Para muitas tags ao mesmo tempo: --tags
- **git push origin --tags**

Sumário

- Neste ponto, você pode executar todas as operações locais básicas do Git:
- criar ou clonar um repositório;
- efetuar mudanças;
- fazer o stage e commit de suas mudanças;
- ver o histórico de todas as mudanças do repositório.
- A seguir, vamos trabalhar com um repositório remoto no GitHUB.

Referência

- **Livro: ProGit - Scott Chacon and Ben Straub**
- **https://git-scm.com/book/pt-br/v1/**