Universidade Federal do Acre
Programa de Pós-Graduação em Ciência da Computação

Curso de Git

Prof. Manoel Limeira de Lima Júnior

Agenda

- Introdução
- Comandos Básicos
- Repositórios Remotos
- *Branches*
- *Workflows*
- GitHub
- Ferramentas

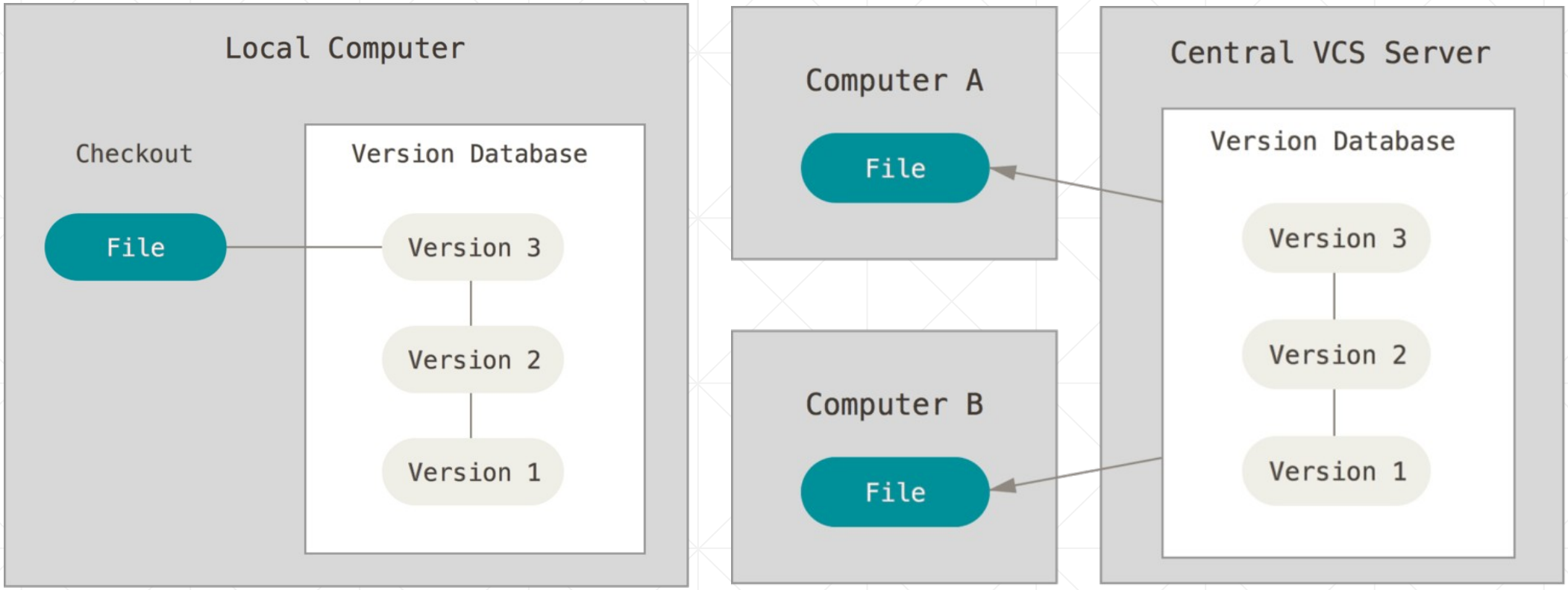
Introdução – Conceito e Motivação

- Sistemas de 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.”

- O que pode ser versionado?
- Praticamente tudo, exemplos: livros, imagens, código fonte, documentos (dissertação).

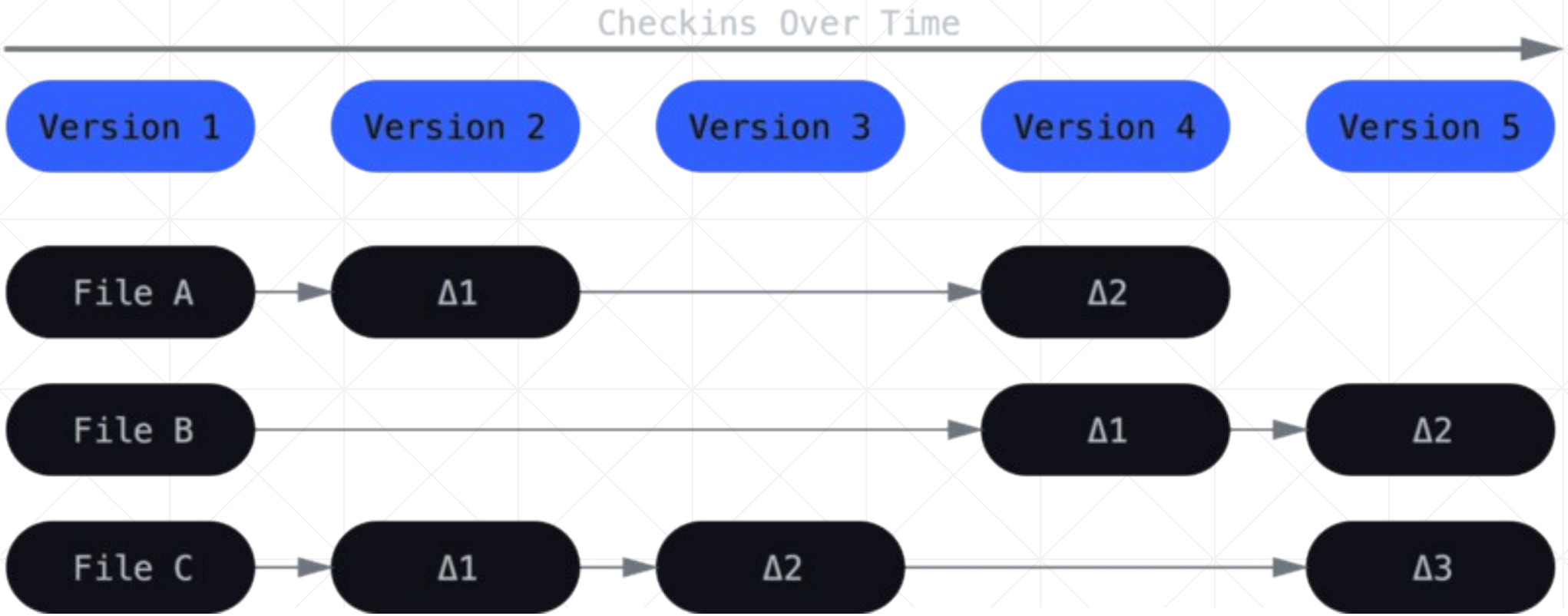
Introdução - Conceito

- Sistemas de Controle de Versão Centralizados



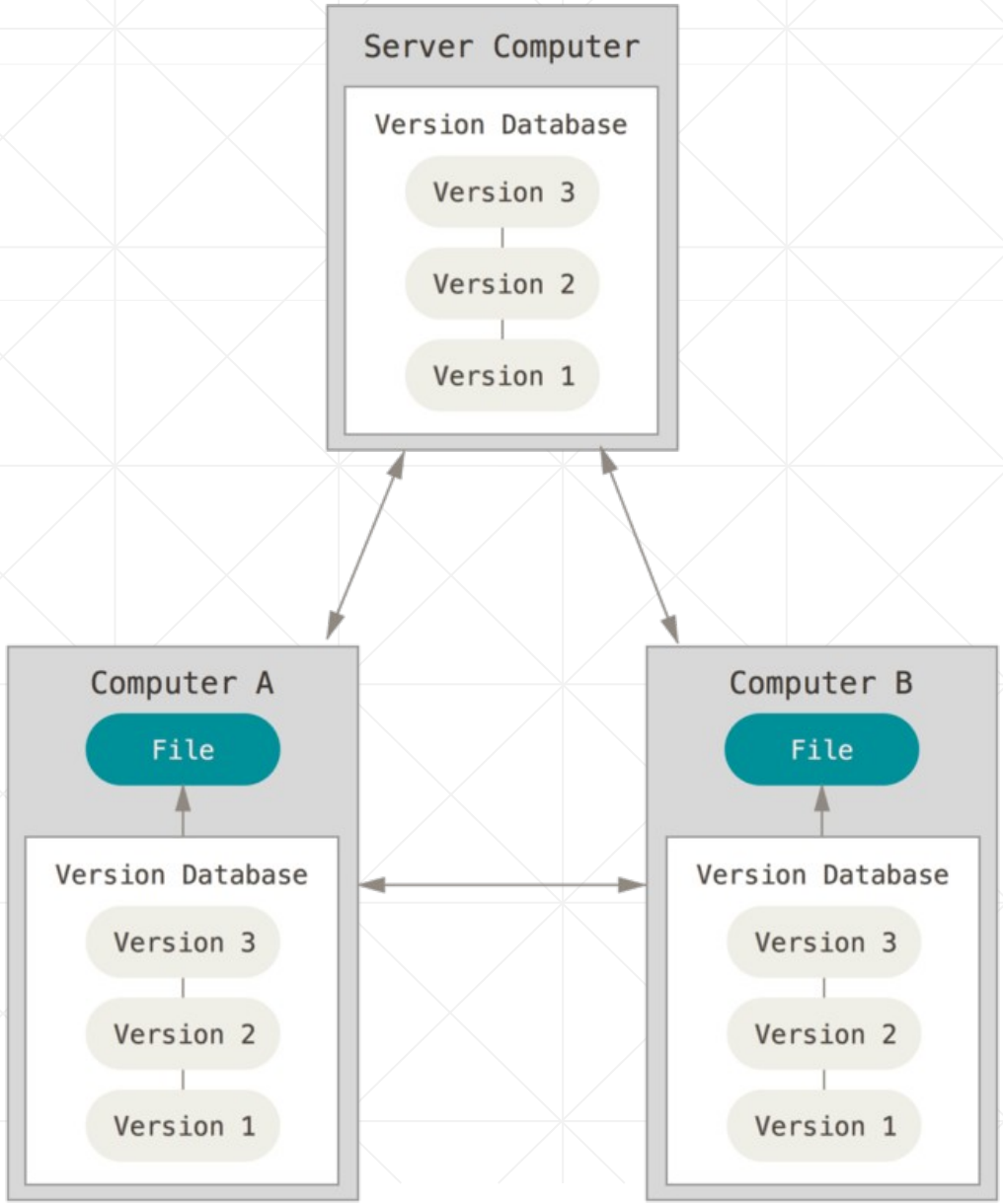
Introdução - Conceito

- Sistemas de Controle de Versão Centralizados



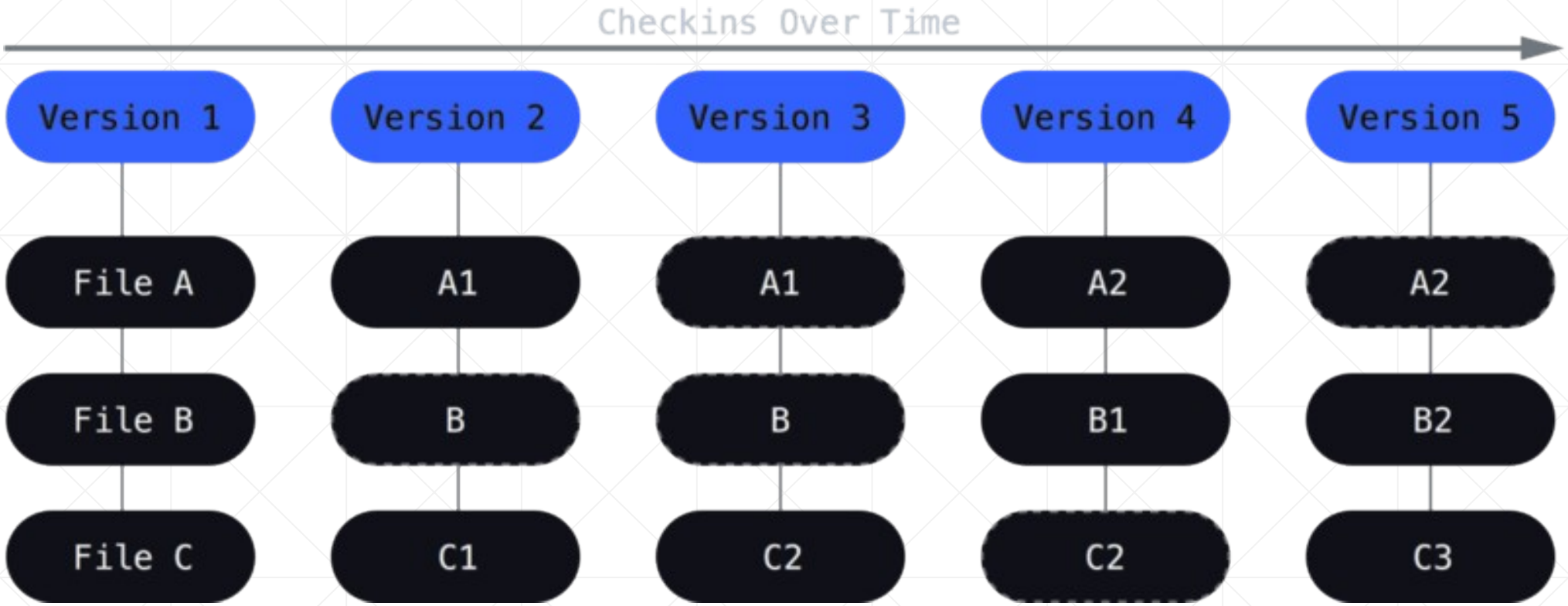
Introdução - Conceito

- Sistemas de Controle de Versão Distribuídos



Introdução - Conceito

- Sistemas de Controle de Versão Distribuídos

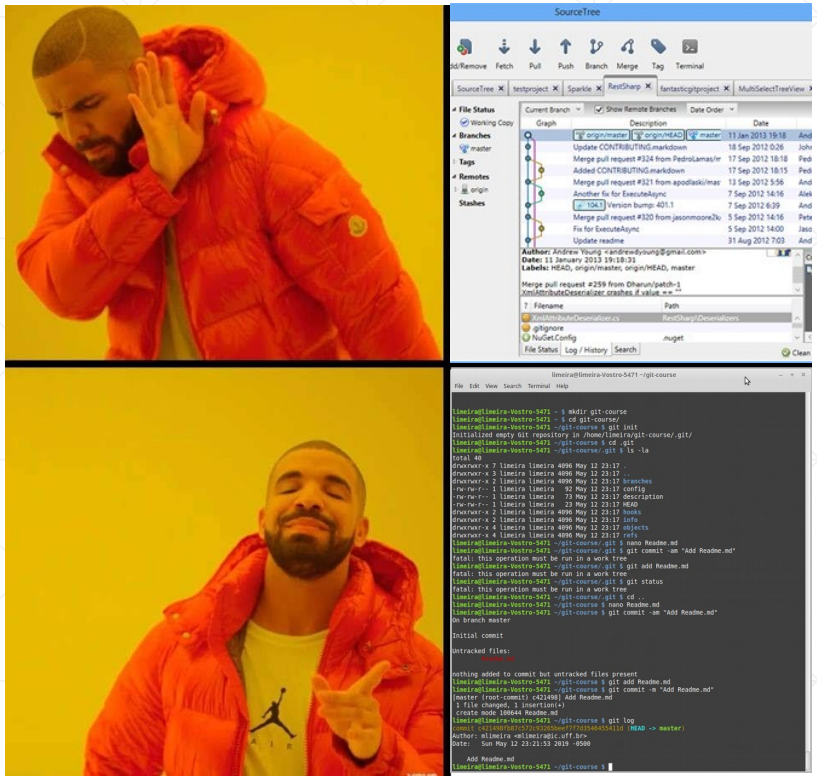


Introdução - História

- Em 2002, o projeto do núcleo do Linux começou usar uma ferramenta proprietária chamada BitKeeper.
- Em 2005, Linus Torvalds começou a desenvolver a sua própria ferramente com alguns objetivos em mente:
  - Velocidade
  - Projeto simples
  - Suporte para desenvolvimento não-linear (*branches*)
  - Completamente distribuído
  - Capaz de lidar com projetos grandes

Comandos Básicos – Git

- Download Git: https://git-scm.com/downloads
  - Mac
  - Linux
  - Windows
- Linha de Comando



Comandos Básicos – Git

- Configuração inicial

```
$ git config --global user.name “Manoel Limeira”
$ git config --global user.email “mlimeira@ic.uff.br”
$ git config --global core.editor nano
```

- Verificar

```
$ git config user.name
$ git config --list
```

- Ajuda

```
$ git help config
```

Comandos Básicos – Git

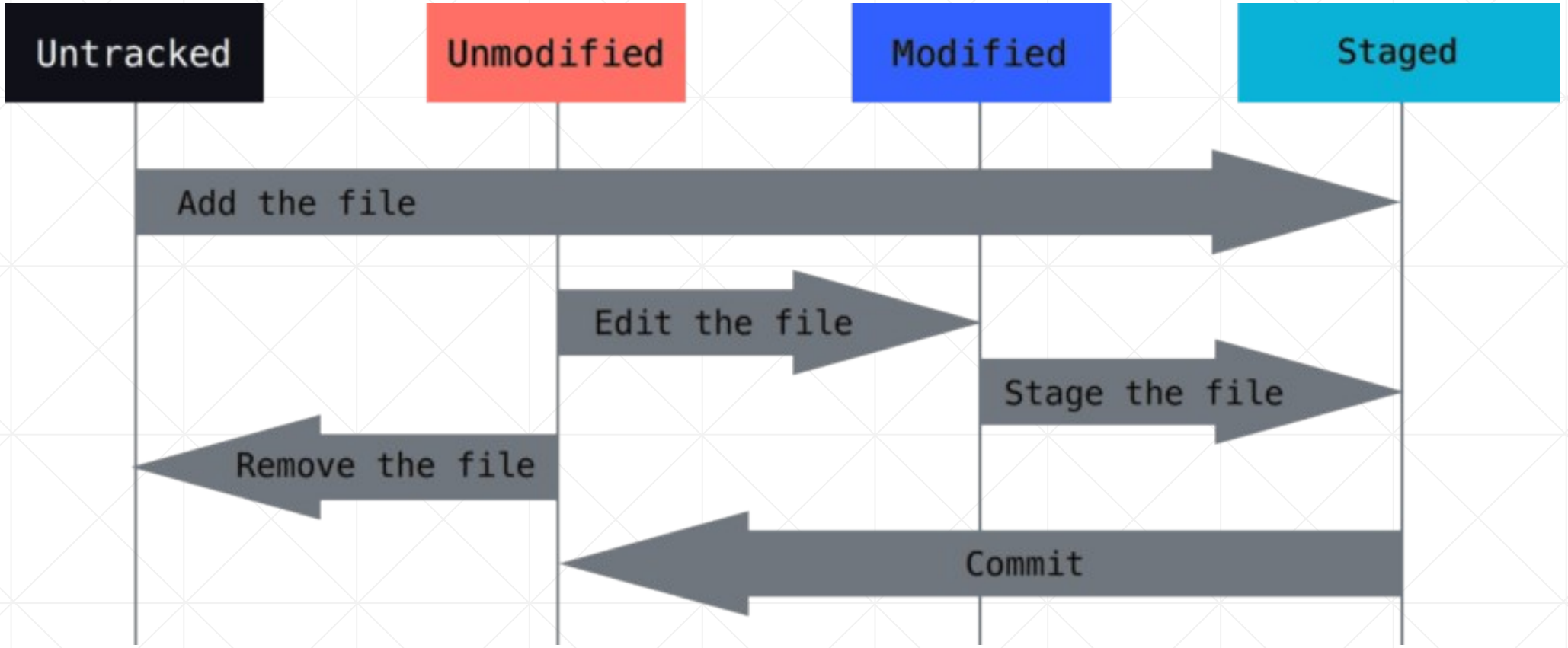
- Inicialização de um repositório

```
$ mkdir curso-git //cria um diretório
$ cd curso-git //ativa um diretório
$ git init //inicializa o versionamento do diretório
$ ls -la //lista o conteúdo do diretório
```

- Informações sobre o repositório (diretório **.git**)

```
$ cd .git
$ ls
branches  config  description  HEAD  hooks  info  objects  refs
```

Ciclo de Vida dos Arquivos – Git



Comandos Básicos – Git

- Criar e alterar um arquivo para ser versionado

```
$ cd curso-git //diretório controlado pelo git
$ nano Readme.md //pode usar qualquer editor
Ctrl + o //escreve no arquivo
Ctrl + x //encerra o editor
```

- Ocultar arquivos do versionamento

```
$ cd .git/info
$ nano exclude //arquivo (file.cpp) ou regra (*.class)
```

Comandos Básicos – Git

- Verificação do *status* de um repositório

```
$ git status
```

- Adicionar uma versão de um arquivo para a área transferência

```
$ git add Readme.md
```

- Confirmar uma versão de um arquivo no repositório

```
$ git commit -m “Add Readme.md”
$ git status
```

Comandos Básicos – Git

- Verificar *log* do repositório

```
$ git log [--decorate,--graph,--stat,author=“Manoel”]
$ git log --pretty=oneline
$ git shortlog
```

- Mostrar um *commit* do repositório

```
$ git show 0856ee55ba88084595e54dfb5857a55c3407af3a
```

- Verificar diferenças no repositório

```
$ git diff
```

Comandos Básicos – Git

- Desfazer alterações no repositório (modo de edição)

```
$ git checkout Readme.md
$ git diff
$ git status
```

- Desfazer alterações (modo de transferência)

```
$ git reset HEAD Readme.md
```

- Desfazer alterações (modo de versões)

```
$ git reset [--soft,--mixed,--hard] hash
```

Repositórios Remotos – GitHub

- Criar ou acessar uma conta (`github.com`)
- Criar um repositório (**`user`**/**`repository`**)
- Enviar as versões para o repositório remoto

```
$ git remote add origin https://github.com:user/repository.git
$ git push -u origin master
$ Username for 'https://github.com': user
$ Password for 'https://user@domain@github.com': password
```

Repositórios Remotos – GitHub

- Criar ou acessar uma conta (`github.com`)
- Criar um repositório (**`user`**/**`repository`**)
- Criar uma chave de acesso (https://help.github.com/en/articles/about-ssh)

```
$ cd ~/.ssh (no linux)
$ ssh-keygen -t rsa -b 4096 -C "user@domain.com"
```

- Adicionar chave nas configurações do GitHub
- Enviar as versões para o repositório remoto

```
$ git remote add origin git@github.com:user/repository.git
$ git push -u origin master
```

Repositórios Remotos – GitHub

- Mostrar repositório remoto

```
$ git remote show origin
```

- Renomear repositório remoto

```
$ git remote rename origin origin-new
```

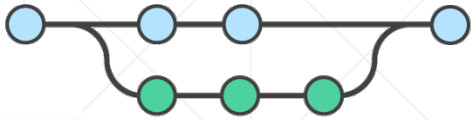
- Remover repositórios remotos

```
$ git remote remove origin-new
```

- Clonar repositórios remotos

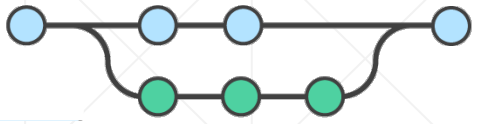
```
$ git clone git@github.com:user/repository repository-clone
$ cd repository-clone
$ cat Readme.md
```

Branches

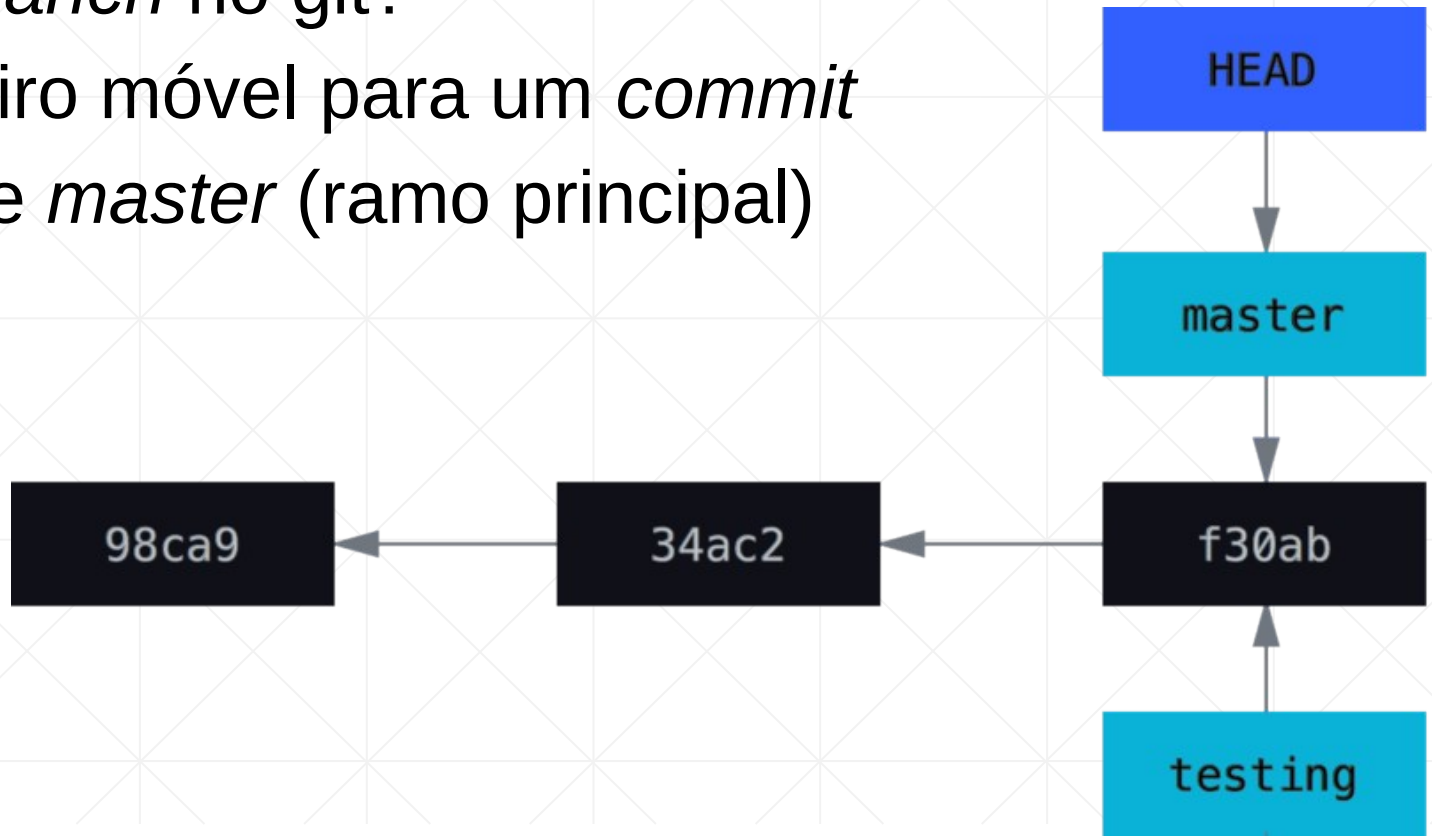


- Por que usar *branches* no git?
  - É leve e facilmente desligável
  - Modificar os arquivos sem alterar o *branch master*
  - Permite o trabalho de múltiplas pessoas no projeto
  - Evita alguns conflitos
  - Fornece ferramentas para gerenciar outros conflitos

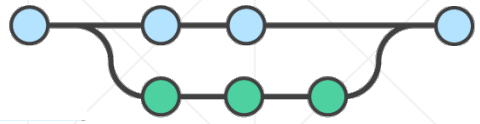
Branches



- O que é um *branch* no git?
  - É um ponteiro móvel para um *commit*
  - Ex: *testing* e *master* (ramo principal)



Branches



- Criar *branches*

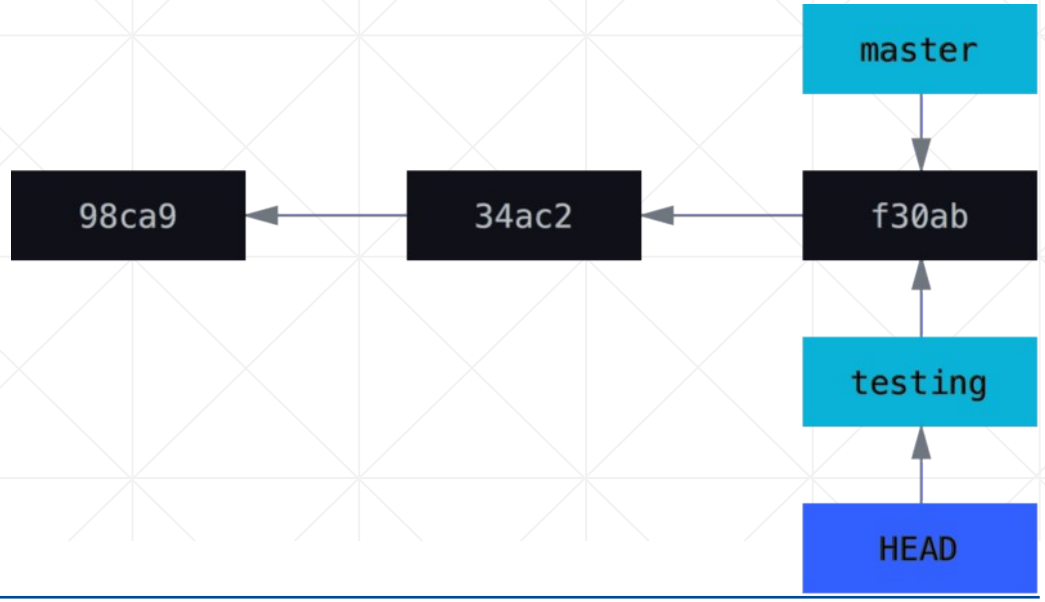
```
$ git branch v1 ou git checkout -b testing
```

- Listar *branches*

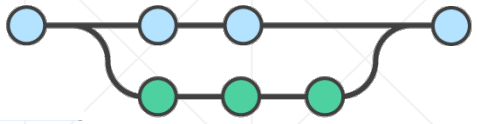
```
$ git branch
```

- Alternar entre os *branches*

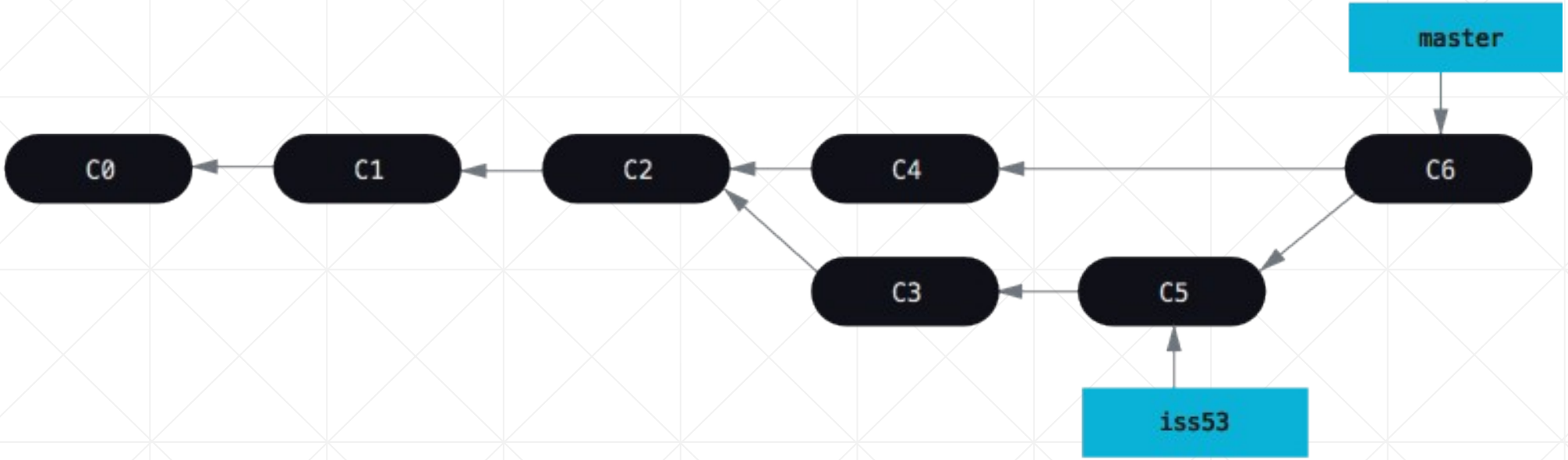
```
$ git checkout testing
```



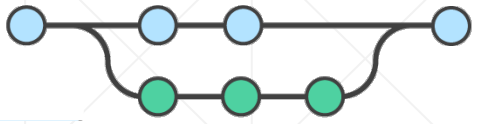
Branches – merge



- Exemplo: Criar um *branch* para resolver um problema no software, retornar para o *master* e continuar o trabalho



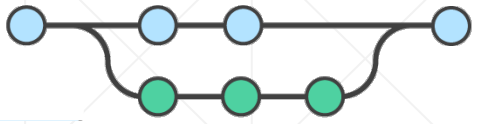
Branches – merge



- Código para o cenário **sem** conflito (*fast-forward*)

```
$ git checkout -b issue71
$ nano arq1.txt //editar o conteúdo de arq1.txt
$ git add . //adicionar arquivos para área de transferência
$ git commit -m “add arq1”
$ git checkout master //alterar outros arquivos no master
$ git merge issue71
$ git log --graph
$ git branch -d issue71 //passo opcional
```

Branches – merge

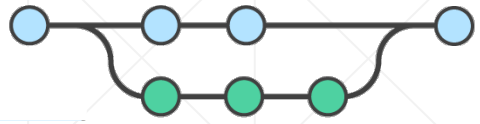


- Código para o cenário **com** conflito (recursive-*strategy*)

```
$ git checkout -b issue72
$ nano arq1.txt //editar o conteúdo de arq1.txt
$ nano arq2.txt //editar o conteúdo de arq2.txt
$ git add . //adicionar arquivos para área de transferência
$ git commit -m “add arq1”
$ git checkout master //alterar arq1.txt no master
$ git merge issue72 //exibe o conflito que deve ser resolvido
$ nano arq1.txt //resolver o conflito
$ git commit -am “merge issue72”
```

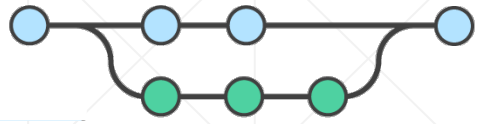
```
$ git log --graph --oneline
```

Branches – Exercício



- O objetivo é colaborar no desenvolvimento de um aquivo
- O arquivo será o readme.md do repositório mlimeira/curso-git
- O arquivo deve conter o conteúdo visto nas aulas de git
- Cada aluno deve:
  - Criar um *branch* com o seu nome
  - Escolher um tema/assunto/comando
  - Realizar as alterações localmente
  - Enviar as versões para o repositório remoto

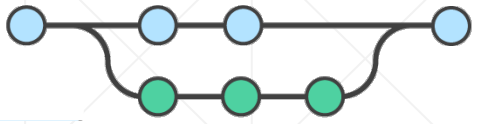
Branches



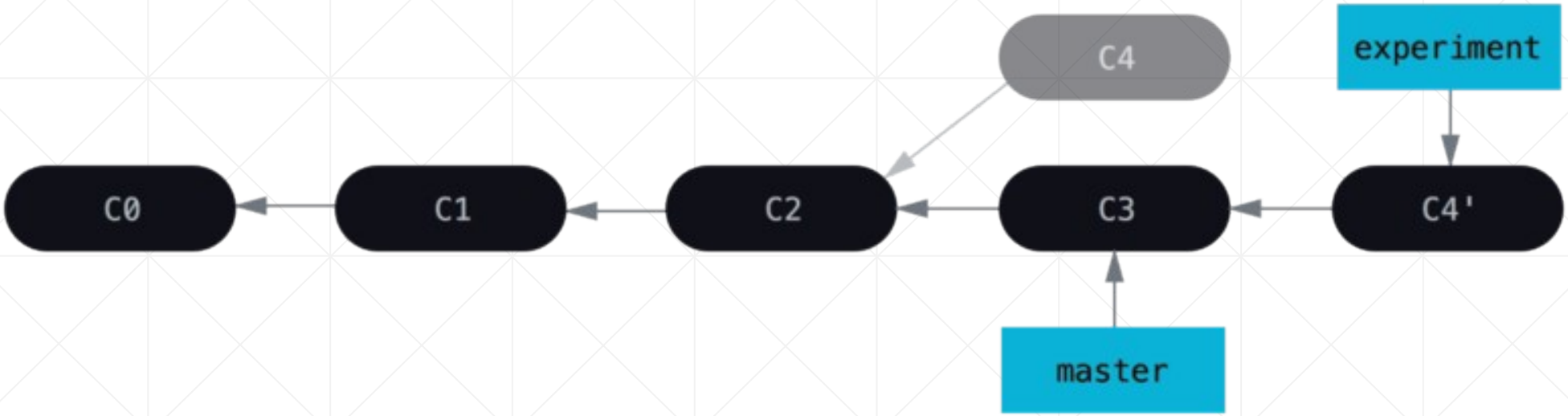
- Opções para gerenciar os *branches*

```
$ git branch -v //listar branches com hash e mensagem
$ git branch --merged //listar branches que você fez merge
$ git branch --no-merged //listar branches que você não fez merge
$ git branch -d branch-name //excluir branch que você já fez merge
$ git branch -D branch-name //excluir branch que você não fez merge
$ git fetch origin //atualizar as referências locais
$ git branch -vv //listar branches mostrando as referências
$ git pull //atualizar o conteúdo do repositório local
```

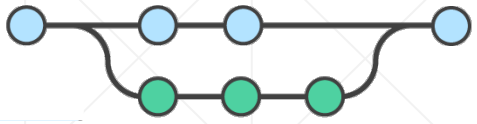
Branches – rebase



- Exemplo: Criar um *branch* para resolver um problema no software, retornar para o *master* e continuar o trabalho

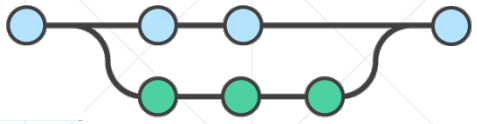


Branches – rebase

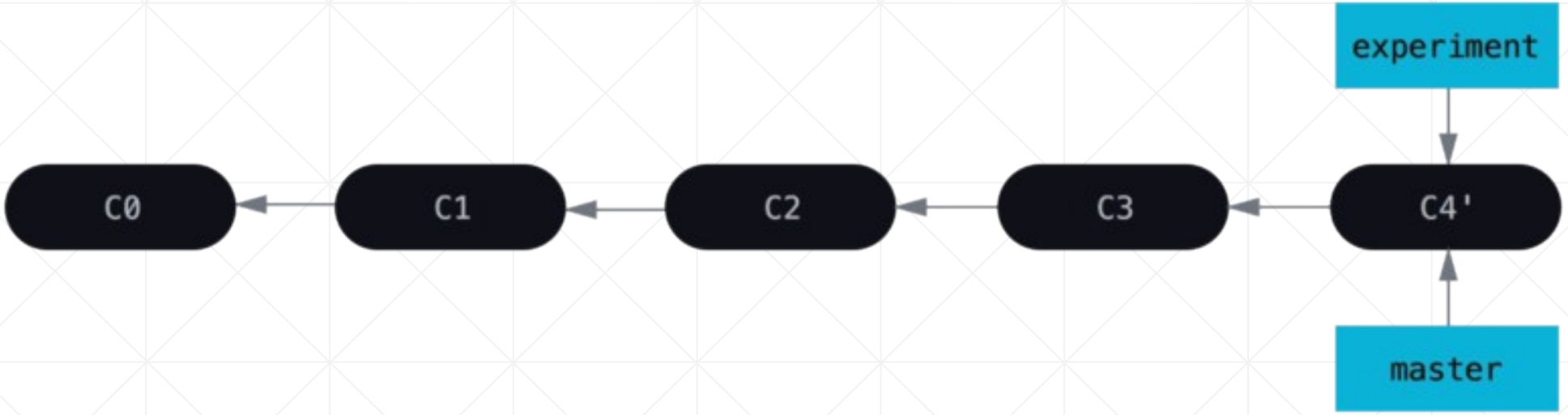


```
$ git checkout -b experiment
$ nano exp.txt //editar o conteúdo de exp.txt
$ git add . //adicionar arquivos para área de transferência
$ git commit -m “add exp.txt”
$ git checkout master //alterar outros arquivos no master
$ git checkout experiment
$ git rebase master
$ git checkout master
$ git merge experiment
$ git log --graph --oneline
```

Branches – rebase

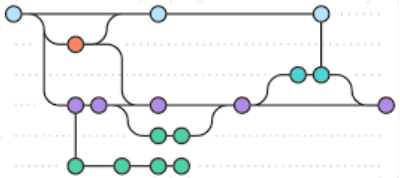


- Status do repositório após *rebase*



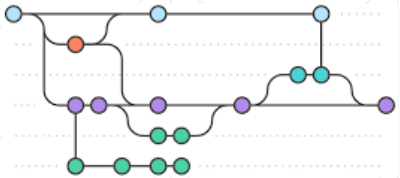
- Não faça *rebase* de *commits* que você enviou para um repositório público.

Workflows



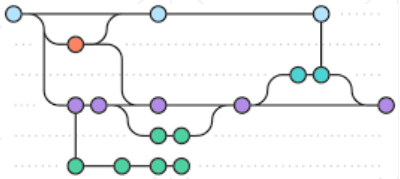
- ***Workflow*** (fluxo de trabalho) é uma sequência de passos necessários para automatizar processos, de acordo com um conjunto de regras definidas, permitindo que estes possam ser transmitidos de uma pessoa para outra.
- Deve levar em consideração o tamanho de equipe, a facilidade de utilização e adequação ao contexto da equipe.
- Exemplos:
  - *Centralized*
  - *Feature Branch*
  - *Forking*
  - *GitFlow*

Workflows – Centralized

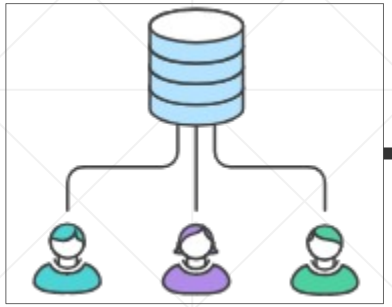


- *Branch* único (*master*)
- Familiaridade com SVN
- Atualização e conflitos constantes
- Funciona bem com equipes bem pequenas
- Projetos pessoais

Workflows – Centralized

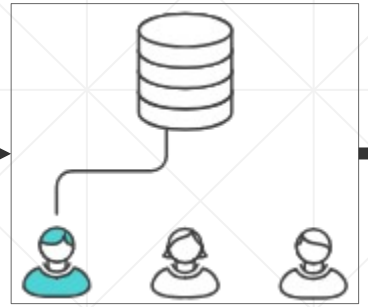


Equipe pequena



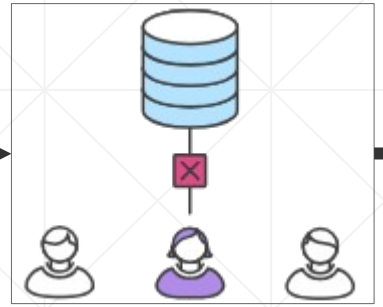
Desenvolvedor A

envia uma *feature*

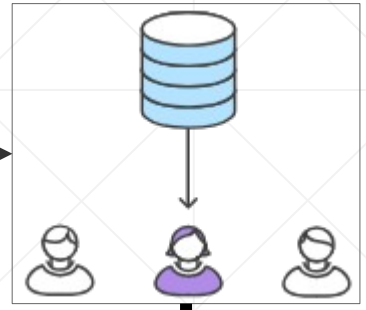


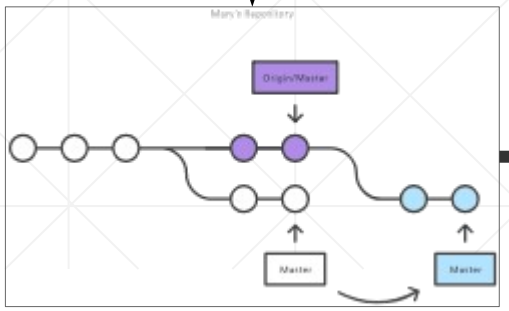
Desenvolvedor B

envia outra *feature*



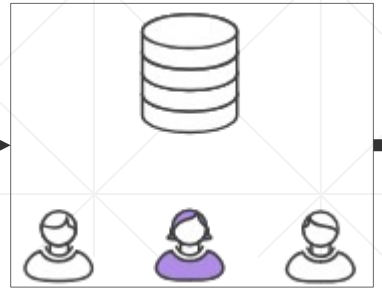
- $ git pull --rebase origin master





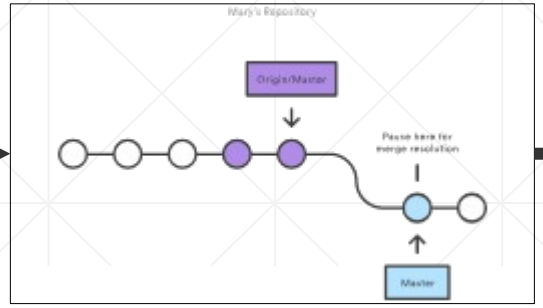
Desenvolvedor B

resolve conflitos

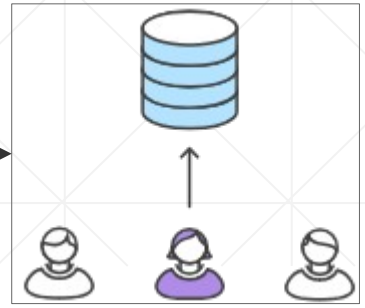


$ git add [filename]

$ git rebase --continue

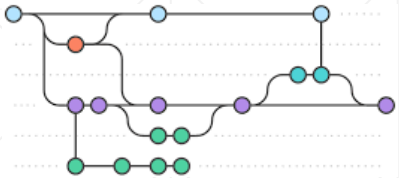


- $ git push origin master



$ git rebase --abort

Workflows – Centralized



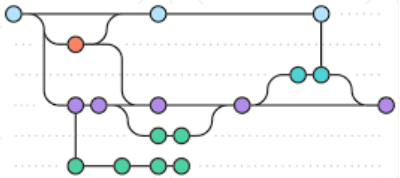
User 1

```
$ git checkout master
$ nano file.txt //editar o conteúdo de file.txt
$ git commit -am “update file.txt”
$ git push origin master
$ git log --oneline
```

User 2

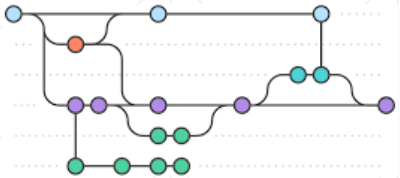
```
$ git checkout master
$ nano file.txt //editar o conteúdo de file.txt
$ git commit -am “new update file.txt”
$ git push origin master //rejected
$ git pull origin master --rebase //Resolver o conflito em file.txt
$ git add .
$ git rebase --continue
$ git push origin master
```

Workflows – Feature Branch



- Vários *branches*
- Um *branch* para cada *feature*
- Evita alguns conflitos
- Permite trabalho paralelo
- Funciona muito bem com equipes maiores
- Promove a colaboração com membros da equipe por meio de *Pull Requests* e revisões de *merge*

Workflows – Feature Branch



Novo *branch* para *feature*

Commit e almoçar

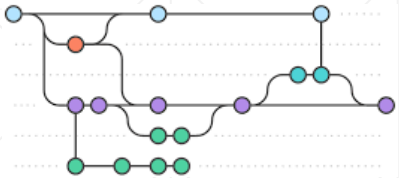
Push para *origin*

Pull request

Novos *commits*

Merge Pull request

Workflows – Feature Branch



User 1

```
$ git checkout -b issue7
$ nano feature.java //editar o conteúdo de feature.java
$ git add .
$ git commit -m “issue #7”
$ git push origin issue7
Criar um Pull Request no GitHub
```

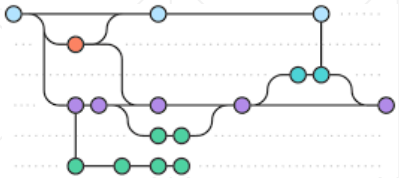
User 2

```
Visualiza o Pull Request no GitHub
Realiza o merge do Pull Request no GitHub
Opcionamente exclui o branch do repositório
```

User 1

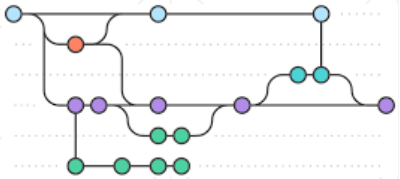
```
$ git checkout master
$ git pull origin master //Atualizar o repositório local
$ git log --graph --oneline
```

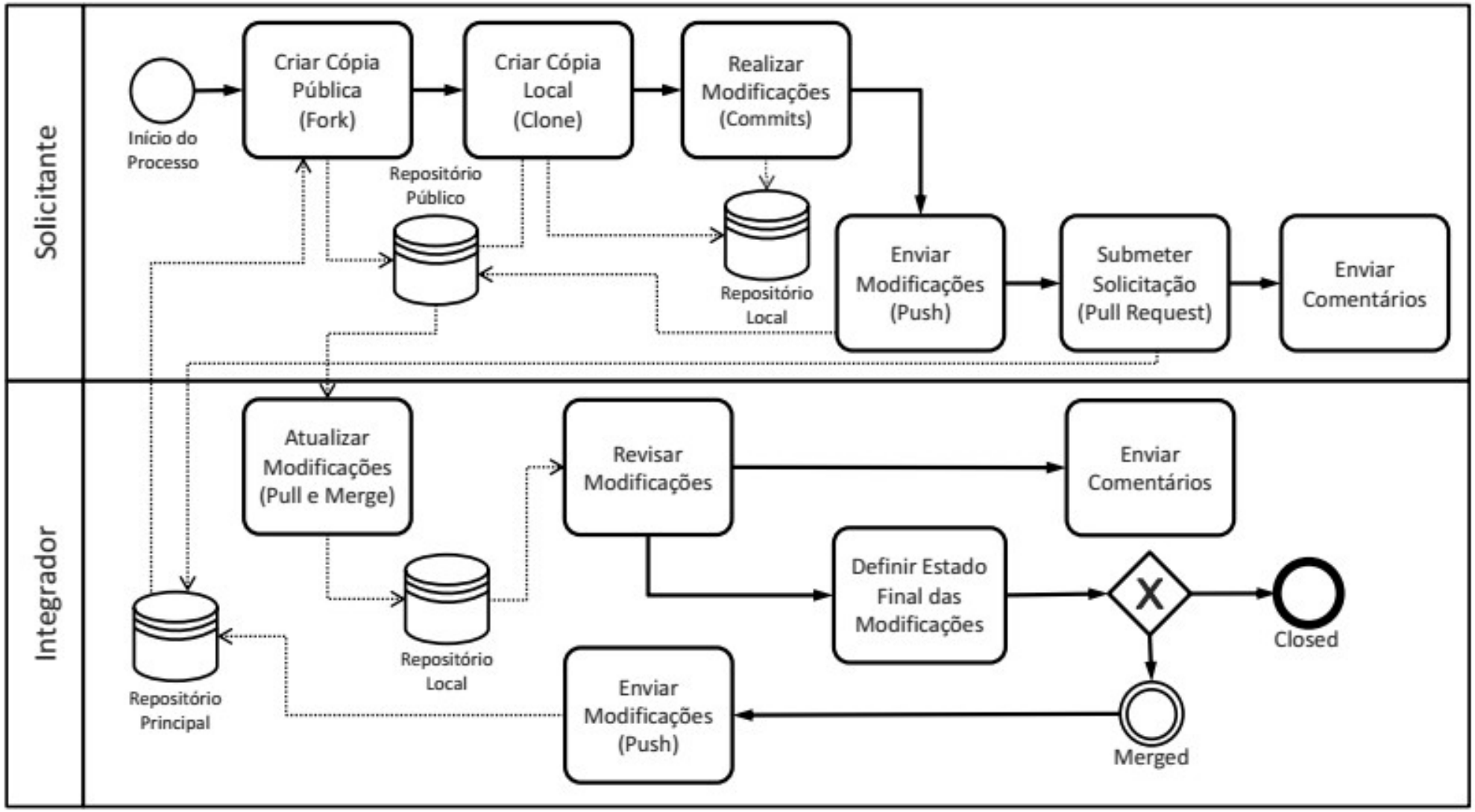
Workflows – Forking



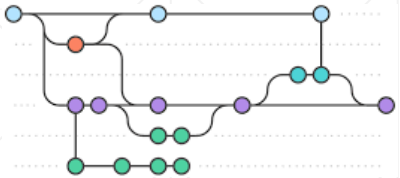
- Baseado no *Feature Branch*
- Muito comum em projetos *open-source*
- Permite a colaboração em qualquer projeto público
- Caracterizado pela presença de três repositórios
  - Local
  - Público
  - Principal

Workflows – Forking





Workflows – Forking



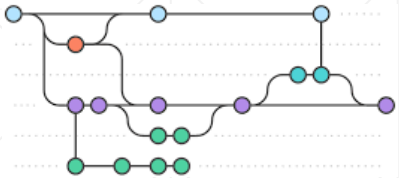
User 1

Criar um *fork* do repositório principal no GitHub

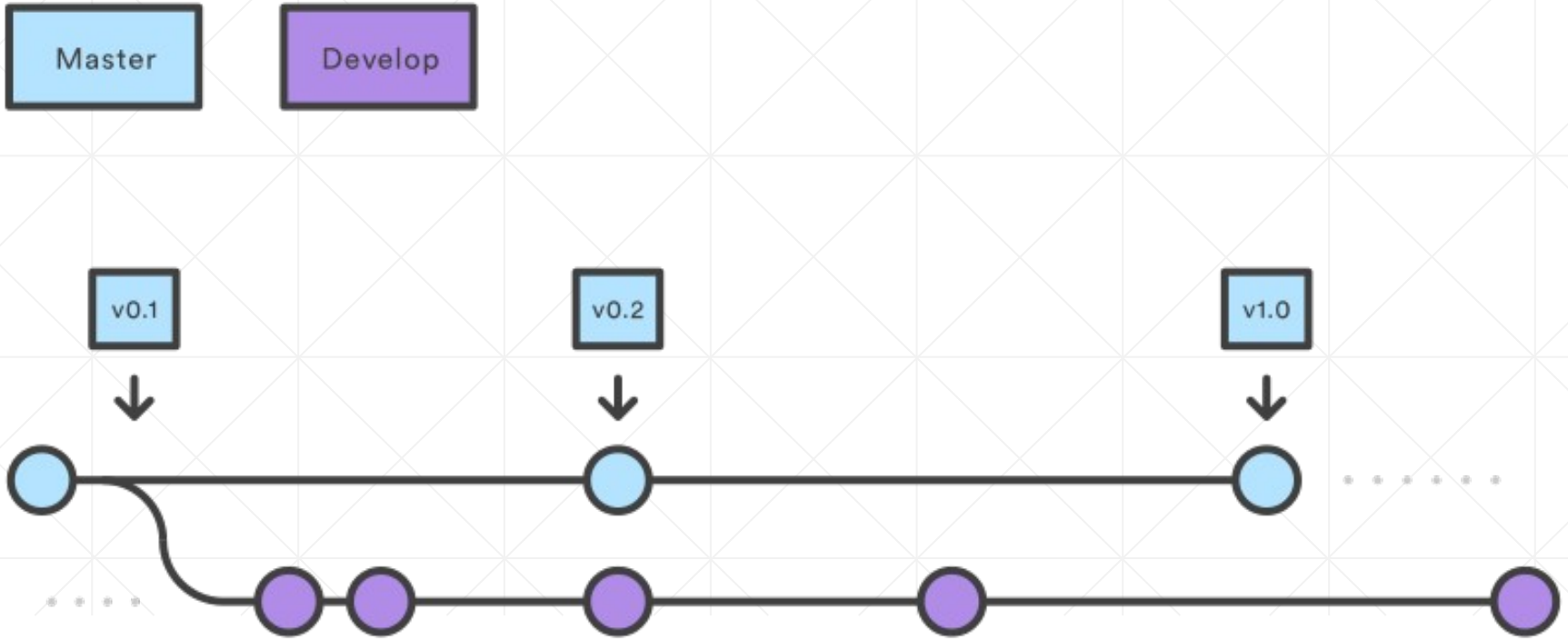
```
$ git clone git@github.com:user/repository repository-clone
$ cd repository-clone
$ git remote add upstream url-repository-principal
$ git remote -v
$ git checkout -b issue2
$ nano NewFeature.java //editar o conteúdo
$ git add .
$ git commit -m “add NewFeature.java closes #issue2”
$ git pull upstream master //atualizar o repositório local
$ git push origin issue2
```

Criar um *Pull Request* no repositório principal ou público

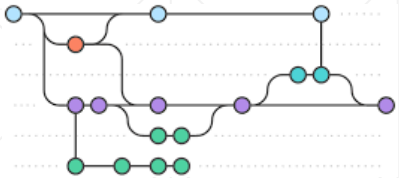
Workflows - GitFlow



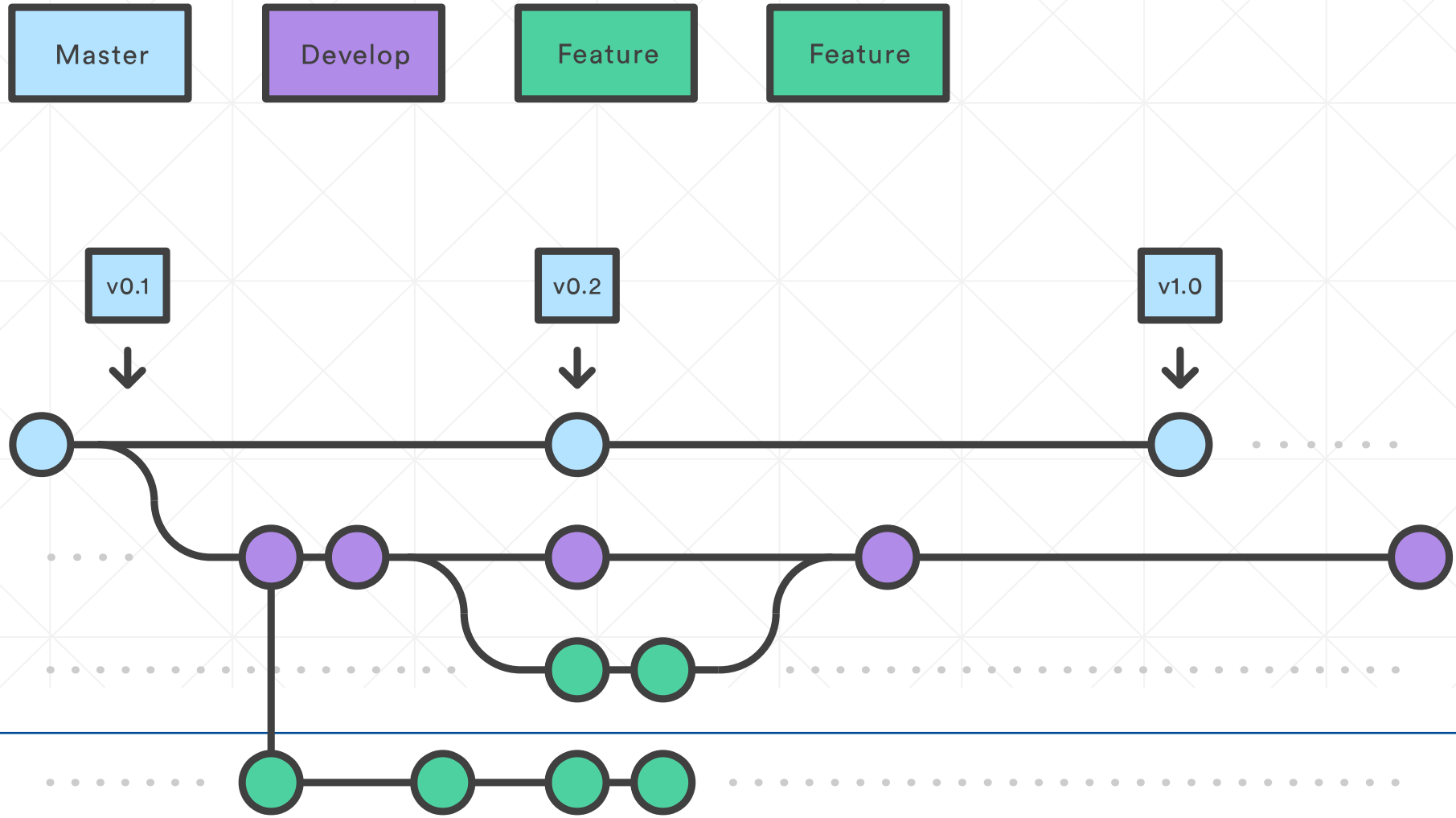
- Dois *branches* para registrar o histórico do projeto
  - As versões em ***master*** e os *commits* em ***develop***



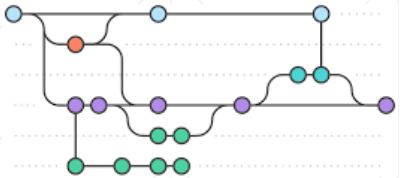
Workflows - GitFlow



- *I'm back, Feature Branch*



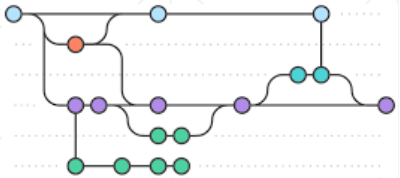
Workflows - GitFlow



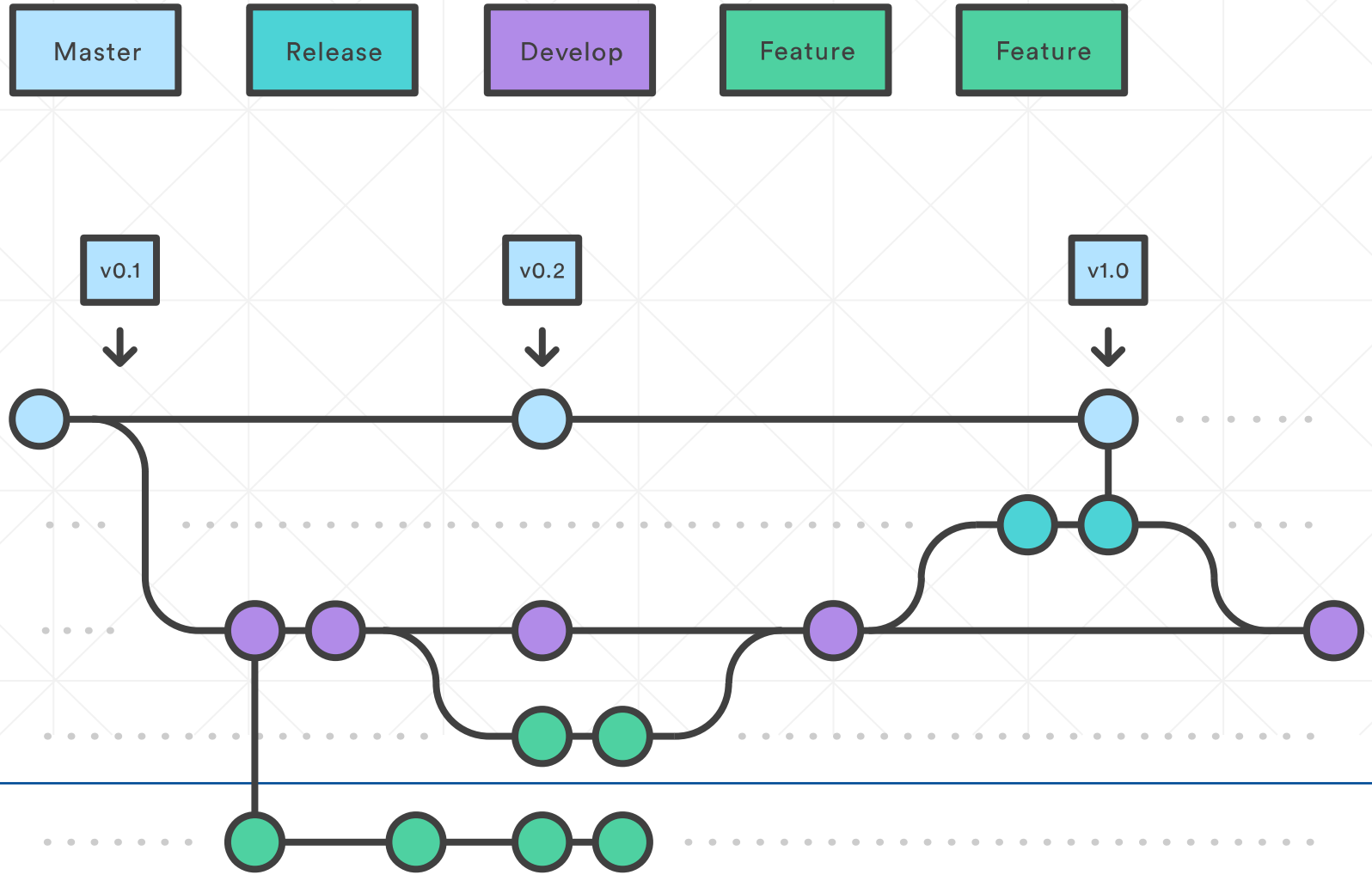
- Exemplo:

```
$ git checkout master
$ git checkout -b develop
$ git checkout -b feature_branch
//Realizar os commits no feature_branch
$ git checkout develop
$ git merge feature_branch
$ git checkout master
$ git merge develop
$ git branch -d feature_branch
$ git push origin master
```

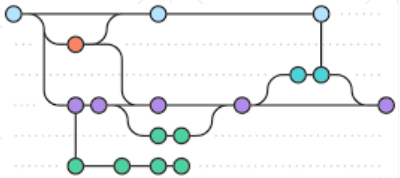
Workflows - GitFlow



- *Releases*



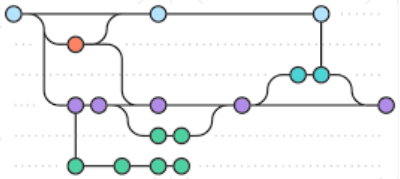
Workflows - GitFlow



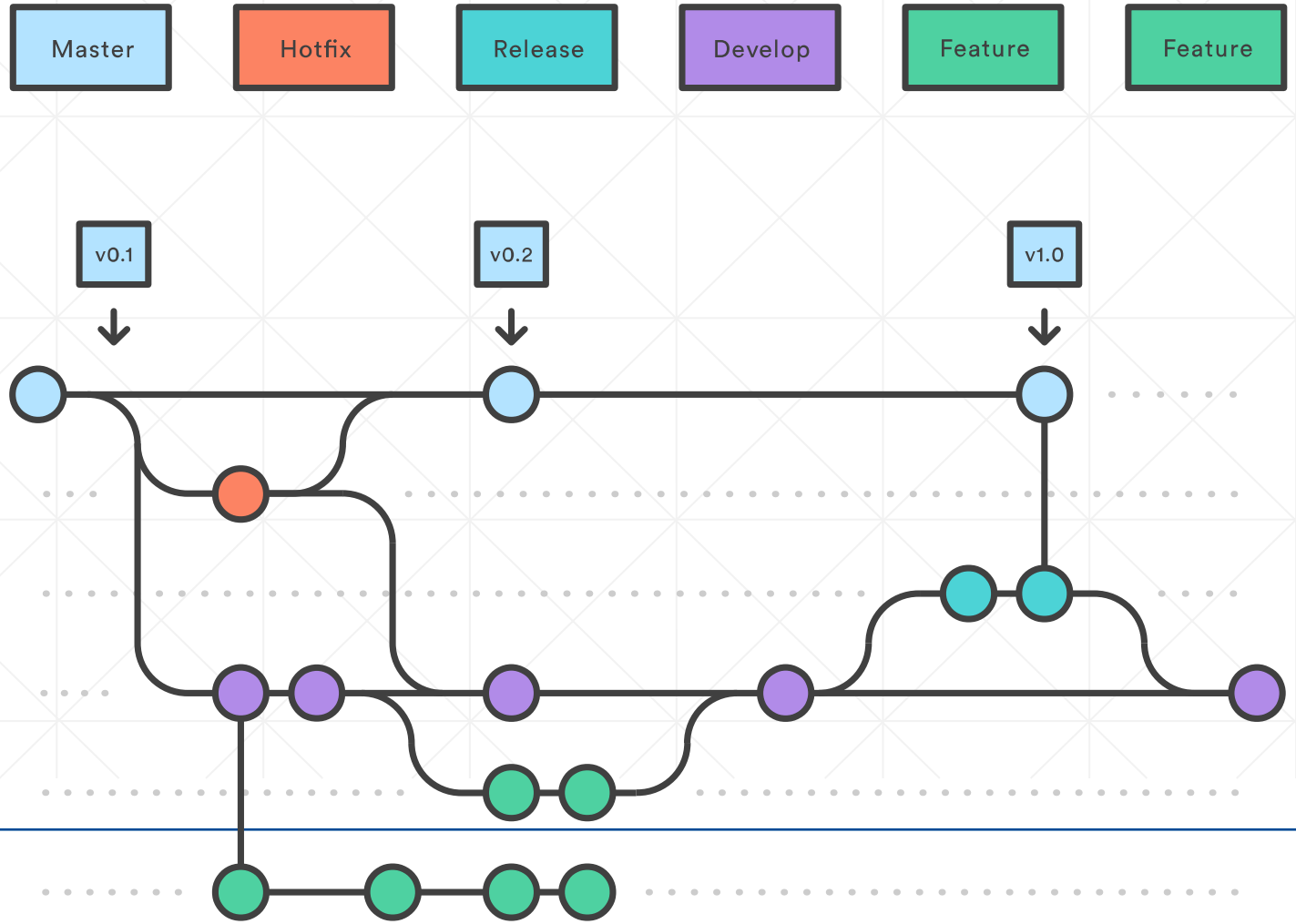
- Exemplo*:*

```
$ git checkout develop
$ git checkout -b release/1.0.0
$ git tag 1.0.0
//Realizar possíveis commits (bugs) no branch release/1.0.0
$ git checkout master
$ git merge release/1.0.0
$ git branch -d release/1.0.0
$ git push origin master --tags
$ git push origin develop
```

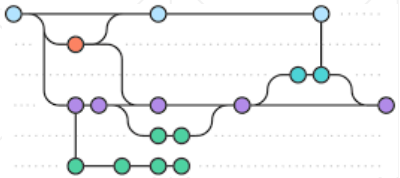
Workflows - GitFlow



- *Hotfix*



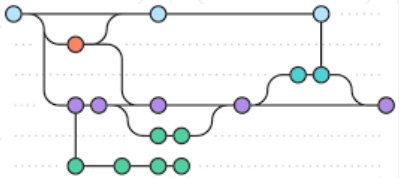
Workflows - GitFlow



- Exemplo:

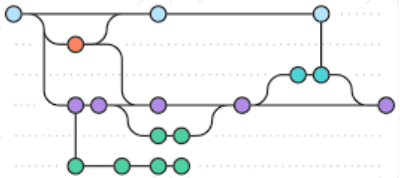
```
$ git checkout master
$ git checkout -b hotfix_branch
//Realizar os commits no hotfix_branch
$ git checkout develop
$ git merge hotfix_branch
$ git checkout master
$ git merge hotfix_branch
$ git push origin master --tags
```

Workflows - GitFlow



```
$ git flow init //inicializar o git flow
//Responder as perguntas sobre os branches (default)
$ git branch //*develop | master
$ git flow feature start feature_branch //checkout develop + checkout -b feature
//Realizar os commits no feature_branch
$ git flow feature finish feature_branch //checkout develop + merge
$ git flow release start 1.0.0 //checkout develop + checkout -b 1.0.0
$ git flow release finish '1.0.0' //merge master + develop
$ git flow hotfix start hotfix-branch //checkout master+checkout hotfix-name
$ git flow hotfix finish hotfix_branch //merge master + develop
```

Workflows - GitFlow



- Resumo:
  - Um *branch* ***develop*** é criado a partir do ***master***
  - Um *branch* ***release*** é criado a partir do ***develop***
  - ***Feature*** *branches* são criados a partir do ***develop***
  - Quando uma ***feature*** é concluída, o *merge* é feito no ***develop***
  - Quando o *branch* ***release*** é feito, o *merge* é feito em ***develop*** e ***master***
  - Se um problema surgir no ***master***, um ***branch hotfix*** é criada a partir do ***master***
  - Depois que problema for concluído, o *merge* é feito em ***develop*** e ***master***
- Mais informações:
  - http://danielkummer.github.io/git-flow-cheatsheet/index.pt_BR.html

Universidade Federal do Acre
Programa de Pós-Graduação em Ciência da Computação

Desenvolvimento Distribuído de Software: perspectivas e oportunidades com Mineração de Dados

Prof. Dr. Daricélio Soares
Prof. Dr. Manoel Limeira