GIT - CURSO BÁSICO

Sérgio Vieira
sergiosvieira@gmail.com

previsão de 8 horas

referência: http://git-scm.com/book/pt-br/

AGENDA

- 1 Primeiros Passos
    - 1.1 Sobre Controle de Versão
    - 1.2 Uma Breve História do Git
    - 1.3 Noções Básicas de Git

AGENDA

- 1 Primeiros Passos
    - 1.4 Instalando Git
    - 1.5 Configuração Inicial do Git
    - 1.6 Obtendo Ajuda

AGENDA

- 2. Git Essencial
  - 2.1 Obtendo um Repositório Git
  - 2.2 Gravando Alterações no Repositório
  - 2.3 Visualizando o Histórico de Commits
  - 2.4 Desfazendo Coisas

AGENDA

- 2. Git Essencial
    - 2.5 Trabalhando com Remotos
    - 2.6 Tagging
    - 2.7 Dicas e Truques

AGENDA

- 3. Ramificação (Branching) no Git
- 3.1 O que é um Branch
- 3.2 Básico de Branch e Merge
- 3.3 Gerenciamento de Branches
- 3.4 Fluxos de Trabalho com Branches

AGENDA

- 3. Ramificação (Branching) no Git
- 3.5 Branches Remotos
- 3.6 Rebasing

AGENDA

- 4. Git no Servidor
- 4.1 Os Protocolos
- 4.2 Configurando Git no Servidor
- 4.3 Gerando Sua Chave Pública SSH
- 4.4 Configurando o Servidor
- 4.5 Acesso Público

AGENDA

- 4. Git no Servidor
- 4.6 GitlabHQ
- 4.7 Serviço Git
- 4.8 Git Hospedado

1.1 SOBRE O CONTROLE DE VERSÃO

1.1 SOBRE O CONTROLE DE VERSÃO

- 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.

1.1 SOBRE O CONTROLE DE VERSÃO

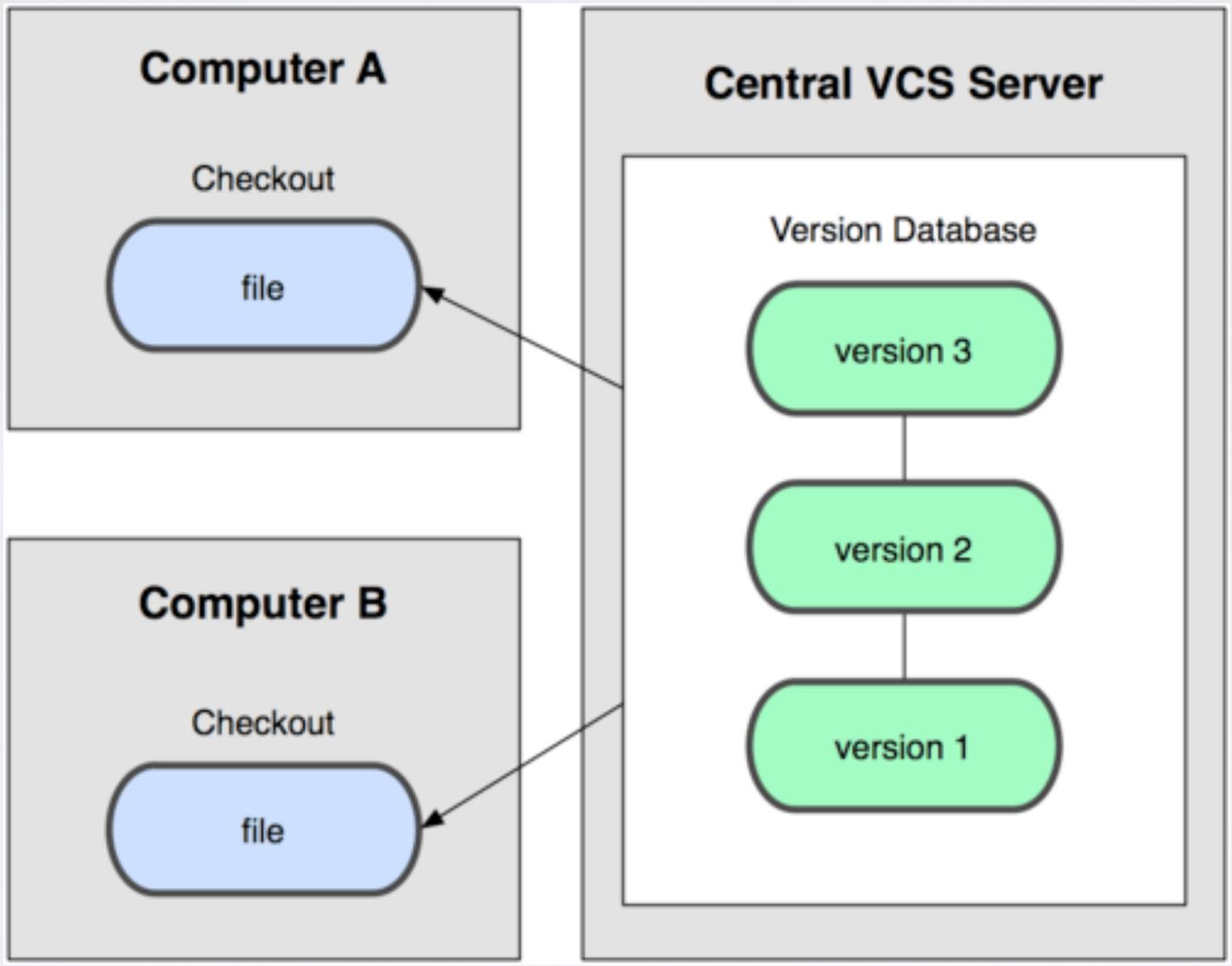


Figura 1-2. Diagrama de Controle de Versão Centralizado.

1.1 SOBRE O CONTROLE DE VERSÃO

Entretanto, esse arranjo também possui grandes desvantagens. O mais óbvio é que 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.

1.1 SOBRE O CONTROLE DE VERSÃO

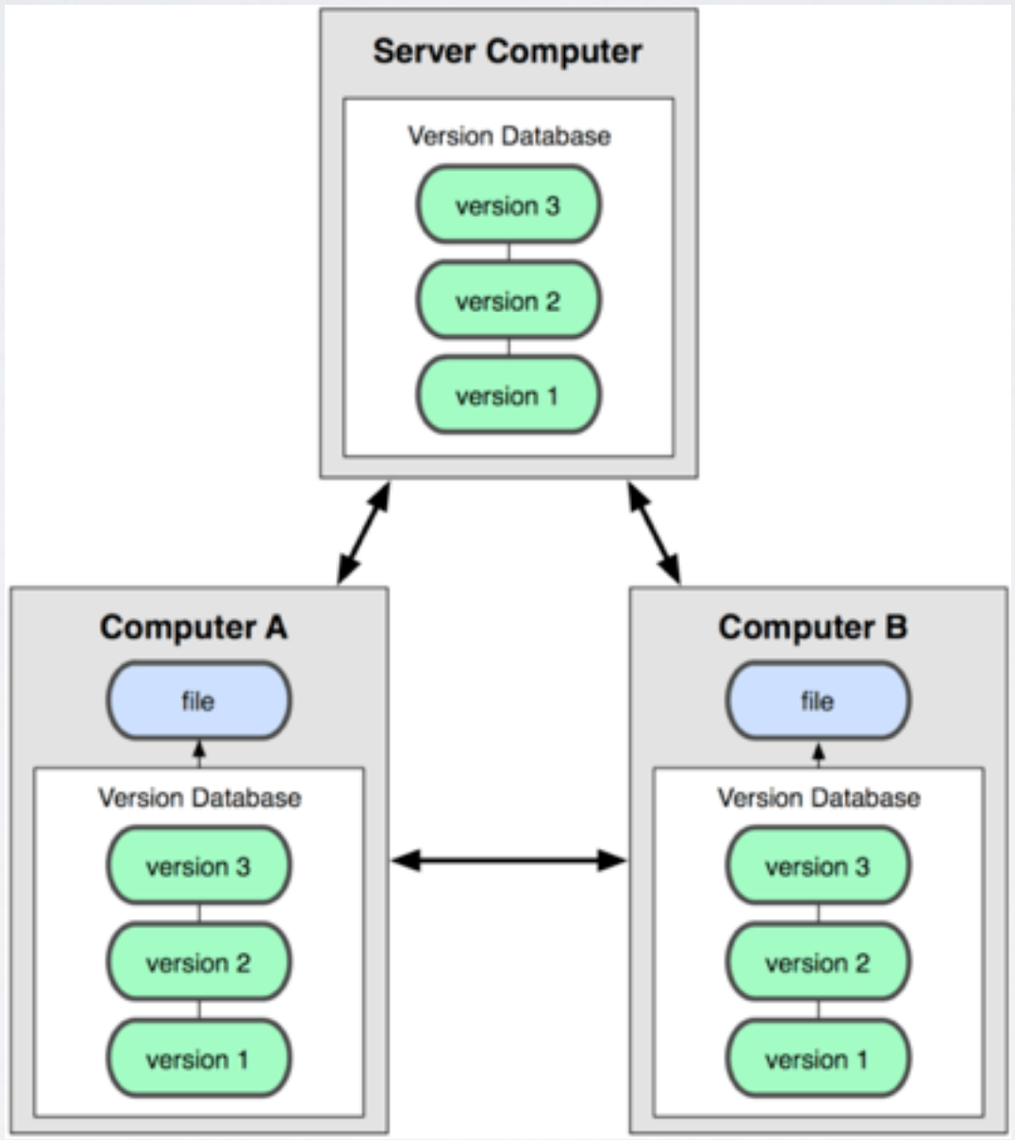


Figura 1-2. Diagrama de Controle de Versão Distribuído.

1.1 SOBRE O CONTROLE DE VERSÃO

- 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.

- Além disso, muitos desses sistemas lidam muito bem com o aspecto de ter vários repositórios remotos com os quais eles podem colaborar, permitindo que você trabalhe em conjunto com diferentes grupos de pessoas, de diversas maneiras, simultaneamente no mesmo projeto

1.2 PRIMEIROS PASSOS

UMA BREVE HISTÓRIA DO GIT

- Durante a maior parte do período de manutenção do kernel do Linux (1991-2002), as mudanças no software eram repassadas como patches e arquivos compactados.

- Em 2002, o projeto do kernel do Linux começou a usar um sistema DVCS proprietário chamado BitKeeper.

- Em 2005, o relacionamento entre a comunidade que desenvolvia o kernel e a empresa que desenvolvia comercialmente o BitKeeper se desfez, e o status de isento-de-pagamento da ferramenta foi revogado. Isso levou a comunidade de desenvolvedores do Linux (em particular Linus Torvalds, o criador do Linux) a desenvolver sua própria ferramenta baseada nas lições que eles aprenderam ao usar o BitKeeper.

UMA BREVE HISTÓRIA DO GIT

- Alguns dos objetivos do novo sistema eram:
    - Velocidade
    - Design simples
    - Suporte robusto a desenvolvimento não linear (milhares de branches paralelos)
    - Totalmente distribuído
    - Capaz de lidar eficientemente com grandes projetos como o kernel do Linux (velocidade e volume de dados)

NOÇÕES BÁSICAS DE GIT

- **Atenção**
- À medida que você aprende a usar o Git, tente não pensar no que você já sabe sobre outros VCSs como Subversion e Perforce; assim você consegue escapar de pequenas confusões que podem surgir ao usar a ferramenta.

NOÇÕES BÁSICAS DE GIT

- **Atenção**
  - Apesar de possuir uma interface parecida, o Git armazena e pensa sobre informação de uma forma totalmente diferente desses outros sistemas; entender essas diferenças lhe ajudará a não ficar confuso ao utilizá-lo.

SNAPSHOTS, E NÃO DIFERENÇAS

NOÇÕES BÁSICAS DE GIT

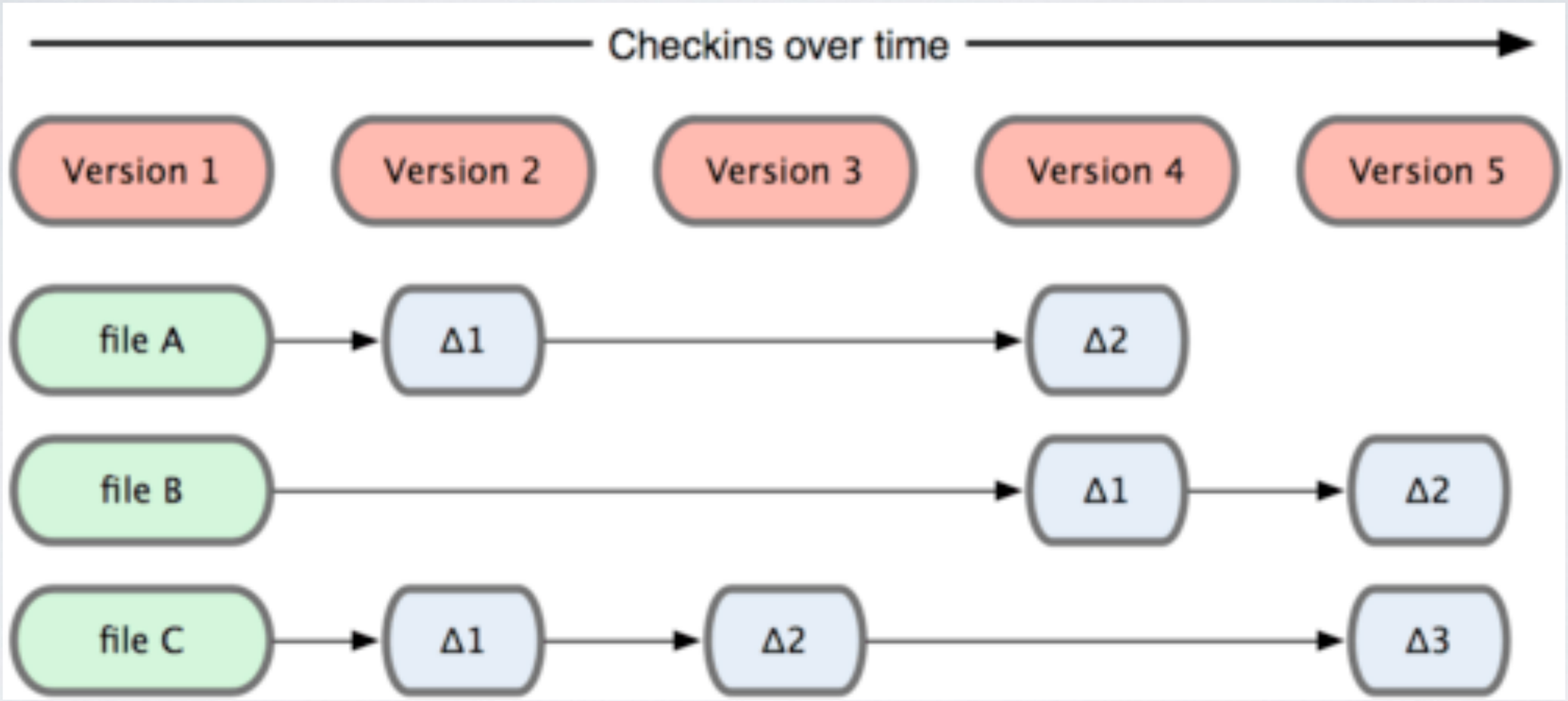


Figura 1-4. Outros sistemas costumam armazenar dados como mudanças em uma versão inicial de cada arquivo.

NOÇÕES BÁSICAS DE GIT

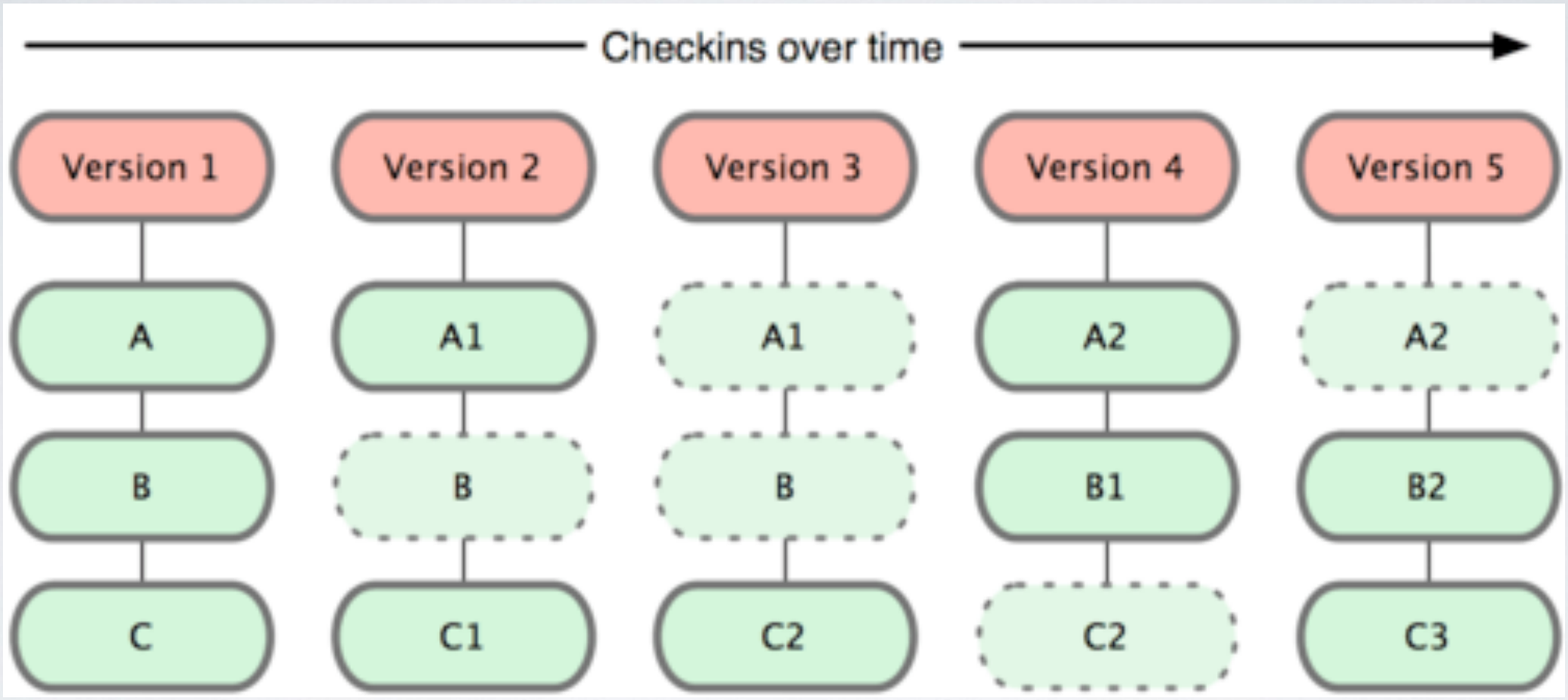


Figura 1-5. Git armazena dados como snapshots do projeto ao longo do tempo.

QUASE TODAS AS OPERAÇÕES SÃO LOCAIS

NOÇÕES BÁSICAS DE GIT

- A maior parte das operações no Git precisam apenas de recursos e arquivos locais para operar — geralmente nenhuma outra informação é necessária de outro computador na sua rede.

- Para navegar no histórico do projeto, o Git não precisa requisitar ao servidor o histórico para que possa apresentar a você — ele simplesmente lê diretamente de seu banco de dados local. Isso significa que você vê o histórico do projeto quase instantaneamente.

GIT TEM INTEGRIDADE

NOÇÕES BÁSICAS DE GIT

- Tudo no Git tem seu checksum (valor para verificação de integridade) calculado antes que seja armazenado e então passa a ser referenciado pelo checksum.

- O mecanismo que o Git usa para fazer o checksum é chamado de hash SHA-1, uma string de 40 caracteres composta de caracteres hexadecimais (0-9 e a-f) que é calculado a partir do conteúdo de um arquivo ou estrutura de um diretório no Git. Um hash SHA-1 parece com algo mais ou menos assim:

24b9da6552252987aa493b52f8696cd6d3b00373

NOÇÕES BÁSICAS DE GIT

- tudo que o Git armazena é identificado não por nome do arquivo mas pelo valor do hash do seu conteúdo.

GIT GERALMENTE SÓ ADICIONA DADOS

NOÇÕES BÁSICAS DE GIT

- Dentre as ações que você pode realizar no Git, quase todas apenas acrescentam dados à base do Git.

- Você pode perder ou bagunçar mudanças que ainda não commitou; mas depois de fazer um commit de um snapshot no Git, é muito difícil que você o perca.

- Isso faz com que o uso do Git seja uma alegria no sentido de permitir que façamos experiências sem o perigo de causar danos sérios.

OS TRÊS ESTADOS

Essa é a coisa mais importante pra se lembrar sobre Git se você quiser que o resto do seu aprendizado seja tranquilo.

NOÇÕES BÁSICAS DE GIT

- Git faz com que seus arquivos sempre estejam em um dos três estados fundamentais: consolidado (committed), modificado (modified) e preparado (staged).

NOÇÕES BÁSICAS DE GIT

- Commited
- Dados são ditos consolidados quando estão seguramente armazenados em sua base de dados local.

NOÇÕES BÁSICAS DE GIT

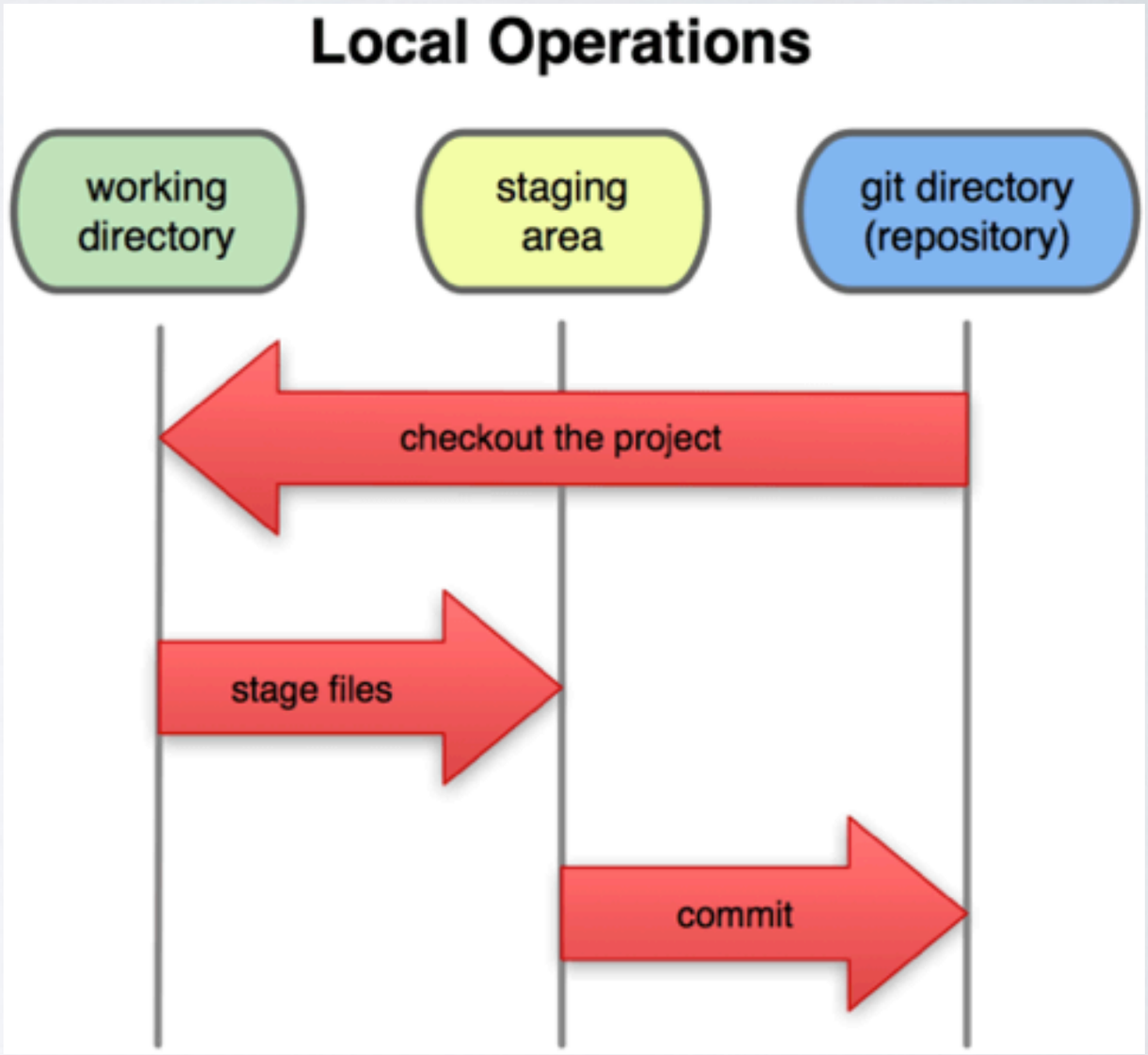
- Modified
- Modificado trata de um arquivo que sofreu mudanças mas que ainda não foi consolidado na base de dados.

NOÇÕES BÁSICAS DE GIT

- 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.

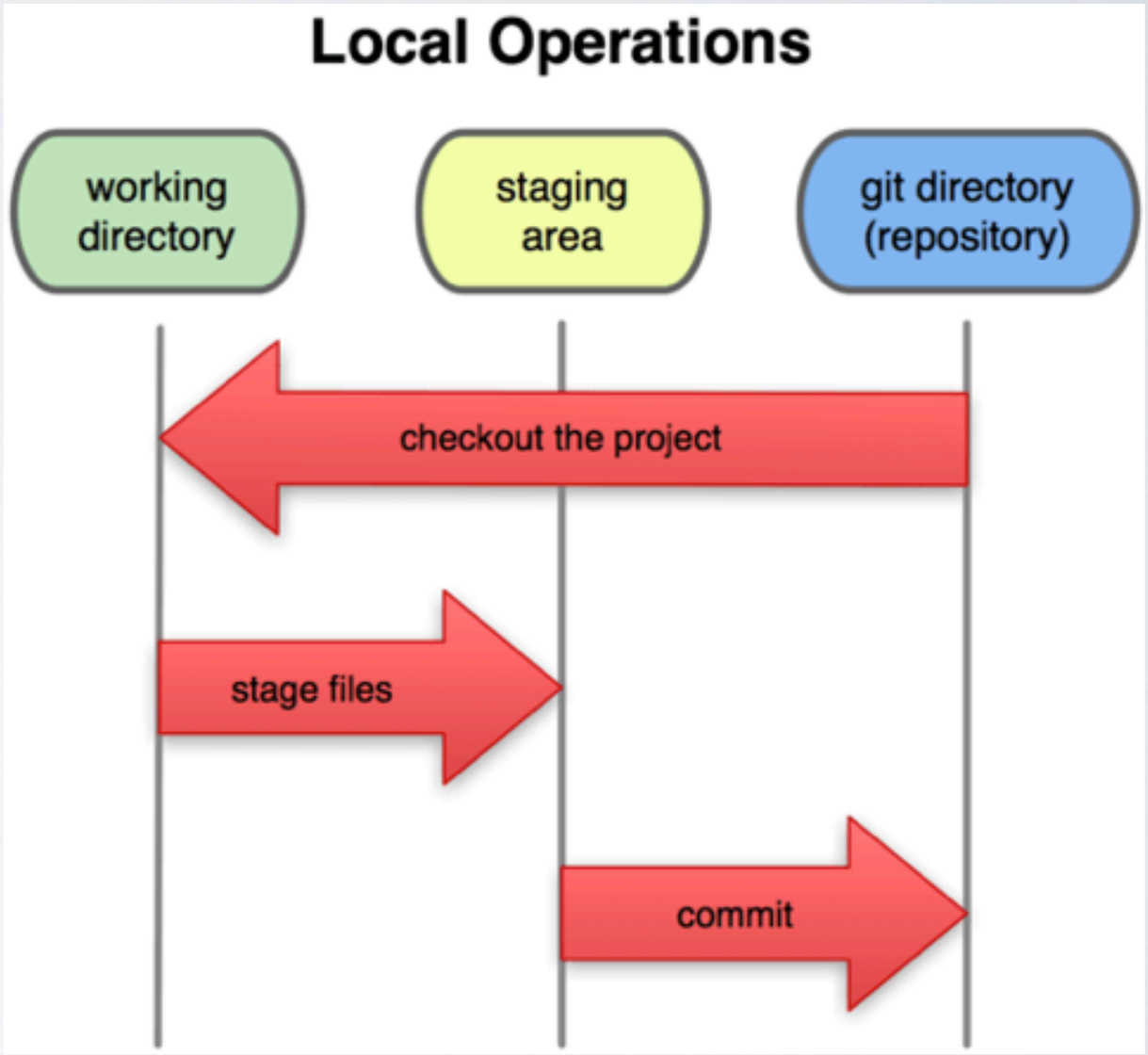
NOÇÕES BÁSICAS DE GIT

- 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.



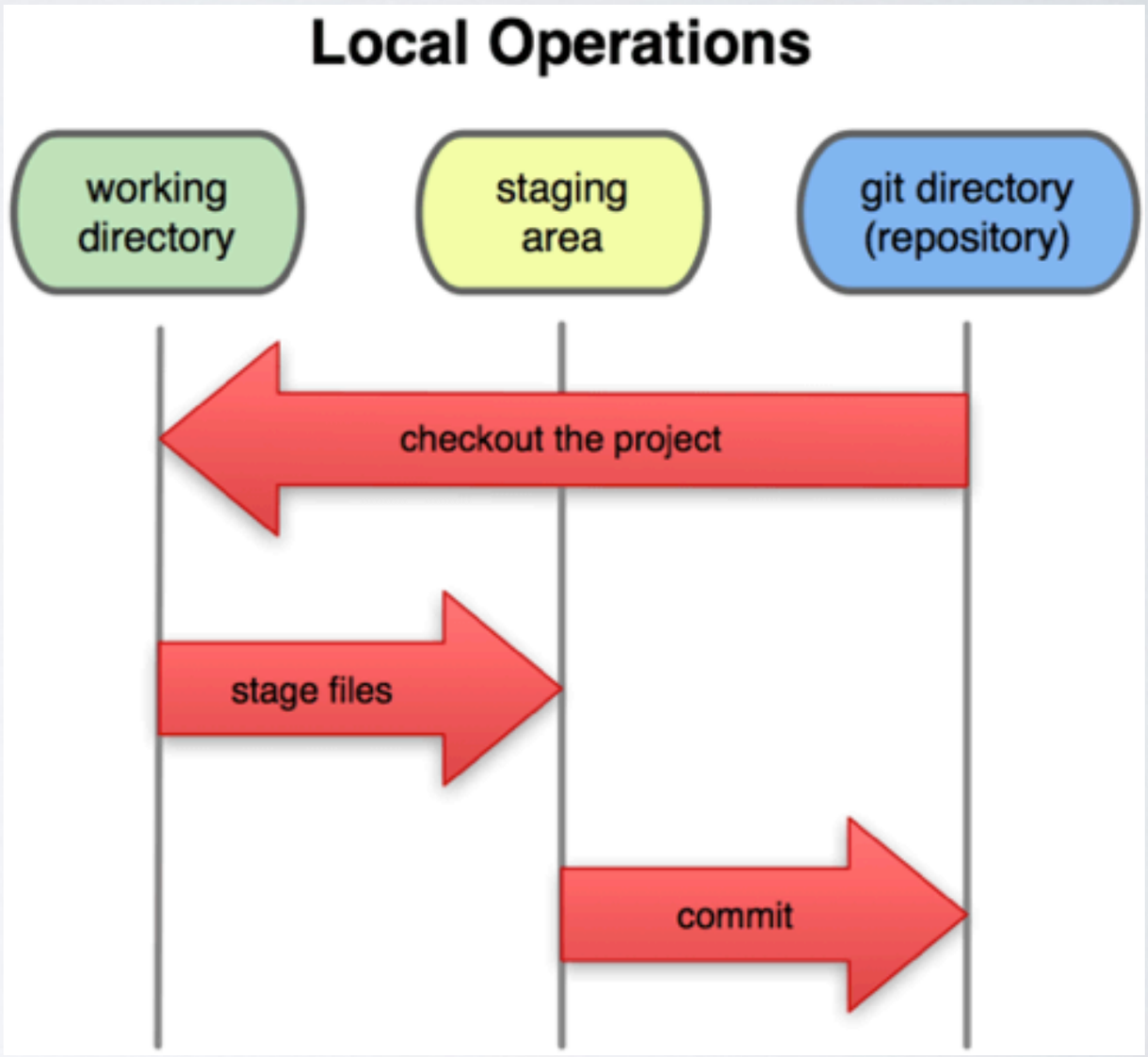
NOÇÕES BÁSICAS DE GIT

- O diretório de trabalho é um único checkout de uma versão do projeto. Estes arquivos são obtidos a partir da base de dados comprimida no diretório do Git e colocados em disco para que você possa utilizar ou modificar.



NOÇÕES BÁSICAS DE GIT

- 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.



NOÇÕES BÁSICAS DE GIT

- O workflow básico do Git pode ser descrito assim:

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.

NOÇÕES BÁSICAS DE GIT

- Se uma versão particular de um arquivo está no diretório Git, é considerada consolidada.

- Caso seja modificada mas foi adicionada à área de preparação, está preparada.

- E se foi alterada desde que foi obtida mas não foi preparada, está modificada.

INSTALANDO GIT

INSTALANDO NO LINUX

NOÇÕES BÁSICAS DE GIT

- Fedora

```
$ sudo yum install git-core
```

- Ubuntu/Debian

```
$ sudo apt-get install git
```

INSTALANDO NO MAC

NOÇÕES BÁSICAS DE GIT

- Instalador gráfico do Git

  http://sourceforge.net/projects/git-osx-installer/

- MacPorts

  $ sudo port install git-core +svn +doc +bash_completion +gitweb

INSTALANDO NO WINDOWS

NOÇÕES BÁSICAS DE GIT

- Instalador

http://msysgit.github.com

CONFIGURAÇÃO INICIAL DO GIT

CONFIGURAÇÃO INICIAL DO GIT

- Git vem com uma ferramenta chamada git config que permite a você ler e definir variáveis de configuração que controlam todos os aspectos de como o Git parece e opera.

- arquivo /etc/gitconfig: Contém valores para todos usuários do sistema e todos os seus repositórios. Se você passar a opção --system para git config, ele lerá e escreverá a partir deste arquivo especificamente.

CONFIGURAÇÃO INICIAL DO GIT

- arquivo ~/.gitconfig: É específico para seu usuário. Você pode fazer o Git ler e escrever a partir deste arquivo especificamente passando a opção --global.

- arquivo de configuração no diretório git (ou seja, .git/config) de qualquer repositório que você está utilizando no momento: Específico para aquele único repositório. Cada nível sobrepõem o valor do nível anterior, sendo assim valores em .git/config sobrepõem aqueles em /etc/gitconfig.

- Em sistemas Windows, Git procura pelo arquivo .gitconfig no diretório $HOME (C:\Documents and Settins\$USER para a maioria das pessoas).

SUA IDENTIDADE

CONFIGURAÇÃO INICIAL DO GIT

- A primeira coisa que você deve fazer quando instalar o Git é definir o seu nome de usuário e endereço de e-mail. Isso é importante porque todos os commits no Git utilizam essas informações, e está imutavelmente anexado nos commits que você realiza:

```
$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com
```

SEU EDITOR

CONFIGURAÇÃO INICIAL DO GIT

- Agora que sua identidade está configurada, você pode configurar o editor de texto padrão que será utilizado quando o Git precisar que você digite uma mensage.

```
$ git config --global core.editor emacs
```

SUA FERRAMENTA DIFF

CONFIGURAÇÃO INICIAL DO GIT

- Outra opção útil que você pode querer configurar é a ferramente padrão de diff utilizada para resolver conflitos de merge (fusão).

- Git aceita kdiff3, tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, ecmerge e opendiff como ferramentas válidas para merge.

```
$ git config --global merge.tool vimdiff
```

VERIFICANDO SUAS CONFIGURAÇÕES

CONFIGURAÇÃO INICIAL DO GIT

- Caso você queira verificar suas configurações, você pode utilizar o comando git config --list para listar todas as configurações que o Git encontrar naquele momento:

```
$ git config --list
user.name=Scott Chacon
user.email=schacon@gmail.com

...

$ git config user.name
Scott Chacon
```

OBTENDO AJUDA

CONFIGURAÇÃO INICIAL DO GIT

- 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
```

GIT ESSENCIAL

INICIALIZANDO UM REPOSITÓRIO EM UM DIRETÓRIO EXISTENTE

- Caso você esteja iniciando o monitoramento de um projeto existente com Git, você precisa ir para o diretório do projeto e digitar:

```
$ git init
```

INICIALIZANDO UM REPOSITÓRIO EM UM DIRETÓRIO EXISTENTE

- Isso cria um novo subdiretório chamado .git que contem todos os arquivos necessários de seu repositório — um esqueleto de repositório Git.

```
$ ls -lah
drwxr-xr-x   7 sergiosvieira  staff   238B 27 Jul 22:27 .
drwxr-xr-x  10 sergiosvieira  staff   340B 27 Jul 16:11 ..
drwxr-xr-x  15 sergiosvieira  staff   510B 27 Jul
22:29 .git
```

CLONANDO UM REPOSITÓRIO EXISTENTE

- Caso você queira copiar um repositório Git já existente — por exemplo, um projeto que você queira contribuir — o comando necessário é git clone.

- Caso você esteja familiarizado com outros sistemas VCS, tais como Subversion, você perceberá que o comando é clone e não checkout.

- Cada versão de cada arquivo no histórico do projeto é obtida quando você roda git clone.

CLONANDO UM REPOSITÓRIO EXISTENTE

- Você clona um repositório com **git clone [url]**:

$ git clone https://github.com/sergiosvieira/curso-git.git

CLONANDO UM REPOSITÓRIO EXISTENTE

- Isso cria um diretório chamado curso-git e inicializa um subdiretório.git dentro.

- Obtém todos os dados do repositório e verifica a cópia atual da última versão.

CLONANDO UM REPOSITÓRIO EXISTENTE

- Caso você queira clonar o repositório em um diretório diferente de curso-git, é possível especificar esse diretório utilizando a opção abaixo:

$ git clone https://github.com/sergiosvieira/curso-git.git meucurso-git

GRAVANDO ALTERAÇÕES NO REPOSITÓRIO

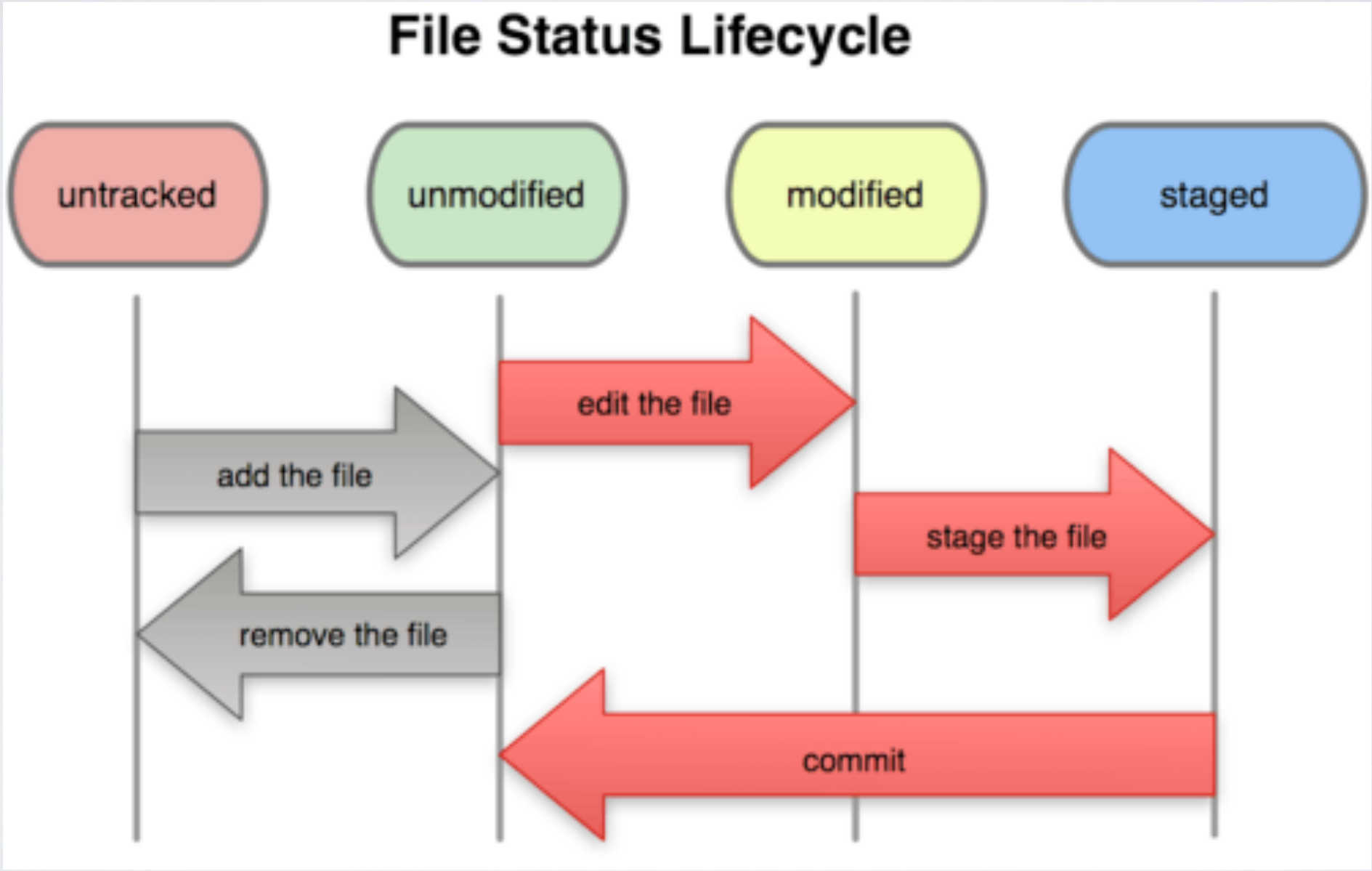


Figura 2-1. O ciclo de vida dos status de seus arquivos.

VERIFICANDO O 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:

```
$ git status
# On branch master
nothing to commit, working directory clean
```

VERIFICANDO O STATUS DE SEUS ARQUIVOS

- Adicione um novo arquivo em seu projeto e execute **git status** novamente.

```
$ touch NEWFILE
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#    NEWFILE
nothing added to commit but untracked files present (use "git add"
to track)
```

MONITORANDO NOVOS ARQUIVOS

- Para passar a monitorar um novo arquivo, use o comando git add.

$ git add NEWFILE

- Rode o comando status novamente:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    new file:   NEWFILE
#
```

SELECIONANDO ARQUIVOS MODIFICADOS

- Para listar os arquivos e pastas que estão sendo monitorados use:

$ git ls-files
README
pasta01/arquivo01
pasta01/arquivo02
pasta02/arquivo01
pasta02/arquivo02
pasta02/arquivo03

- Modifique o arquivo README

$ echo “Curso de Git” >> README

SELECIONANDO ARQUIVOS MODIFICADOS

- Rode novamente git status e você verá algo do tipo:

$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    new file: NEWFILE
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
    modified: README

SELECIONANDO ARQUIVOS MODIFICADOS

- Rode novamente git status e você verá algo do tipo:

$ git status

```
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    modified:   README
    new file:   NEWFILE
```

SELECIONANDO ARQUIVOS MODIFICADOS

- Faça uma nova mudança no arquivo README e rode git status mais uma vez:

```
$ echo “Primeiro dia” >> NEWFILE
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        modified:   README
        new file:   NEWFILE
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
        modified:   NEWFILE
```

SELECIONANDO ARQUIVOS MODIFICADOS

- 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 NEWFILE 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:

SELECIONANDO ARQUIVOS MODIFICADOS

```
$ git add README
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    new file:   NEWFILE
#    modified:   README
#
```

IGNORANDO ARQUIVOS

- Muitas vezes, você terá uma classe de arquivos que não quer que o Git automaticamente adicione ou mostre como arquivos não monitorados.

- Normalmente estes arquivos são gerados automaticamente como arquivos de log ou produzidos pelo seu sistema de build.

- Nestes casos, você pode criar um arquivo contendo uma lista de padrões a serem checados chamado .gitignore.

IGNORANDO ARQUIVOS

- Primeiro crie os arquivos de build usando:

```
$ sh make-build-files.sh
```

- Rode git status e verifique a existência de uma nova pasta em untracked files:

```
$ git status
```

- Lista os arquivos criados:

```
$ ls build
objeto.a  objeto.o
```

IGNORANDO ARQUIVOS

- Crie o arquivo .gitignore e adicione o seguinte padrão:

```
$ echo “build” >> .gitignore
```

- Exemplos de padrões:
  - *.o
  - *.a
  - *.[oa]

IGNORANDO ARQUIVOS

- Existe um site que gera o arquivo gitignore para você.
- Basta especificar o nome da linguagem ou editor para que ele crie a lista de arquivos que deve ser ignorados.

http://www.gitignore.io/

VISUALIZANDO SUAS MUDANÇAS SELECIONADAS E NÃO SELECIONADAS

- O que você alterou, mas ainda não selecionou (stage)?
- E o que você selecionou, que está para ser commitado?
- Apesar do comando git status responder essas duas perguntas de maneira geral, o git diff mostra as linhas exatas que foram adicionadas e removida

VISUALIZANDO SUAS MUDANÇAS SELECIONADAS E NÃO SELECIONADAS

- Para ver o que você alterou mas ainda não selecionou, digite o comando git diff sem nenhum argumento:

- E o que você selecionou, que está para ser commitado?

- git diff —cached ou git diff-staged

VISUALIZANDO SUAS MUDANÇAS SELECIONADAS E NÃO SELECIONADAS

- diff --git a/textot.txt b/textot.txt
  index 9c15ed1..574c337 100644
  --- a/textot.txt
  +++ b/textot.txt
  @@ -1 +1,2 @@
   linha 01
  +linha2

CONSOLIDADO AS MODIFICAÇÕES

- Agora que a sua área de seleção está do jeito que você quer, você pode fazer o commit de suas mudanças.
- Lembre-se que tudo aquilo que ainda não foi selecionado — qualquer arquivo que você criou ou modificou que você não tenha rodado o comando git add desde que editou — não fará parte deste commit.

$ git commit

CONSOLIDANDO AS MODIFICAÇÕES

- Você pode ver que a mensagem default do commit contém a última saída do comando git status comentada e uma linha vazia no início.

- Para um lembrete ainda mais explícito do que foi modificado, você pode passar a opção -v para o git commit.

- Ao fazer isso, aparecerá a diferença (diff) da sua mudança no editor para que possa ver exatamente o que foi feito.

```
$ git commit -v
```

CONSOLIDANDO AS MODIFICAÇÕES

- Alternativamente, você pode digitar sua mensagem de commit junto ao comanto commit ao especificá-la após a flag -m, assim:

```
$  git commit -m "Adiciona linha 01 no arquivo 01"
[master c9e762f] Adiciona linha 01 no arquivo 01
 1 file changed, 1 insertion(+)
```

PULANDO A ÁREA DE SELEÇÃO

- Se você quiser pular a área de seleção, o Git provê um atalho simples. Informar a opção -a ao comando git commit faz com que o Git selecione automaticamente cada arquivo que está sendo monitorado antes de realizar o commit, permitindo que você pule a parte do git add

```
$ git commit -a -m 'added new benchmarks'
```

REMOVENDO ARQUIVOS

- Para remover um arquivo do Git, você tem que removê-lo dos arquivos que estão sendo monitorados (mais precisamente, removê-lo da sua área de seleção) e então fazer o commit.

- O comando git rm faz isso e também remove o arquivo do seu diretório para você não ver ele como arquivo não monitorado (untracked file) na próxima vez.

```
$ git rm remova-me
```

REMOVENDO ARQUIVOS

- Outra coisa útil que você pode querer fazer é manter o arquivo no seu diretório, mas apagá-lo da sua área de seleção.

- Em outras palavras, você quer manter o arquivo no seu disco rígido mas não quer que o Git o monitore mais.

- Isso é particularmente útil se você esqueceu de adicionar alguma coisa no seu arquivo .gitignore e acidentalmente o adicionou, como um grande arquivo de log ou muitos arquivos .a compilados.

- Para fazer isso, use a opção --cached:

```
$ git rm --cached remova-me
```

REMOVENDO ARQUIVOS

- Você pode passar arquivos, diretórios, e padrões de nomes de arquivos para o comando `git rm`. Isso significa que você pode fazer coisas como:

```
$ git rm pasta/*.log
```

MOVENDO ARQUIVOS

- Diferente de muitos sistemas VCS, o Git não monitora explicitamente arquivos movidos.

- Se você renomeia um arquivo, nenhum metadado é armazenado no Git que identifique que você renomeou o arquivo.

- No entanto, o Git é inteligente e tenta descobrir isso depois do fato.

MOVENDO ARQUIVOS

- É um pouco confuso que o Git tenha um comando `mv`. Se você quiser renomear um arquivo no Git, você pode fazer isso com:

```
$ git mv nome_atual novo_nome
```

MOVENDO ARQUIVOS

- É um pouco confuso que o Git tenha um comando `mv`. Se você quiser renomear um arquivo no Git, você pode fazer isso com:

```
$ git mv nome_atual novo_nome
```

VISUALIZANDO O HISTÓRICO DE COMMITS

VISUALIZANDO O HISTÓRICO DE COMMITS

- Depois que você tiver criado vários commits, ou se clonou um repositório com um histórico de commits existente, você provavelmente vai querer ver o que aconteceu.

- A ferramente mais básica e poderosa para fazer isso é o comando git log.

```
$ git clone https://github.com/zedapp/zed.git
$ git log
```

VISUALIZANDO O HISTÓRICO DE COMMITS

- Depois que você tiver criado vários commits, ou se clonou um repositório com um histórico de commits existente, você provavelmente vai querer ver o que aconteceu.

- A ferramente mais básica e poderosa para fazer isso é o comando git log.

```
$ git clone https://github.com/zedapp/zed.git
$ git log
```

VISUALIZANDO O HISTÓRICO DE COMMITS

- Por padrão, sem argumentos, git log lista os commits feitos naquele repositório em ordem cronológica reversa. Isto é, os commits mais recentes primeiro.

- Como você pode ver, este comando lista cada commit com seu checksum SHA-1, o nome e e-mail do autor, a data e a mensagem do commit.

VISUALIZANDO O HISTÓRICO DE COMMITS

```
commit
e7ee713fcb97d6ff16c168d0e7f6079c334d17cc
Merge: a1e22b9 4c31212
Author: Zef Hemel <zef@zef.me>
Date:   Fri Aug 22 18:41:36 2014 +0200
Merge pull request #438 from
TheKiteEatingTree/scrollbars
Toggle Native Scroll Bars
```

VISUALIZANDO O HISTÓRICO DE COMMITS

- Um grande número e variedade de opções para o comando git log estão disponíveis para mostrá-lo exatamente o que você quer ver.

- Uma das opções mais úteis é -p, que mostra o diff introduzido em cada commit. Você pode ainda usar -2, que limita a saída somente às duas últimas entradas.

```
$ git log -p -2
```