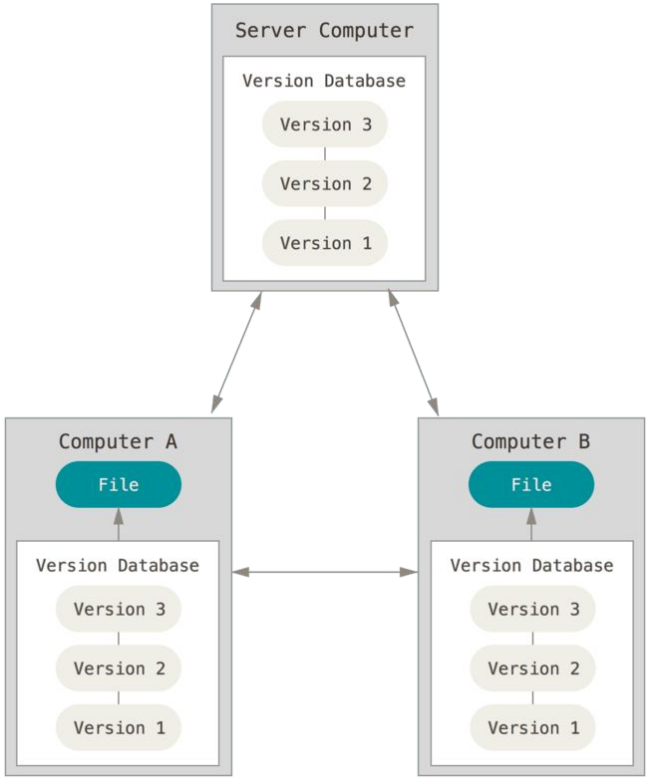
Thiago R. M. Bitencourt

Controle de versão

- Controle de versão é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo para que você possa lembrar versões específicas mais tarde

Sistemas distribuídos de controle de versão



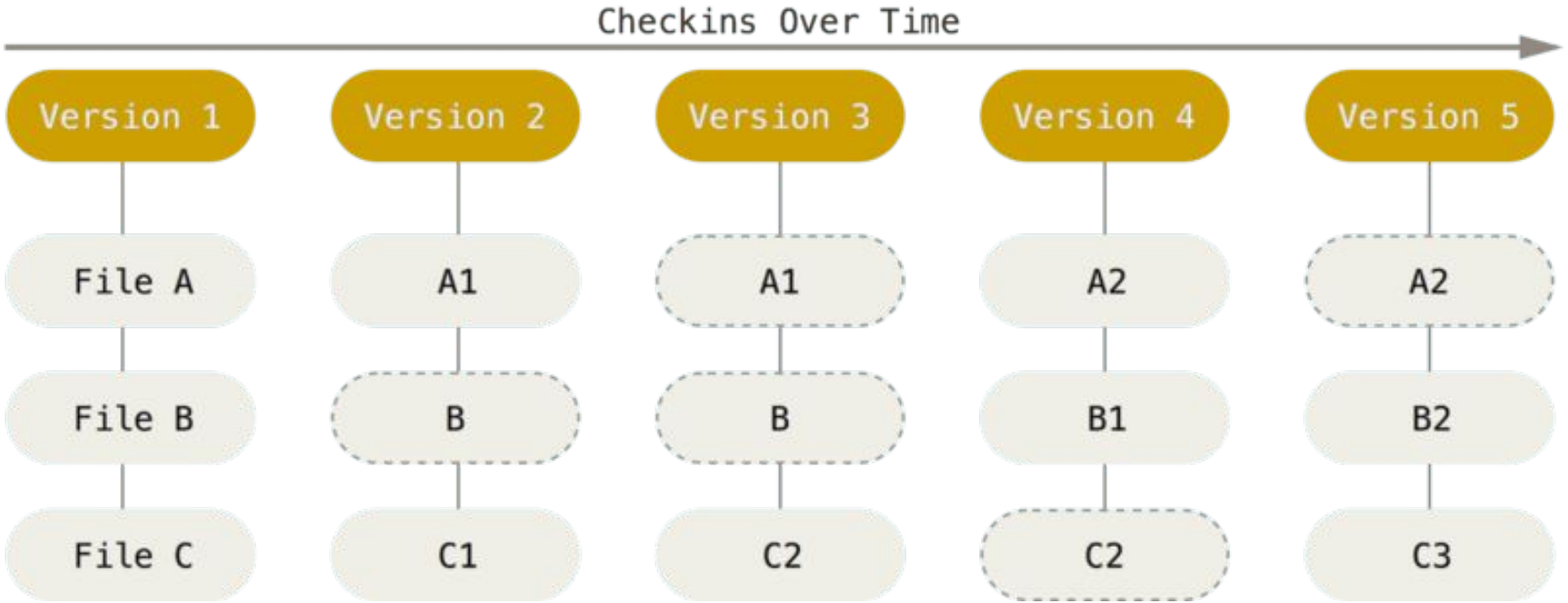
Snapshots (fotos)

- O Git gerencia os arquivos como um conjunto de *snapshots* de um sistema de arquivos em miniatura

- Sempre que ocorre alguma alteração no projeto o Git *“tira uma foto”* do estados dos arquivos no momento e guarda uma referência para esta *foto*

- Para ser mais eficiente, se um arquivo não sofreu alteração o Git não salva o arquivo novamente mas apenas cria um link para a versão anterior armazenada

Snapshots


Checkins Over Time
Version 1
Version 2
Version 3
Version 4
Version 5
File A
A1
A1
A2
A2
File B
B
B
B1
B2
File C
C1
C2
C2
C3

Quase todas as operações são locais

- O git mantém uma cópia completa do projeto localmente
- Todos os arquivos e histórico estão disponíveis
- É possível trabalhar totalmente offline
- E sincronizar com o repositório centralizado apenas quando necessário

Git é íntegro

- Tudo no Git é check-sumeizado (*hash*) antes de ser armazenado

- Por isso, é **impossível(?)** mudar o conteúdo de um arquivo sem que o Git fique sabendo

- Não haverá perda de informações ou arquivos corrompidos sem que o Git fique sabendo

Git é íntegro

- O mecanismo de hash que o Git utiliza é o SHA-1
- SHA-1 é um algoritmo que cria uma *string* de 40 caracteres **única(?)** que é calculada baseado no conteúdo de um arquivo ou de um diretório
- Um hash SHA-1 se parece com isso:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

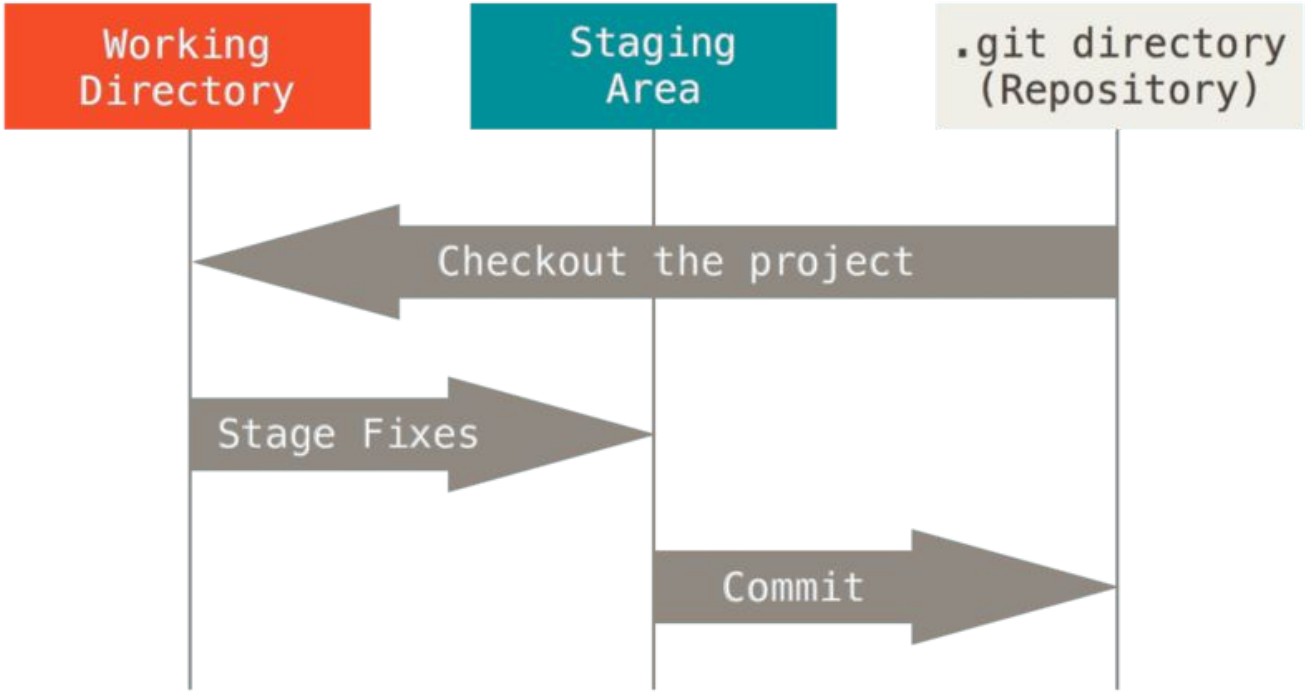
- Você verá muito isso enquanto usa Git

Os três estados

Committed, modified, staged

- **Comitado**: Os dados estão seguros armazenado no repositório local
- **Modificado**: O arquivo sofreu alguma alteração que ainda não foi *comitada*
- **Staged**: O arquivo foi marcado para ser comitado

As três áreas



As três áreas

- *.git repository*: É onde o git armazena os metadados e a base de dados do projeto. É a parte mais importante do Git, e é copiado quando um repositório é clonado

- *Working tree*: É uma versão do projeto na qual é possível se trabalhar

- *Staging area*: É um arquivo do Git que contém informações de quais arquivos estarão no próximo commit

Instalando o Git

Linux:

Linha de comando

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

Windows:

Baixe e instale

https://git-scm.com/download/win

Configuração inicial

- Após instalar o Git é necessário fazer algumas configurações iniciais

- Fará isso apenas **uma vez** após a instalação

- Ferramenta `git config` permite ver e atribuir variáveis de configuração

Configurações

- As variáveis de configuração podem ser armazenadas em três lugares diferentes
  - **/etc/gitconfig**: válido para todos os usuário do sistema e todos os repositórios. Para gravar neste arquivo passe o parâmetro `--system` para o `git config`.
  - **~/.gitconfig** ou **~/.config/git/config**: válido somente para o seu usuário. Para gravar neste arquivo passe o parâmetro `--global` para o `git config`.
  - **.git/config dentro do repositório**: válido apenas para o repositório atual.
- Cada nível sobrescreve os valores do nível anterior

Sua identidade

- A primeira coisa que deve ser feito ao instalar o Git é configurar seu nome de usuário e endereço de e-mail

- Cada *commit* feito utiliza esta informação que será imutavelmente “carimbada” nos commits que você criar

- **Comandos**

```
$ git config --global user.name “Fulano de Tal”
$ git config --global user.email fulanodetal@exemplo.com.br
```

Inicializando um novo repositório

Crie ou acesse um diretório que será seu repositório Git

```
$ cd /c/Users/{username}/{repositorio}
```

Comando

```
$ git init
```

Será criado um novo subdiretório **.git** que contém todos os arquivos necessários para o seu repositório.

Clonando um repositório existente

Cara obter uma cópia de um repositório existente usa-se o comando:

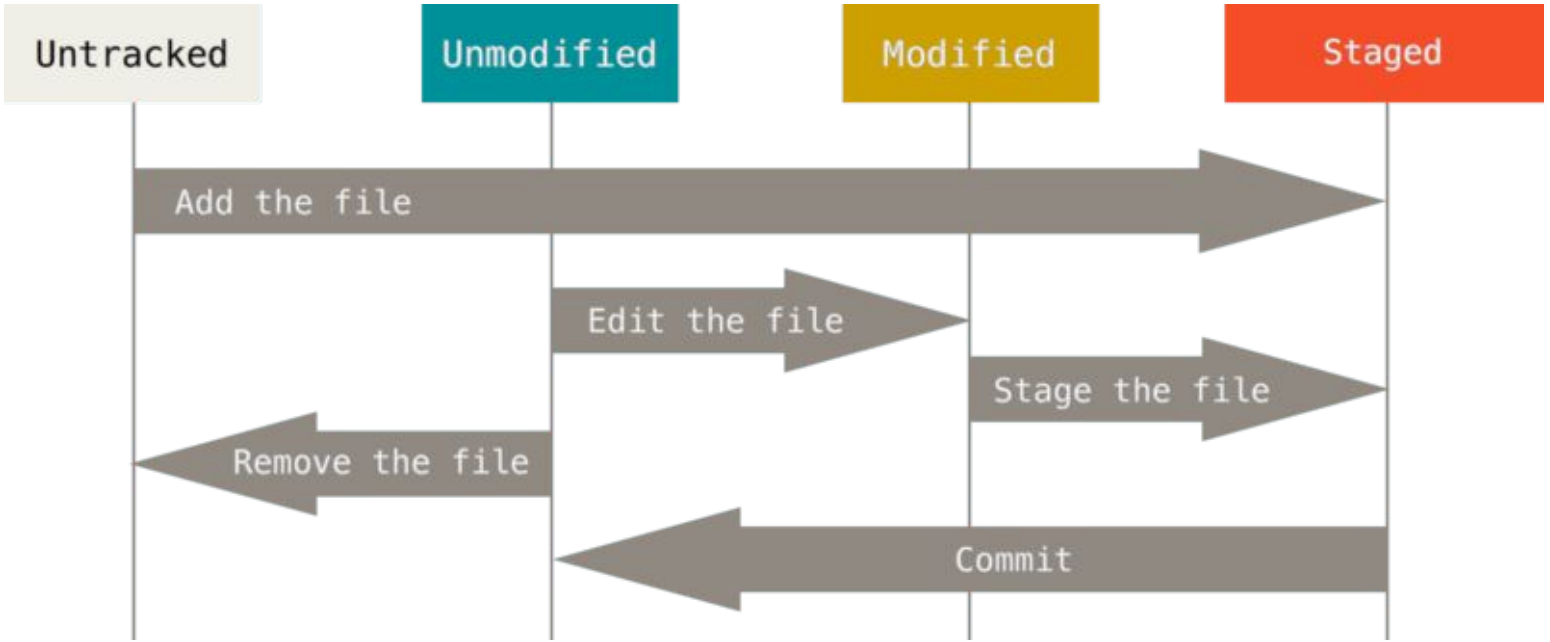
```
$ git clone [url]
```

Ao clonar um repositório todos os arquivos, assim como todo o histórico, é baixado para o repositório local

Comando

```
$ git clone https://github.com/thiagobitencourt/webdev-2
```

Mudanças no repositório



Estado do repositório

Comando

```
$ git status
```

Resultado

```
On branch master
Your branch is up-to-date with 'origin/master'
nothing to commit, working directory clean
```

Arquivo *untracked*

Após adicionar um novo arquivo, verificando o estado do repositório

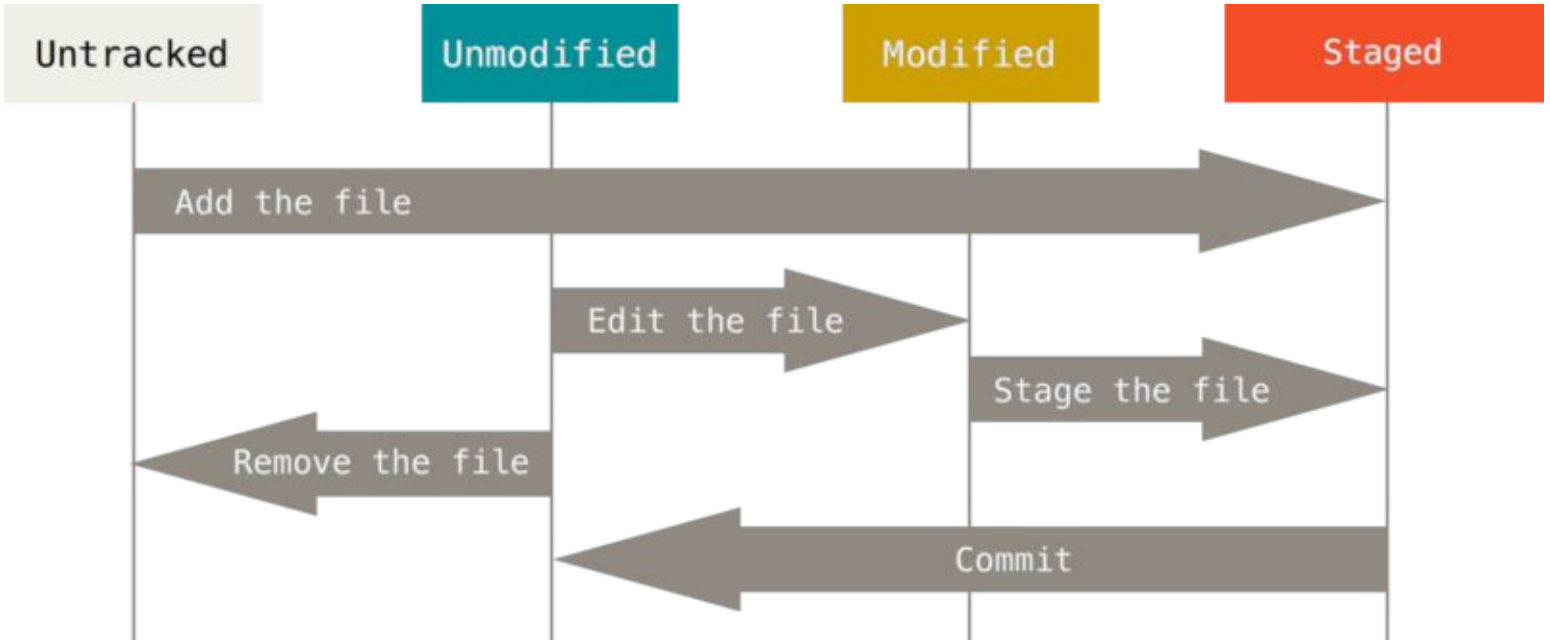
```
$ echo 'My Project' > CONTRIBUTING.md


$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    CONTRIBUTING.md

nothing added to commit but untracked files present (use "git add" to track)
```

Mudanças no repositório



Tracking arquivos

Adicionar um arquivo para ser *rastreado* pelo Git

```
$ git add CONTRIBUTING.md
```

Verificando o estado do repositório após adicionar o arquivo

```
$ 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:  CONTRIBUTING.md
```

Arquivos modificados

Após **alterar um arquivo** rastreado/versionado pelo Git, verificar o estado do repositório

```
$ 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:   CONTRIBUTING.md

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
```

Staging arquivos modificados

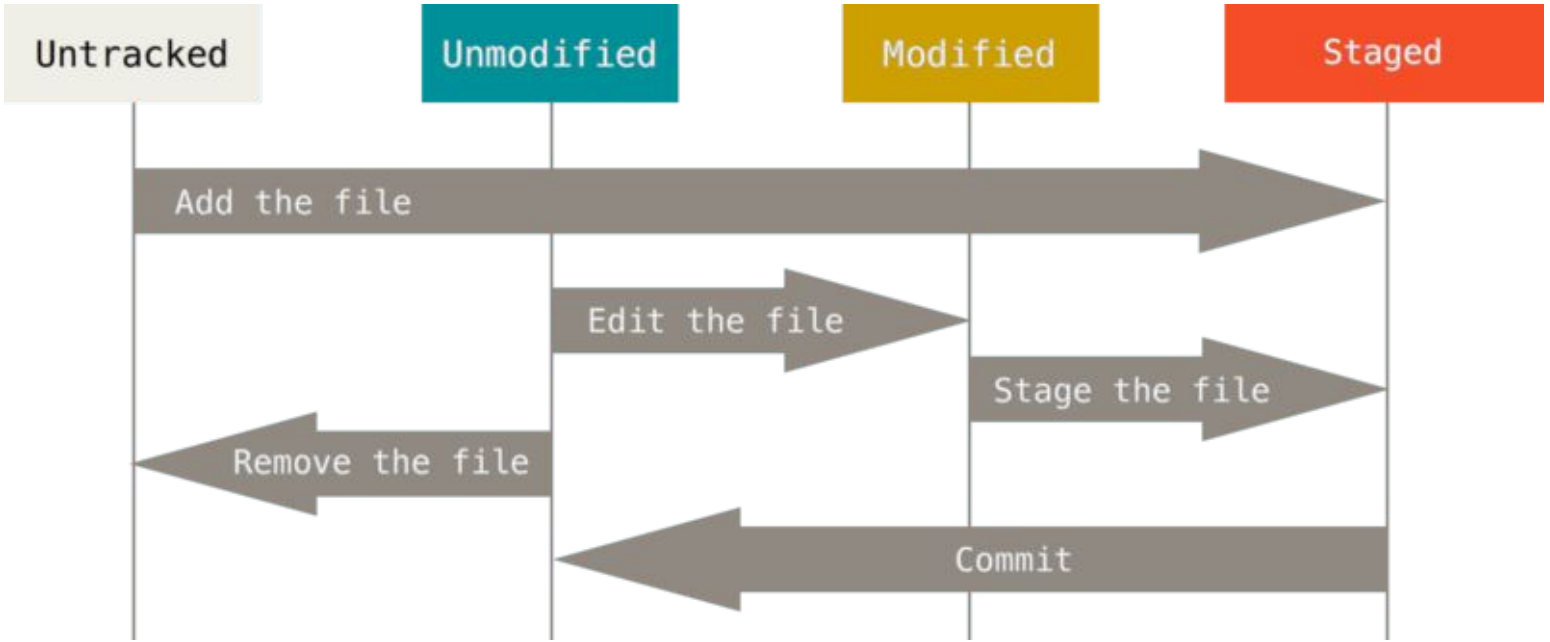
Marcar o arquivo modificado para ser *comitado*

```
$ git add README


$ 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:   CONTRIBUTING.md
    modified:   README
```

Mudanças no repositório



Arquivos modificados novamente

Esquecemos um detalhe no arquivo **README**, e fizemos uma nova alteração

```
$ vim README
$ 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:   CONTRIBUTING.md
    modified:   README

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
```

Staging arquivos modificados

Marcar o arquivo modificado para ser *comitado*

```
$ git add README


$ 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:   CONTRIBUTING.md
    modified:   README
```

Ignorando arquivos

Algumas vezes queremos ignorar arquivos, e não queremos que o Git rastreie este arquivo

Nem mesmo queremos que o Git nos mostre como arquivo *Untracked*

Para isso, criamos o arquivo `.gitignore`

Arquivo .gitignore

```
# no .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in the build/ directory
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory
doc/**/*.pdf
```

Ver o que foi alterado

Com **`git status`** podemos ver quais arquivos foram alterados, mas não vemos o que **exatamente** foi alterado

Para ver o que foi alterado usamos o **`git diff`**

Comando

```
$ git diff [filename]
```

Ver o que foi alterado

O comando **`git diff`** mostra as mudanças de arquivos **NÃO marcados** para *commit*, ou seja, que NÃO estão no *staging*

```
$ git diff
diff --git a/README b/README
index 8ebb991..643e24f 100644
--- a/README
+++ b/README
@@ -31,7 +31,8 @@

  -   Thiago Bitencourt
  +   Thiago R. M. Bitencourt
```

Ver o que foi alterado e está no *stage*

Para ver as alterações de arquivos que **já foram marcados** para *commit*, ou seja, que já estão no *staging area*

```
$ git diff --staged [filename]
```

Commitando as alterações

Para criar uma nova versão dos arquivos alterado, executamos o comando

```
$ git commit

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is up-to-date with 'origin/master'.
#
# Changes to be committed:
#    new file:   CONTRIBUTING.md
#    modified:   README
#
~
~
~
".git/COMMIT_EDITMSG" 9L, 283C
```

Commitando as alterações

A mensagem exibida já apresenta o estado do repositório, mesma informação gerada pelo comando **`git status`**

Adicione uma mensagem que descreva as alterações que foram feitas

* É possível alterar o editor padrão através do comando

```
$ git config --global core.editor [name]
```

Padrão de mensagem de *commit*

A mensagem deve responder a pergunta, **o que faz este *commit*?**

```
Este commit… [ Sua mensagem de commit ]
```

A primeira linha da mensagem deve ser bem sucinta

Se for necessário mais explicações sobre o commit, deixe uma linha em branco e então descreva o commit

Exemplo de mensagem de commit

Este *commit*...

```
Adiciona um novo arquivo ao repositório


O arquivo CONTRIBUTING.md descreve como proceder para contribuir para o meu projeto.

Adicionando este arquivo eu concluo a task/tarefa 10

close #10 #time 30m
```

Commit com mensagem *inline*

Se a mensagem será curta, com apenas uma linha, podemos passá-la diretamente ao comando **`git commit`** com o parâmetro **`-m`**

Comando

```
$ git commit -m "Cria um commit com uma mensagem inline"
```

Stage and commit

Ao invés de executar o comando **`git add [arquivo]`** sempre que alteramos um arquivo, podemos usar parâmetro **`-a`** no comando de *commit*

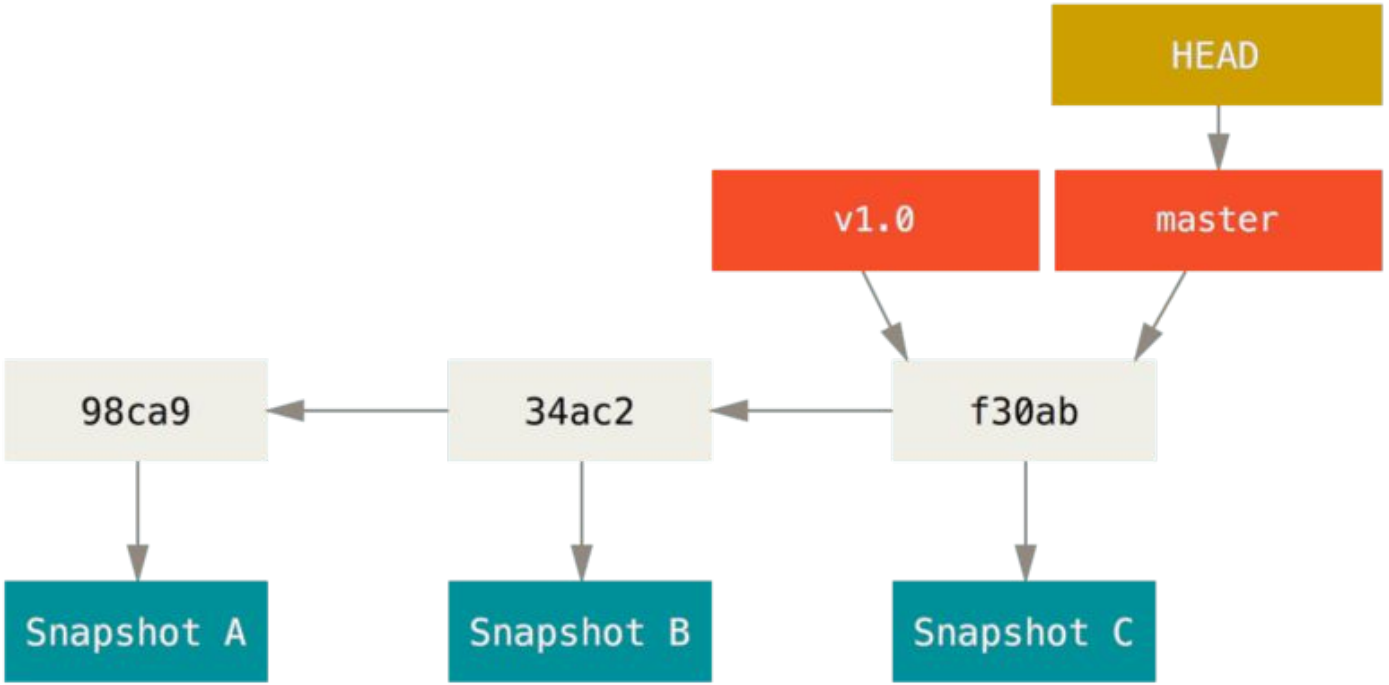
Comando

```
$ git commit -a -m "Faz o stage do arquivo antes do commit"
```

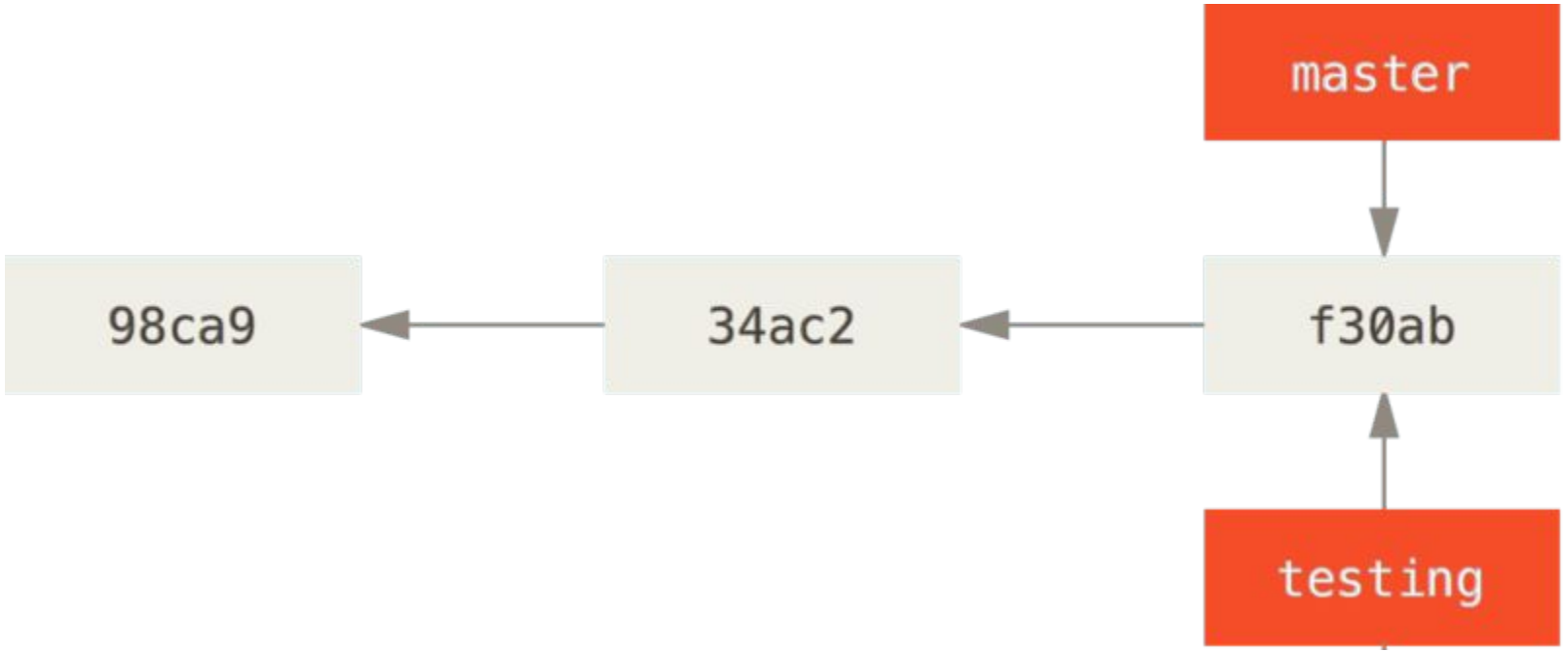
* Funciona apenas para arquivos que já são rastreados pelo Git. Quando adicionamos um arquivo novo é necessário executar o comando **`git add [arquivo]`**

```
$ git commit -am "Faz a mesma coisa que -a -m"
```

Branches

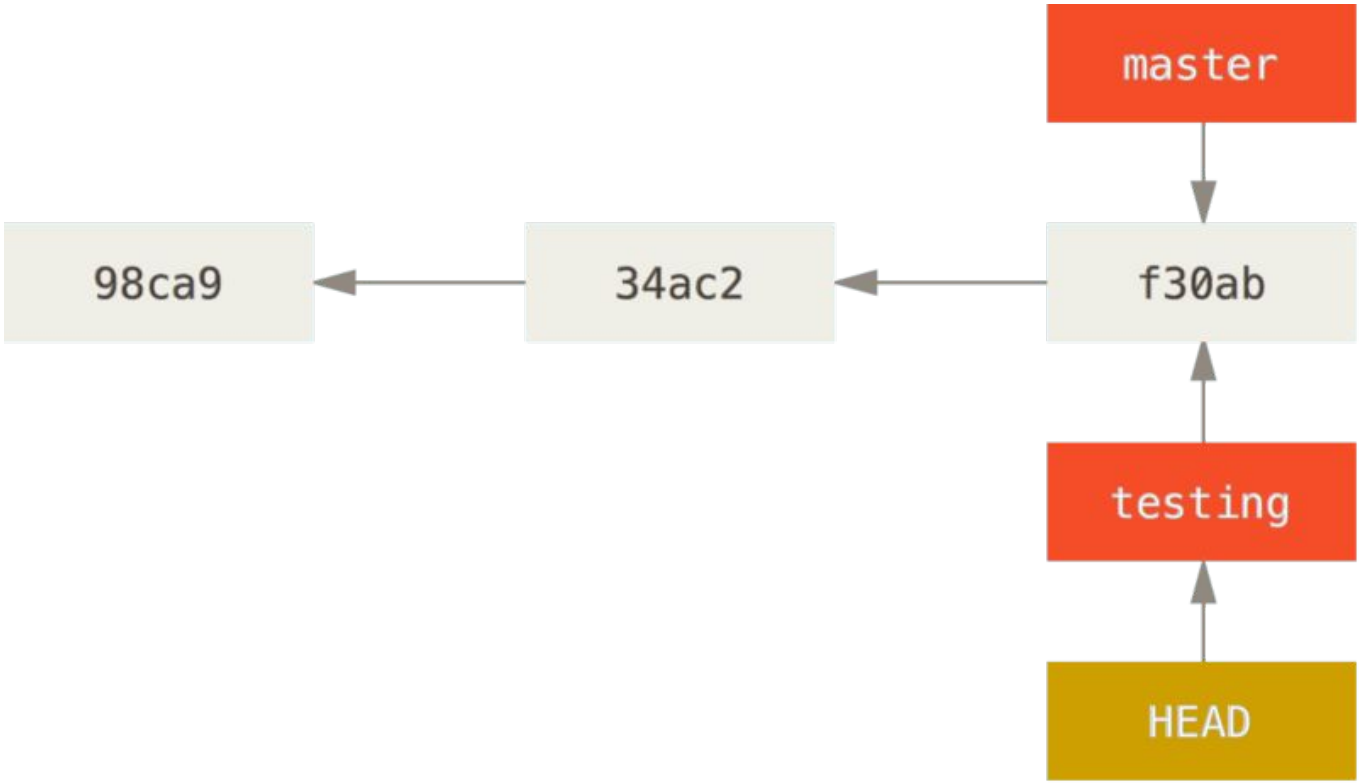

HEAD
v1.0
master
98ca9
34ac2
f30ab
Snapshot A
Snapshot B
Snapshot C

Branches

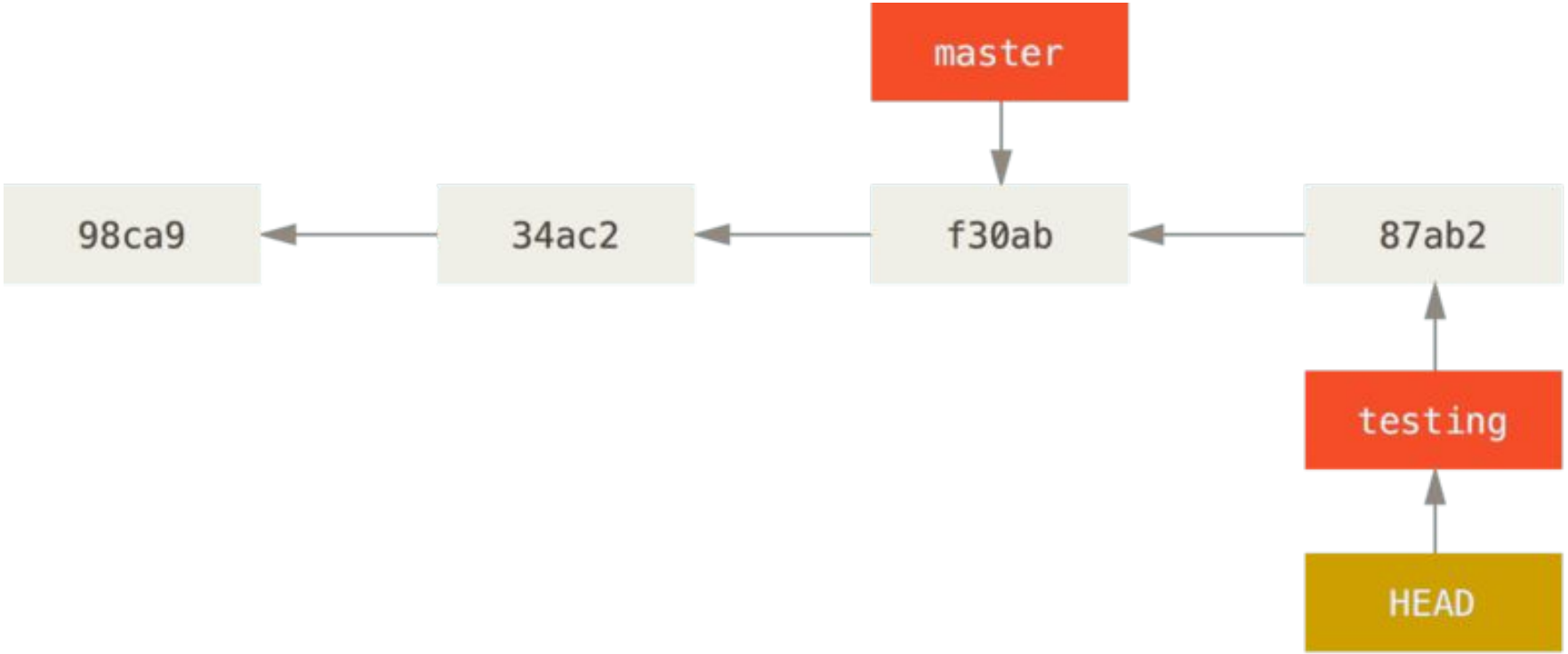


Branches

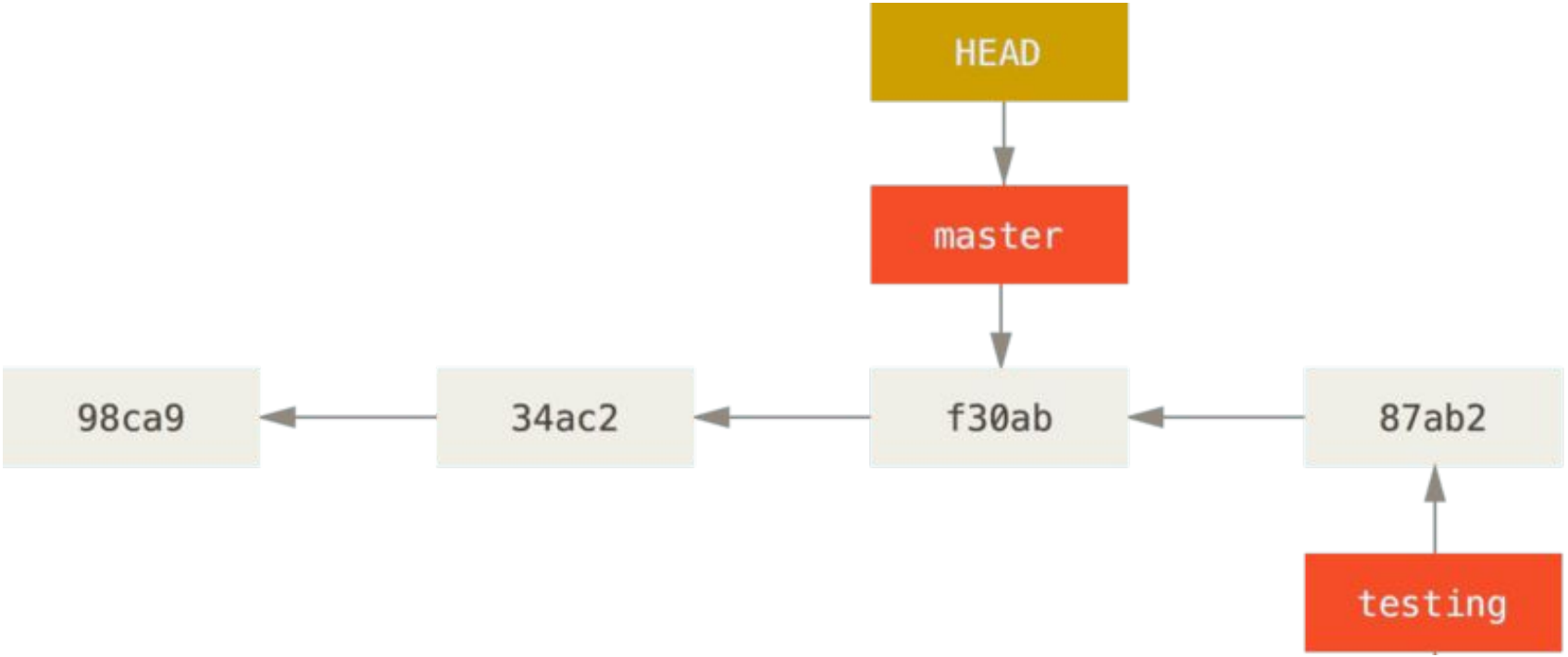
Branches



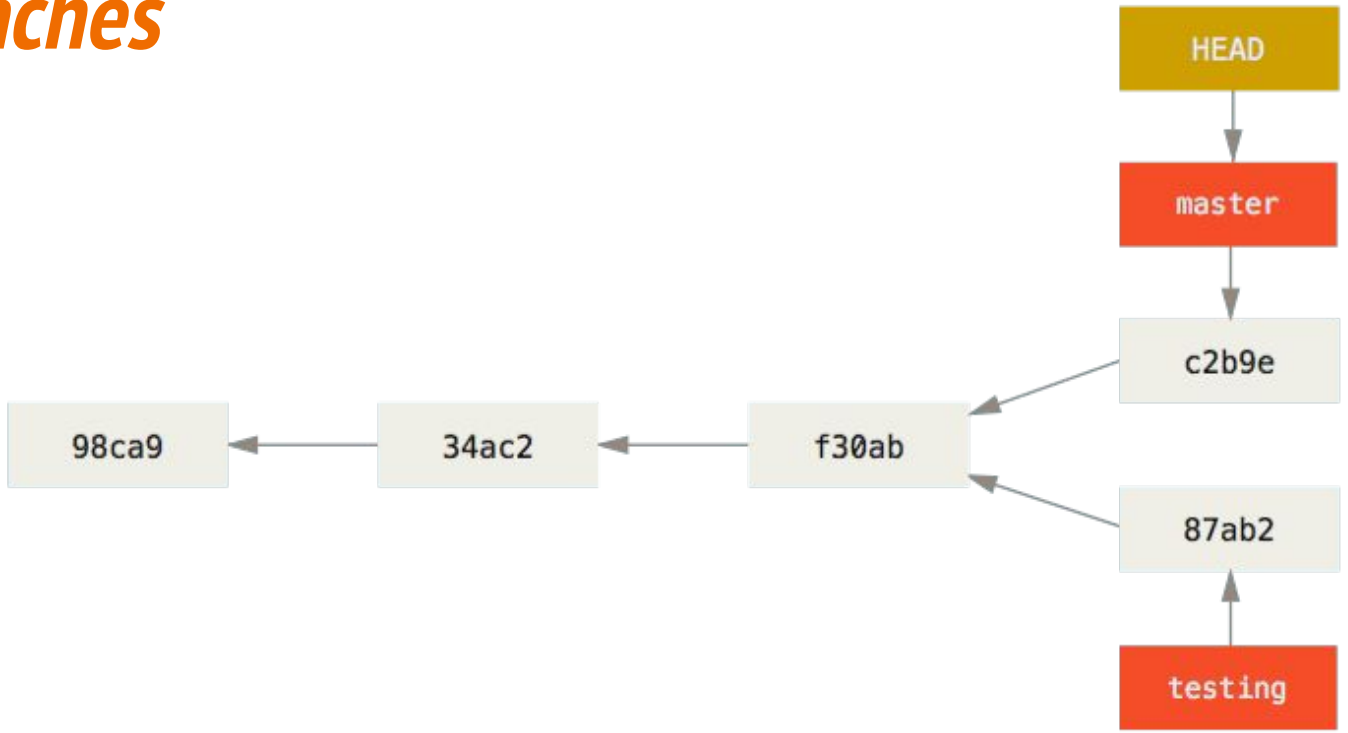
Branches



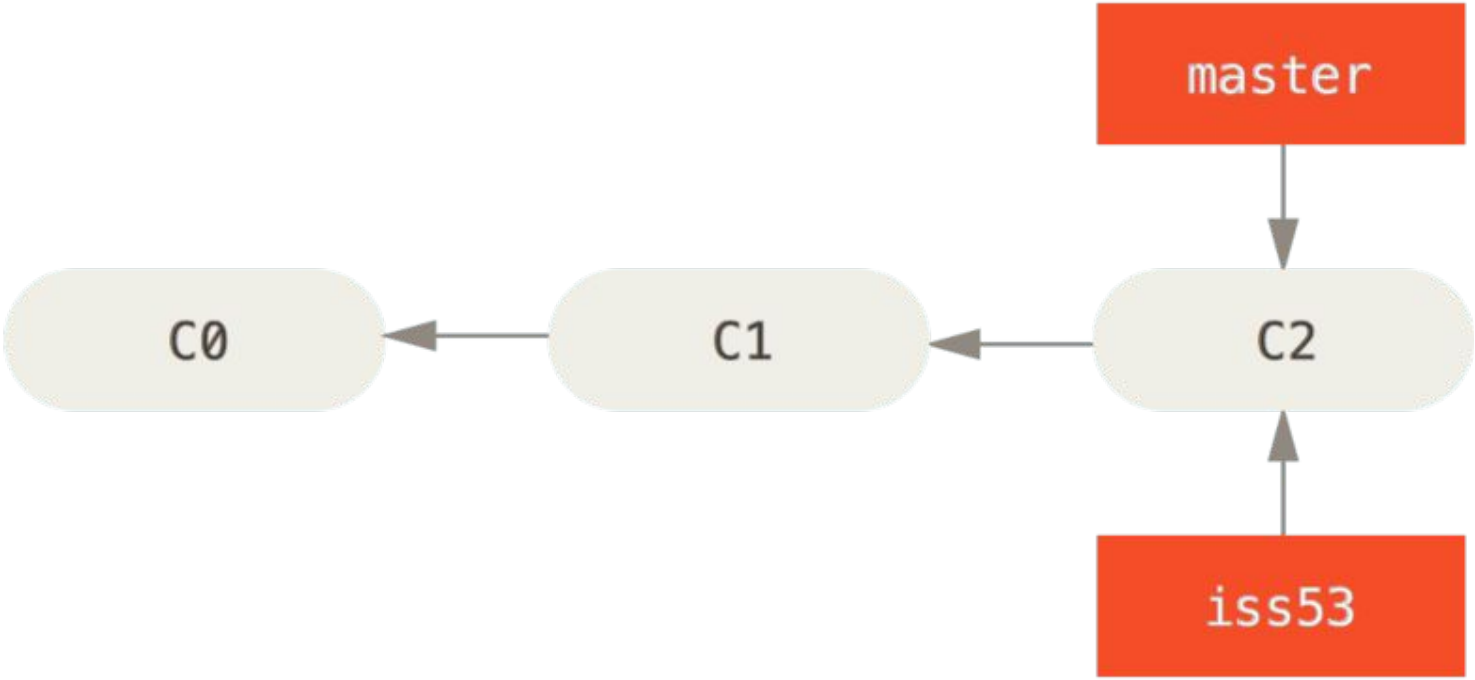
Branches



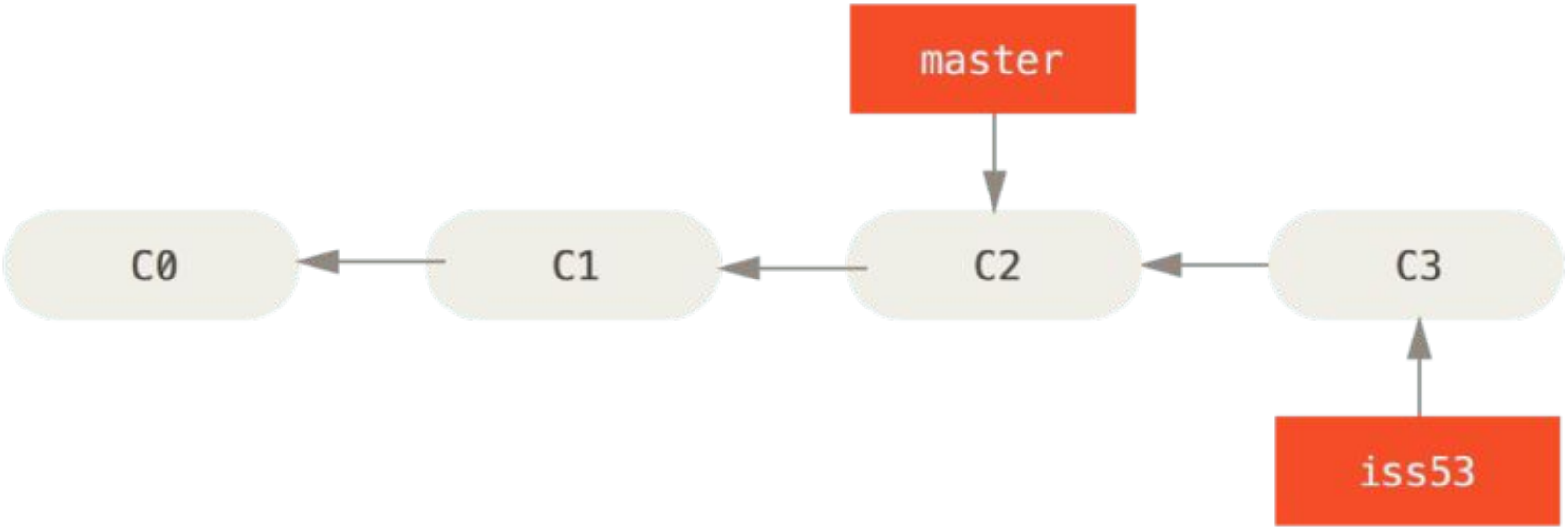
Branches



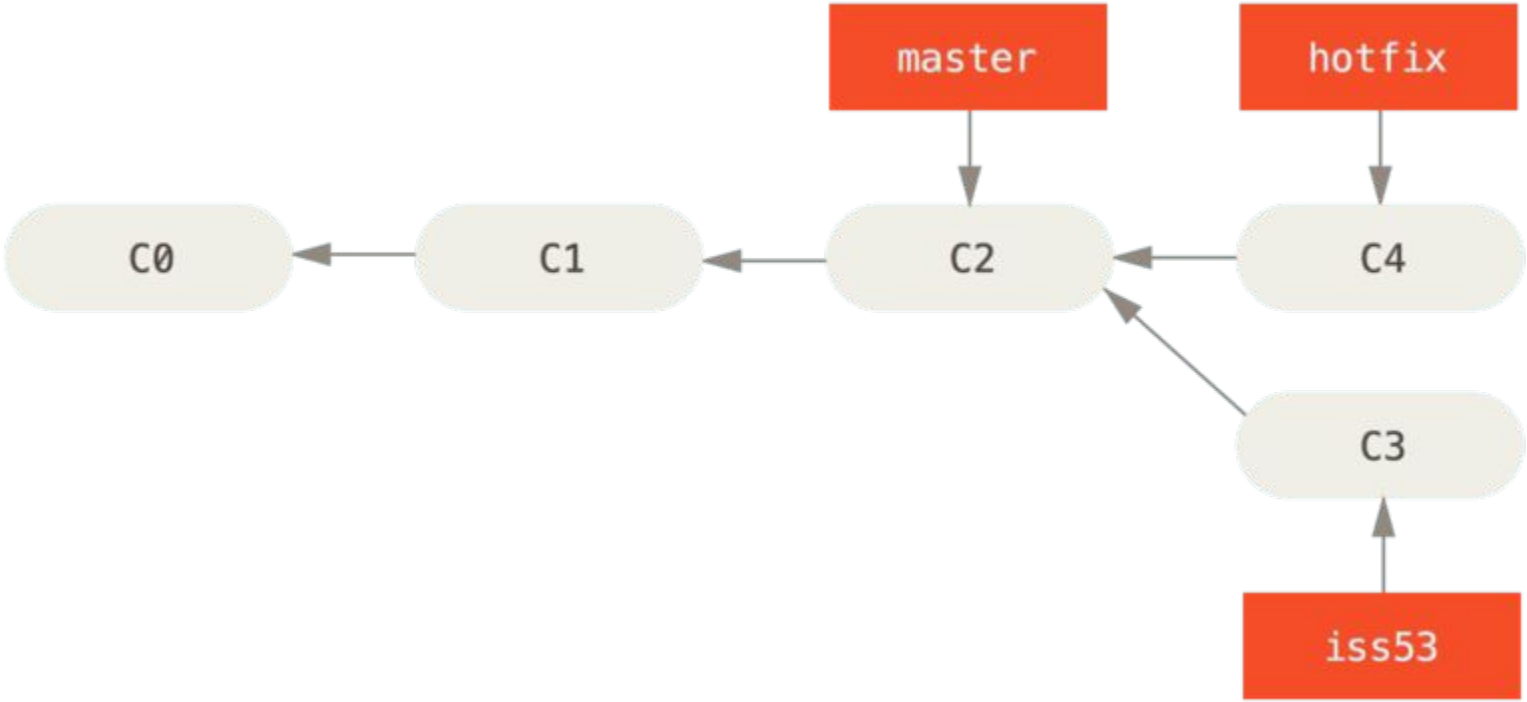
Merge



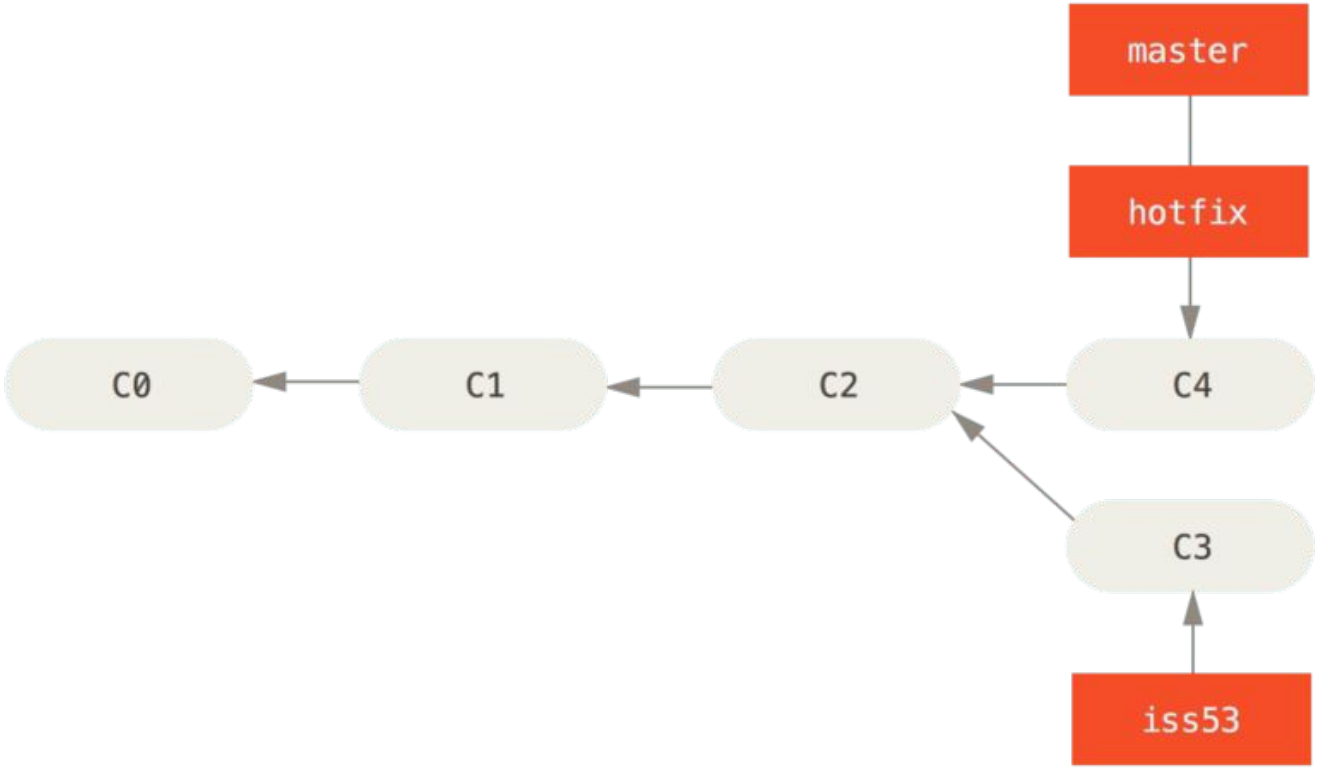
Merge



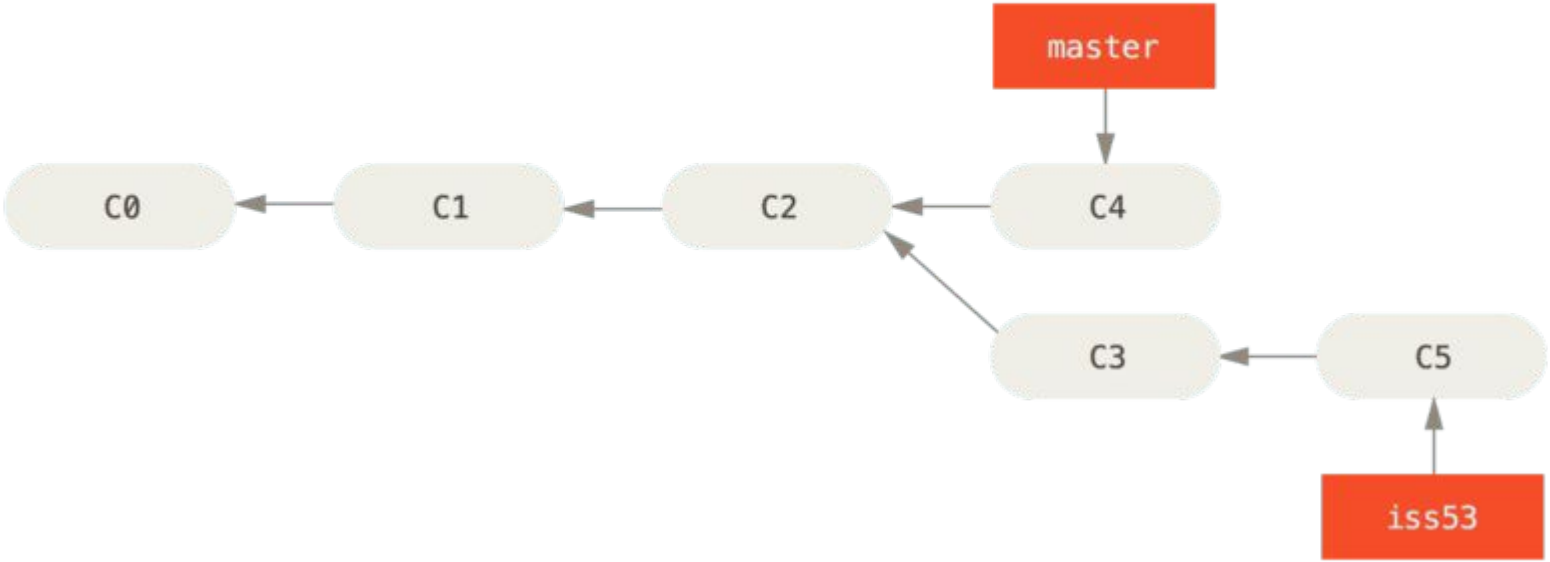
Merge



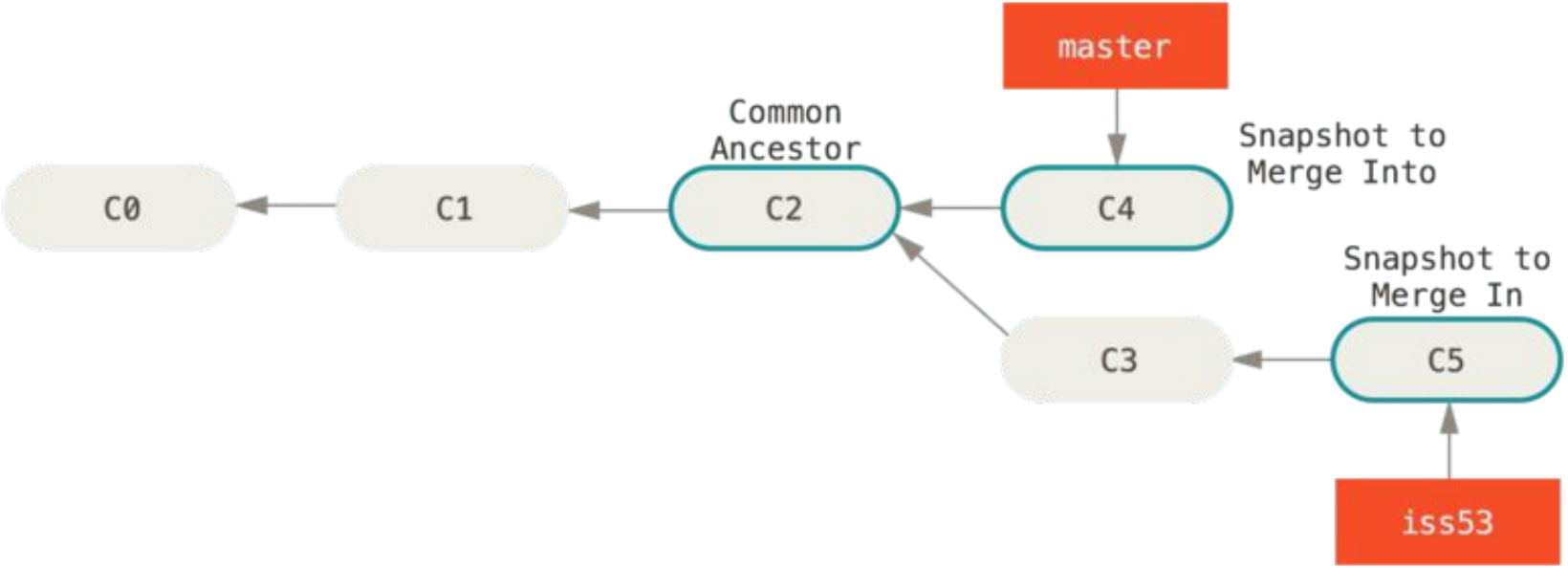
Merge Fast-foward



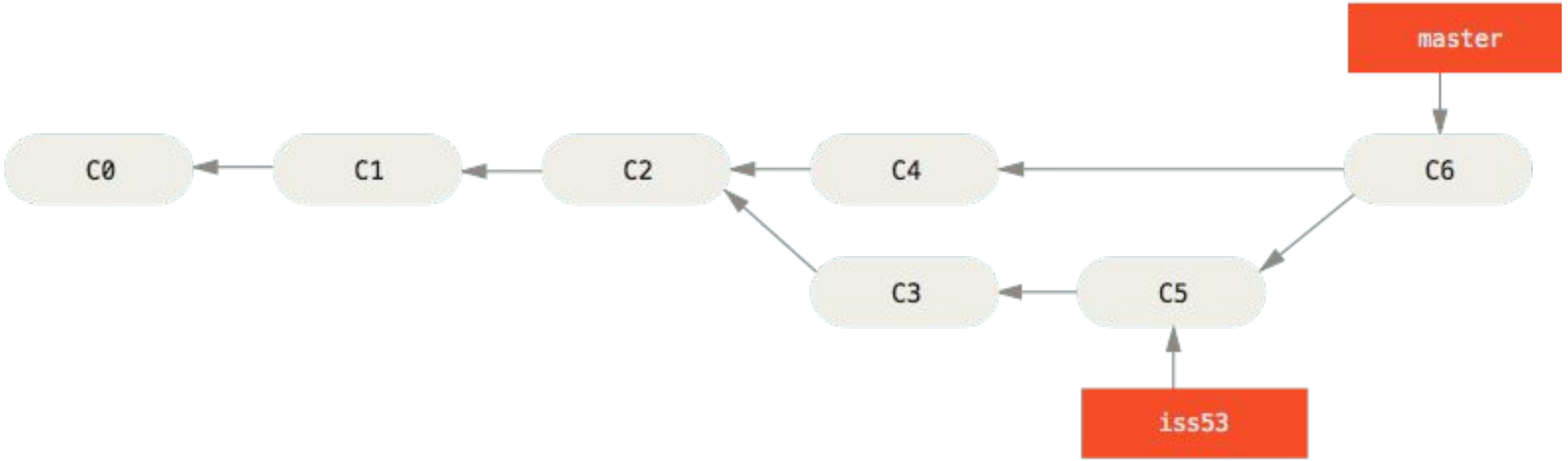
Merge



Merge Recursive strategy



Merge



Conflitos

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Conflitos

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Conflitos

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

```
<div id="footer">
please contact us at email.support@github.com
</div>
```

Referências

1. https://git-scm.com/