

Controle de Versão

Sergio Ordine

sergio.ordine@eldorado.org.br

<https://developeracademy.eldorado.org.br/campinas/>

Agenda

- Introdução
 - História
- Conceitos básicos
- Repositório
- Conflitos (Merge)

Agenda

- Branches
 - Política de Branches
- Pull Request
- Stash

Introdução

Problemas

- Como organizar projetos e suas mudanças ao longo do tempo?
- Como achar o momento em que uma alteração foi feita?
- Como gerar a versão que tínhamos na data X?
- Como trabalhar em paralelo e unir os esforços mais facilmente?

História

História

- 1972 - SCCS
 - Source Code Control System (Bell Labs)
 - Unix
 - Arquivos de Código

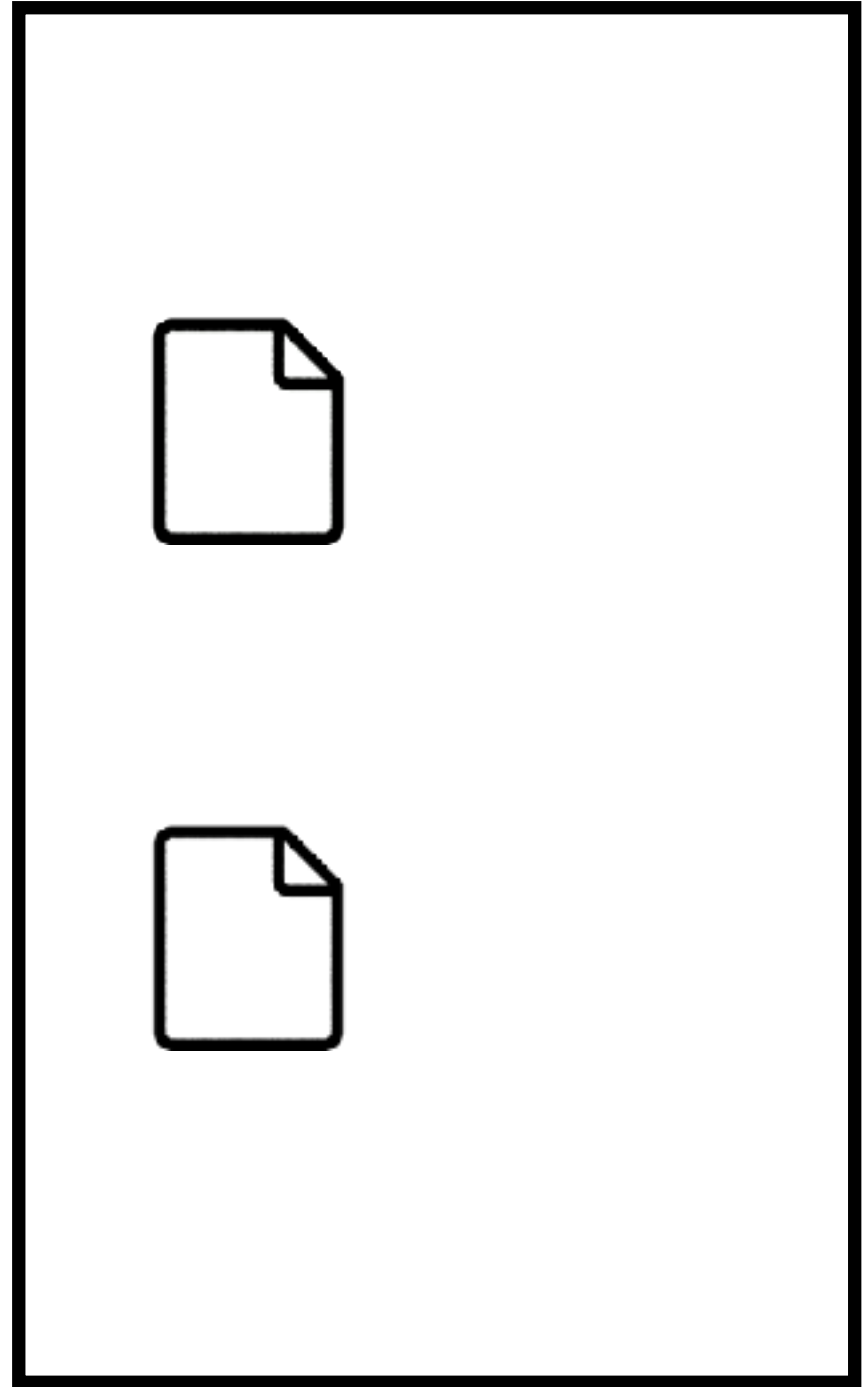
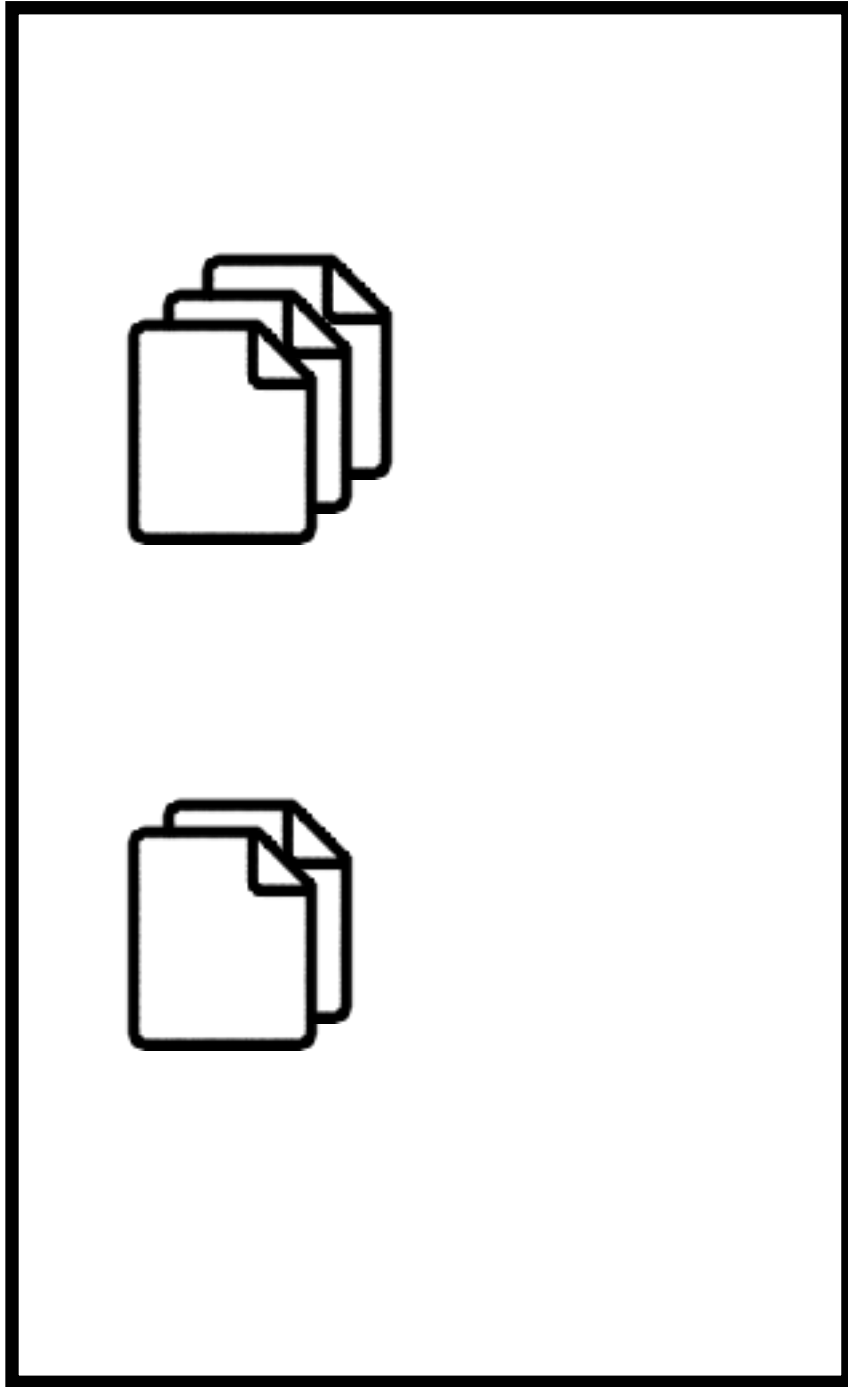
História

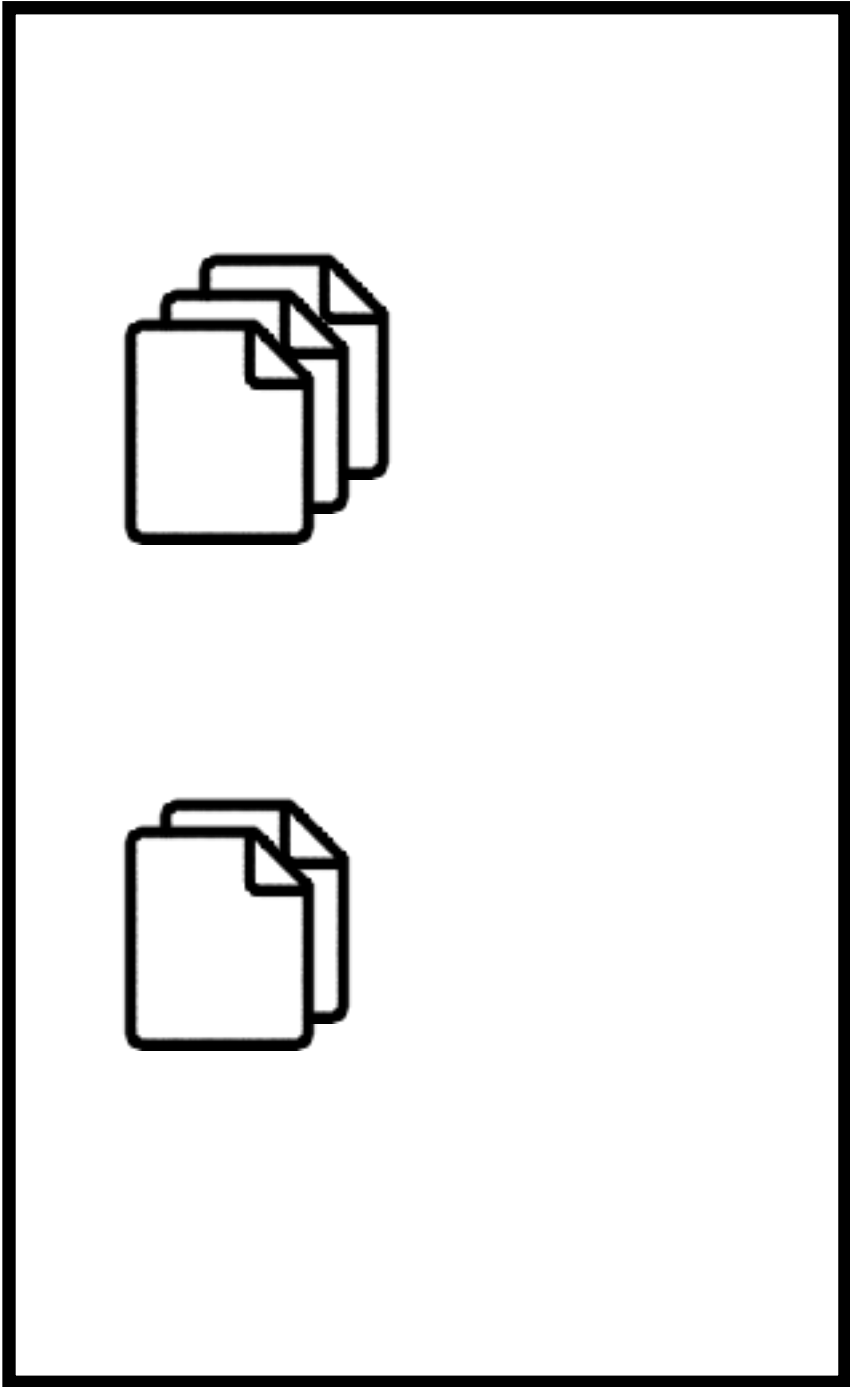
- 1982 - RCS
 - Revision Control System
 - Múltiplas plataformas
 - Arquivos de Texto
- Ambos para ambiente de desenvolvimento
 - Não serviam para compartilhar código

História

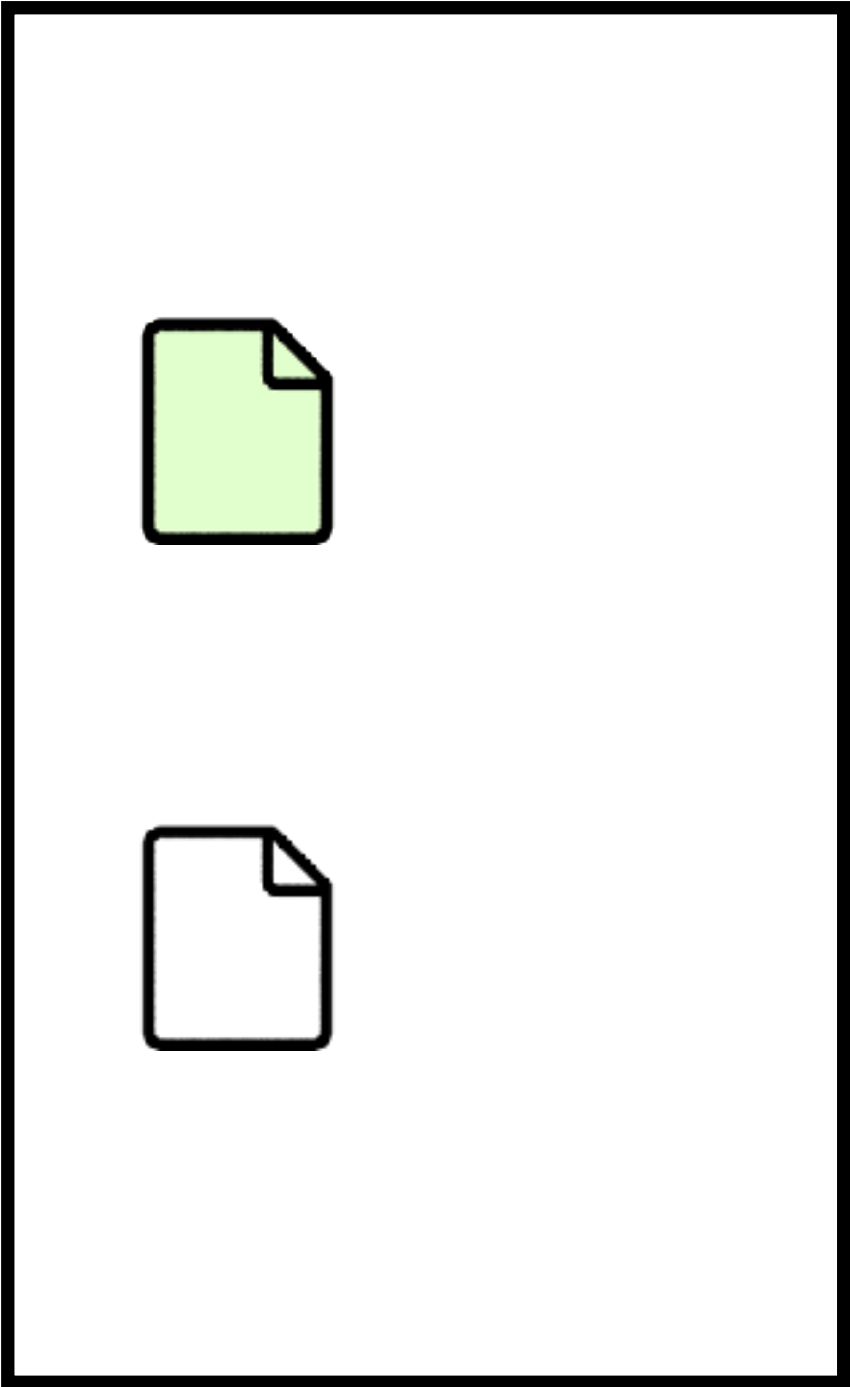
- 1982 - CVS
 - Concurrent Version System
 - Controle de versão centralizado
 - Foco em arquivos

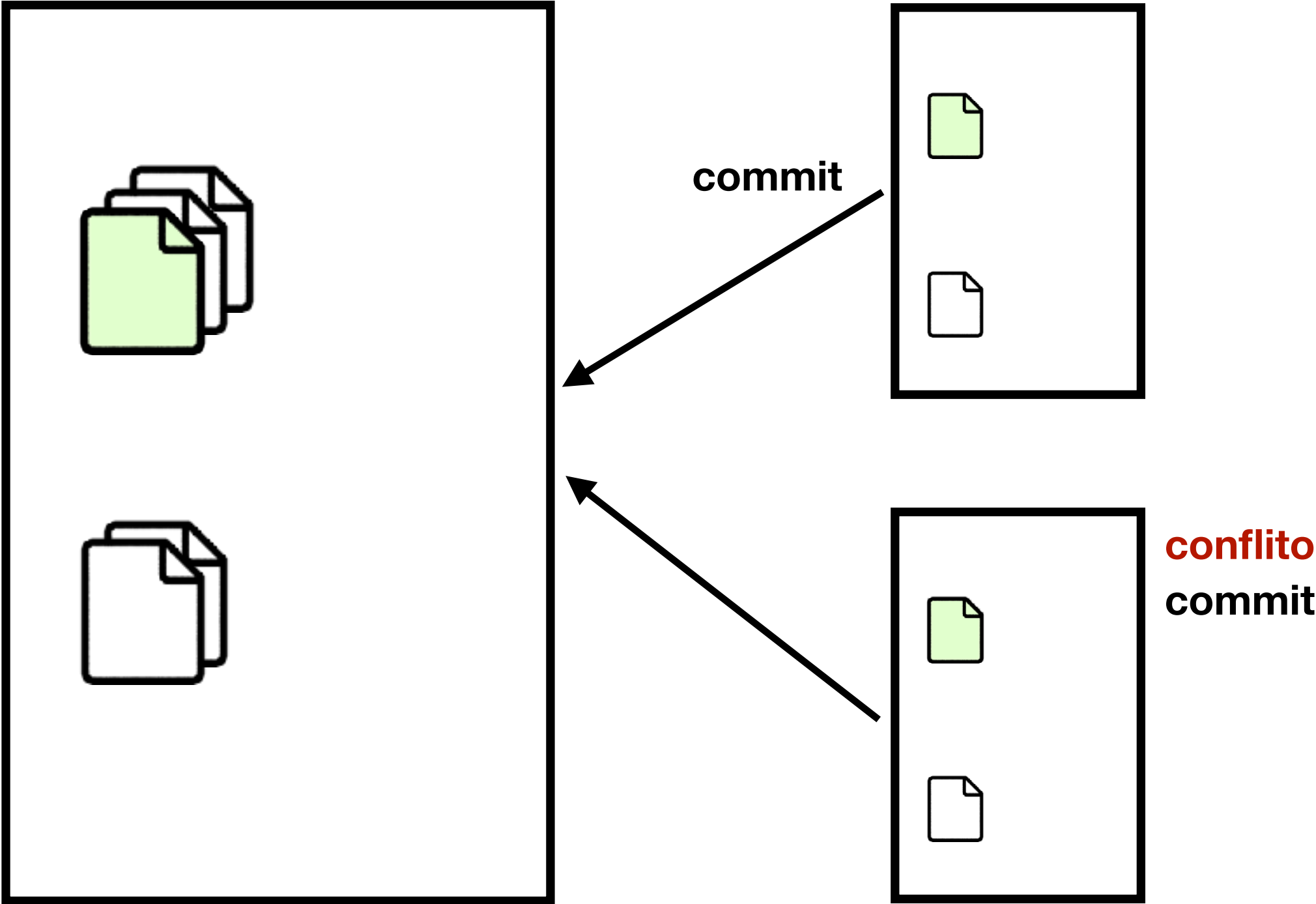






commit





História



- 1995 - perforce
- Controle de versão popular na época “.com”
- Controle de versão ainda hoje vastamente utilizado no Google

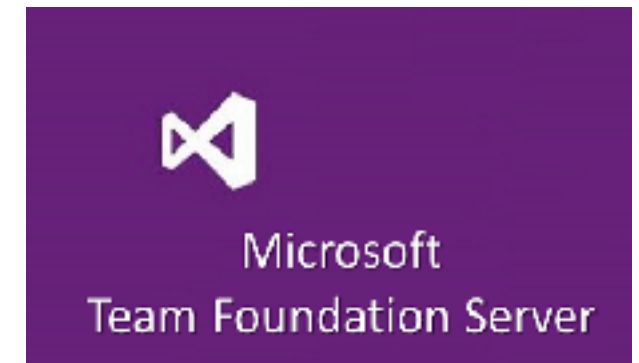
História

- 2000 - subversion
 - Controle de arquivos não textuais
 - Controle baseado em diretórios
 - Inclusão, remoção e renovação de arquivos



História

- 2010 - Microsoft TFS
 - Team Foundation Server
 - Integrado com o Visual Studio
 - Alto custo - associado a assinatura do MSDN



História

- 2005 - Git
- Controle de versão distribuído
- Desenvolvido quando o BitKeeper se tornou proprietário
- Operado principalmente em conjunto com o GitHub
- Plataforma em nuvem que hospeda projetos de código aberto



História

- 2005 - Mercurial
 - Controle de versão distribuído
 - Desenvolvido também quando o BitKeeper se tornou proprietário
 - Operado principalmente em conjunto com o GitHub
 - Plataforma em nuvem que hospeda projetos de código aberto



Atividade

- Crie uma conta em <https://github.com/>
- Registrar como colaborador no projeto (Sergio)
- Aceitar o convite

<https://github.com/sjordine/pucversioncontrol>

<https://github.com/sjordine/pucversioncontrol.git>

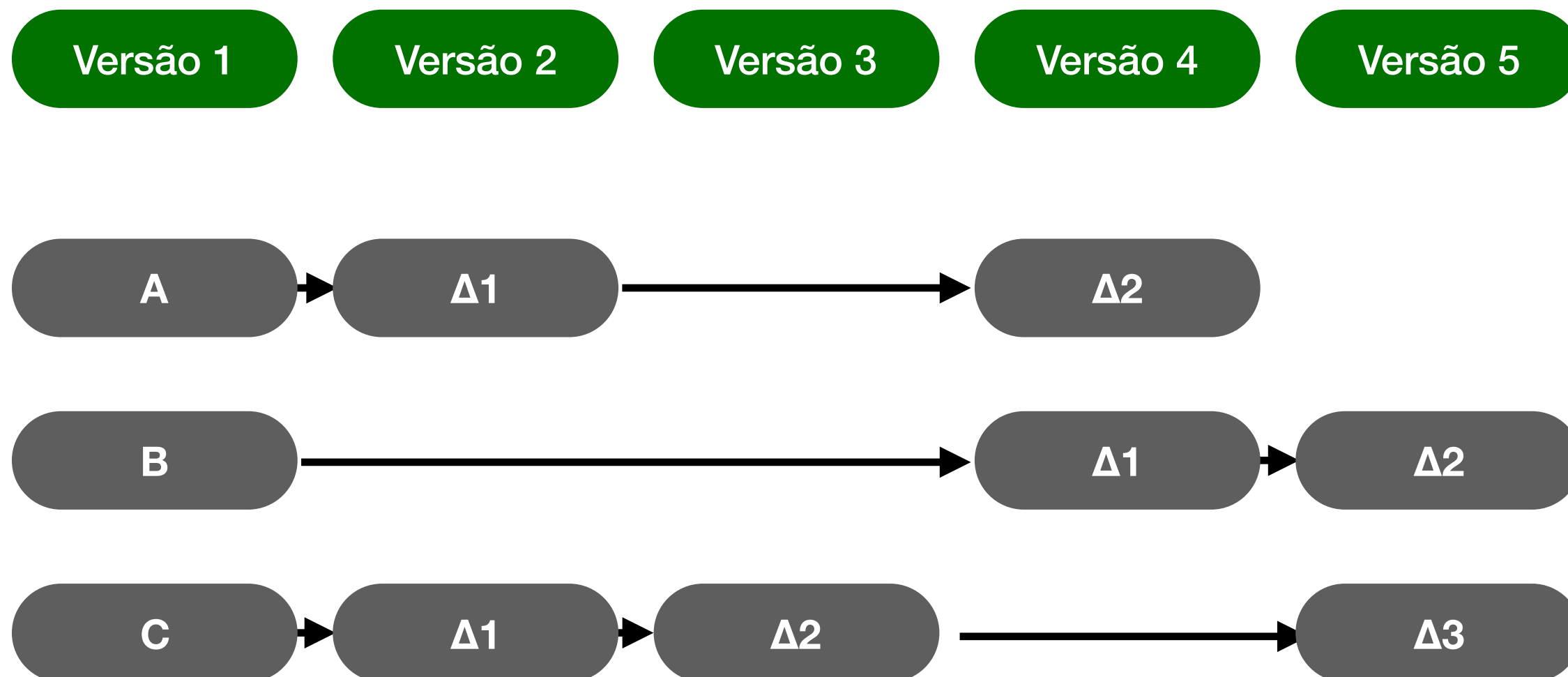
`https://github.com/sjordine/
pucversioncontrol/invitations`

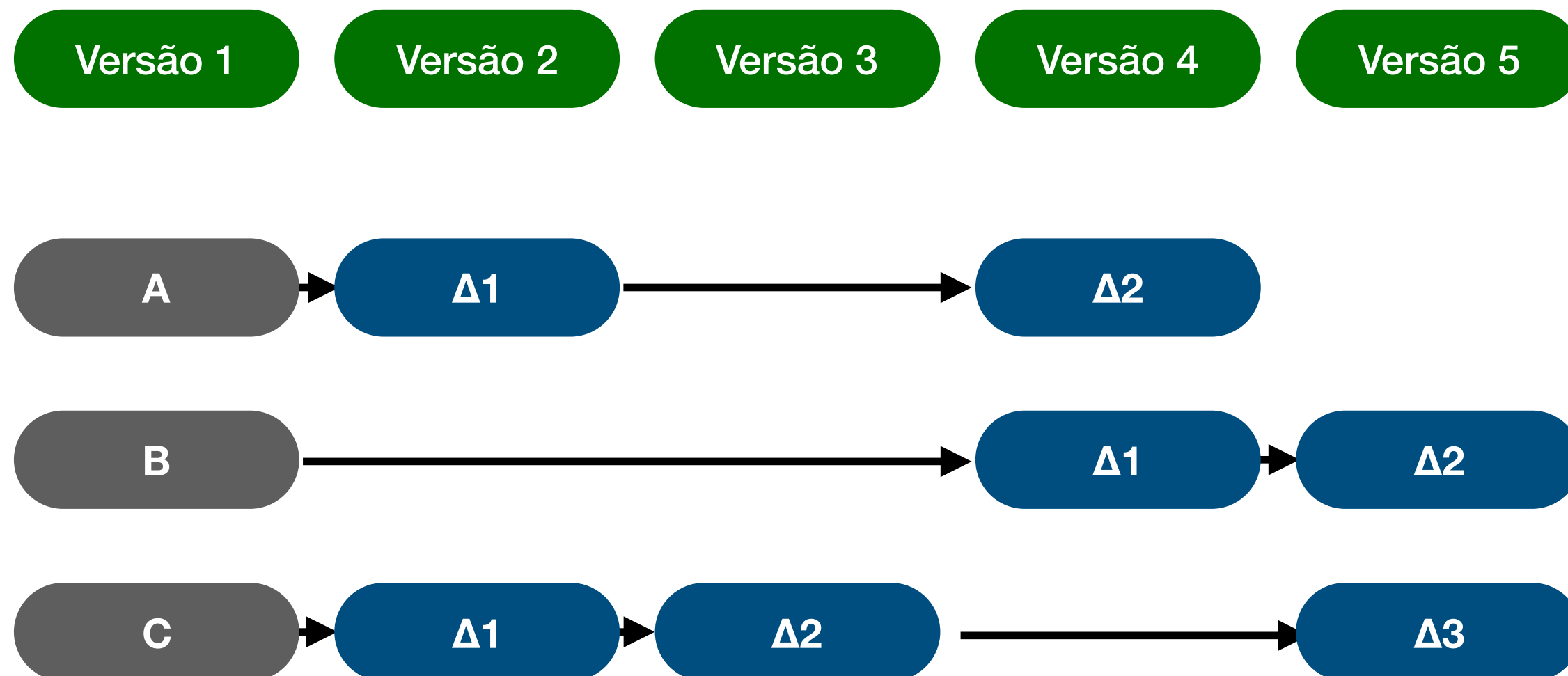
`http://twixar.me/79F3`

Conceitos básicos

Snapshots

- Alguns controles de versão guardam apenas diferenças
 - Mudanças de uma versão para outro
- Git guarda um “snapshot” (fotografia) dos arquivos
 - Réplica do sistema de arquivos





Versão 1

Versão 2

Versão 3

Versão 4

Versão 5

A

A1

A1

A2

A2

B

B

B

B1

B2

C

C1

C2

C2

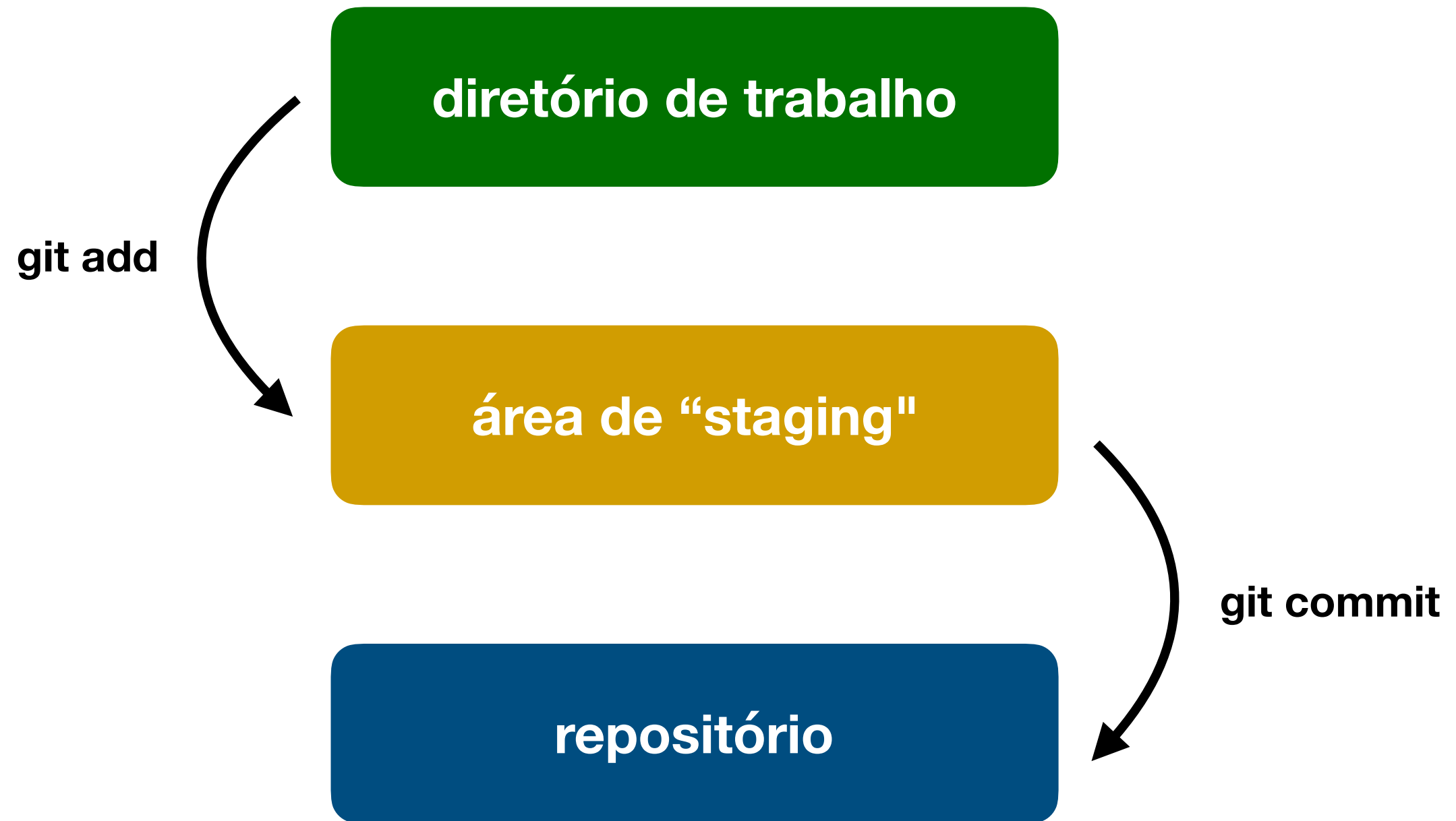
C3

Vantagens

- Quase todas as operações são locais
- Integridade
 - Não se pode alterar dados de uma versão facilmente
- Em geral apenas adiciona dados ao servidor
 - Não existe praticamente nada irreversível

Atividade

- Criar o repositório (clone)
- Criar um arquivo <SeuNome>.txt
- Escrever seu nome no arquivo
- Colocar o arquivo no controle de versão

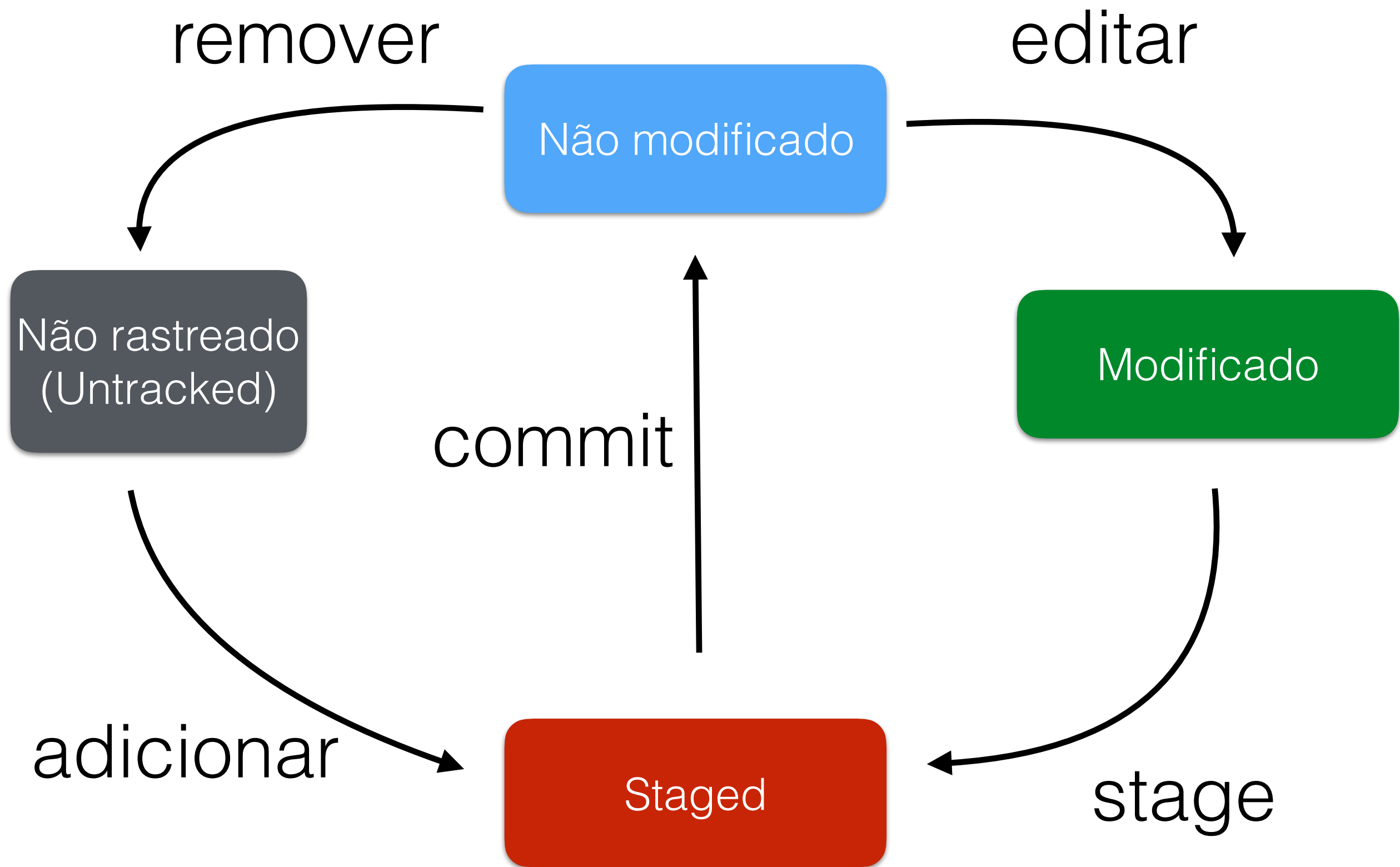


Comandos

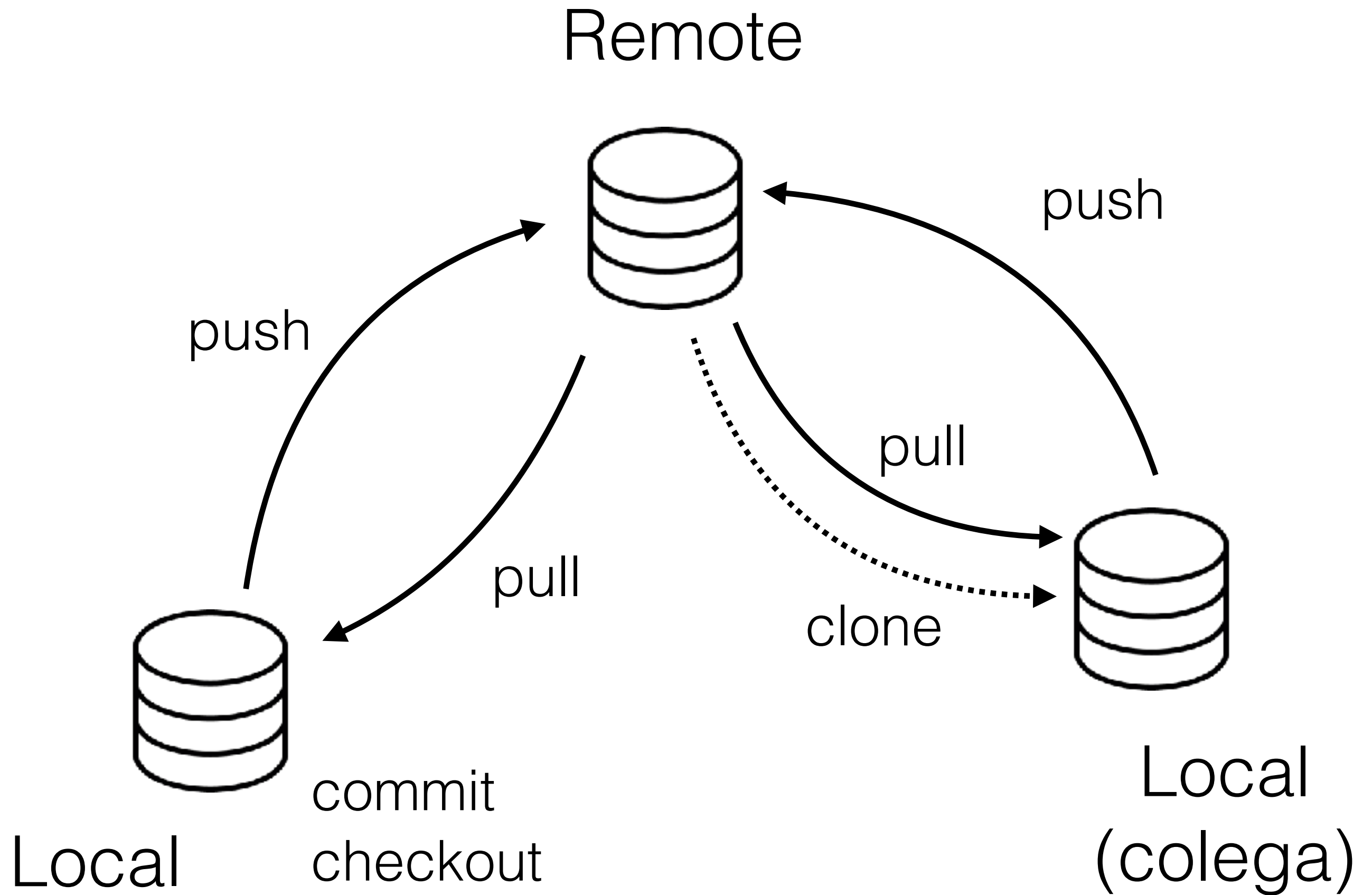
- **git status**
 - Mostra o estado dos arquivos no git
- **git add <file(s)>**
 - Prepara um arquivo para o controle de versão (stage)
- **git reset**
 - Reverte o stage (unstage)
- **git commit**
 - Adiciona o arquivo ao controle de versão

Atividade

- Alterar o arquivo para conter seu RA
- Criar uma nova versão



Repositório



Comandos

- **git push**
 - Atualiza as referências do repositório remoto com as atualizações locais.
- **git pull**
 - Busca e integra o repositório externo no repositório local.

Atividade

- Atualizar o repositório (pull)



- Incluir seu nome como último no arquivo lista.txt

- Fazer commit e push



- Passar para a próxima pessoa fazer o mesmo

Discussão

- Imagine que este é um arquivo utilizado no projeto
- Quais vantagens em lidarmos com ele assim?
- Quais as desvantagens?
- Este método é eficiente?

Atividade

- Vamos todos atualizar nosso repositório
- Voluntário: Incluir uma nova linha no arquivo
- Fazer commit e pull
- Vou alterar uma linha também (sem fazer pull)
- O que acontece?

Conflitos (Merge)

'refs/heads/master' is behind 'refs/remotes/origin/master'. Update your branch by doing a Pull.

Pull (fast-forward if possible)

Force Push

Cancel



Merge Failed

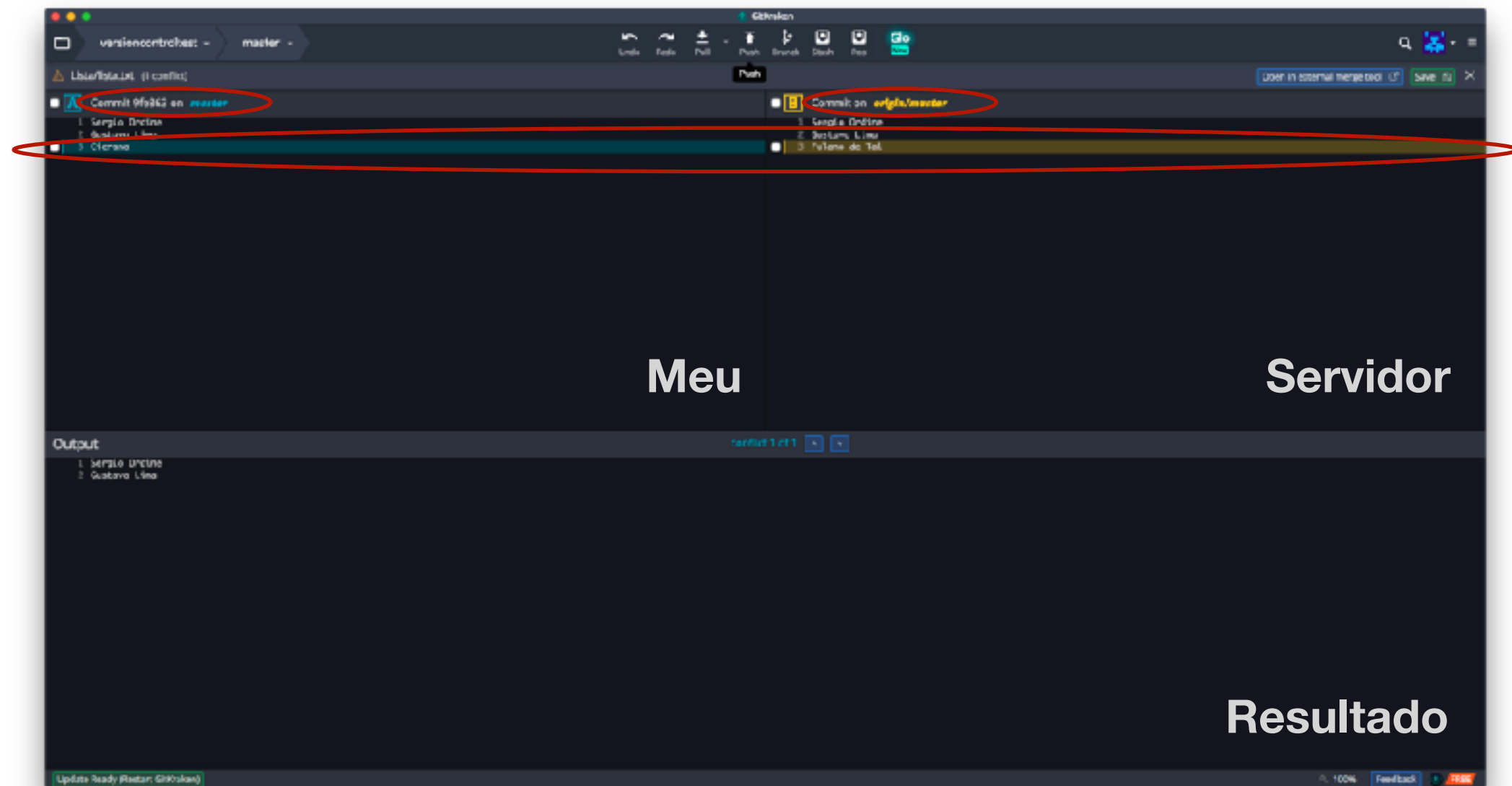


There are merge conflicts that need to be resolved



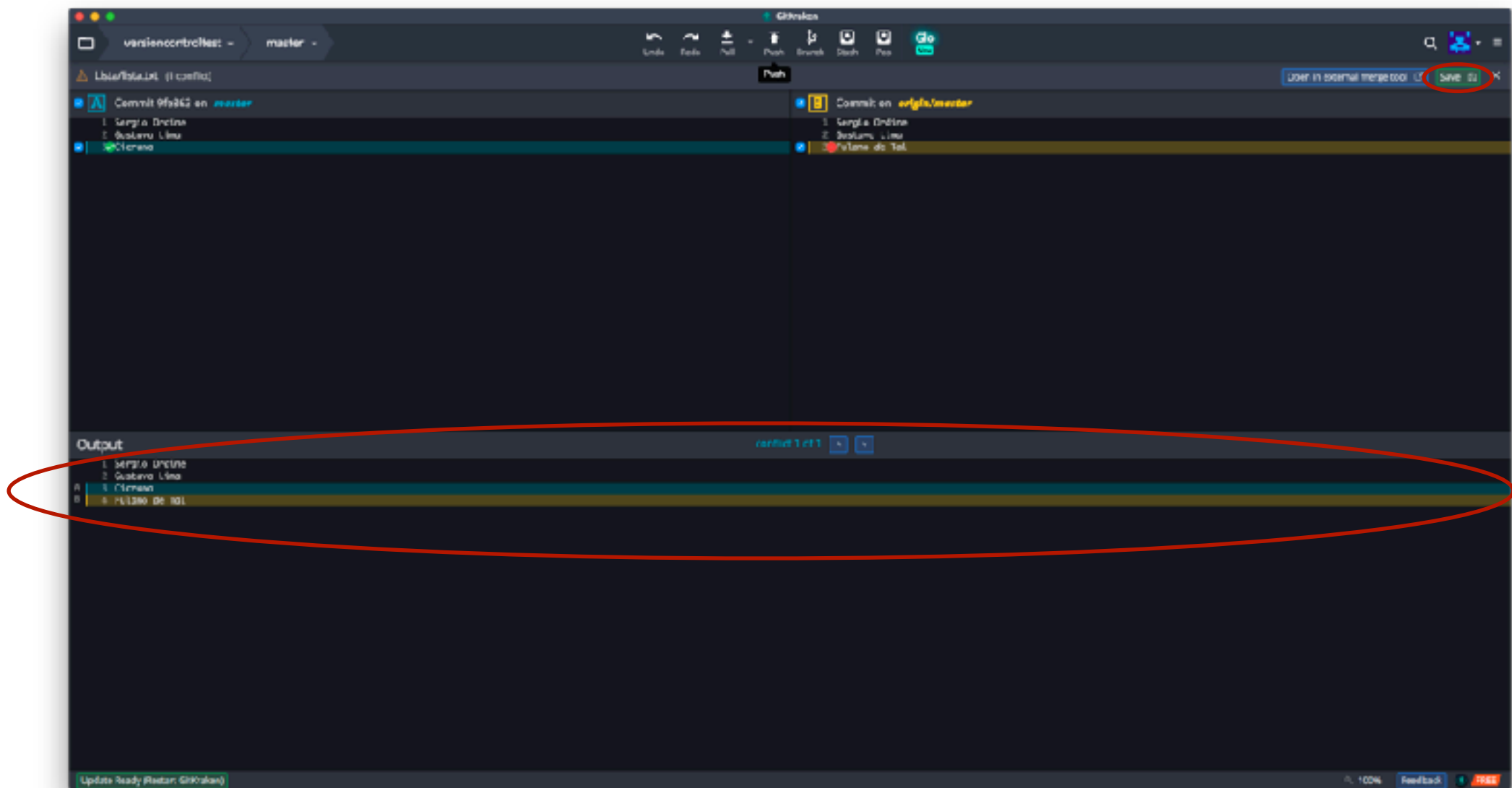
Push Failed

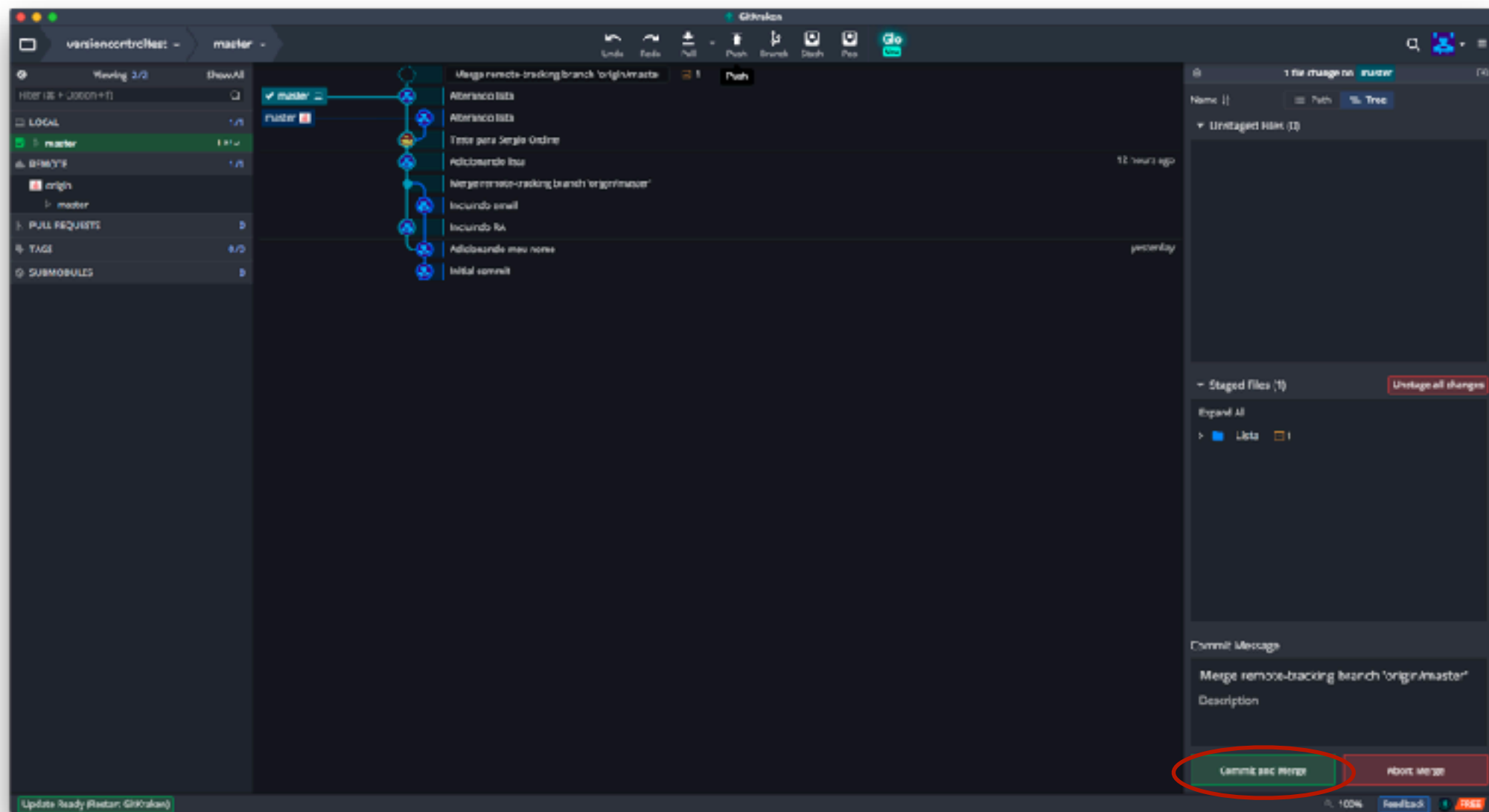




2 Gustavo Lima
3 + Cicrano

2 Gustavo Lima
3 - Fulano de Tal



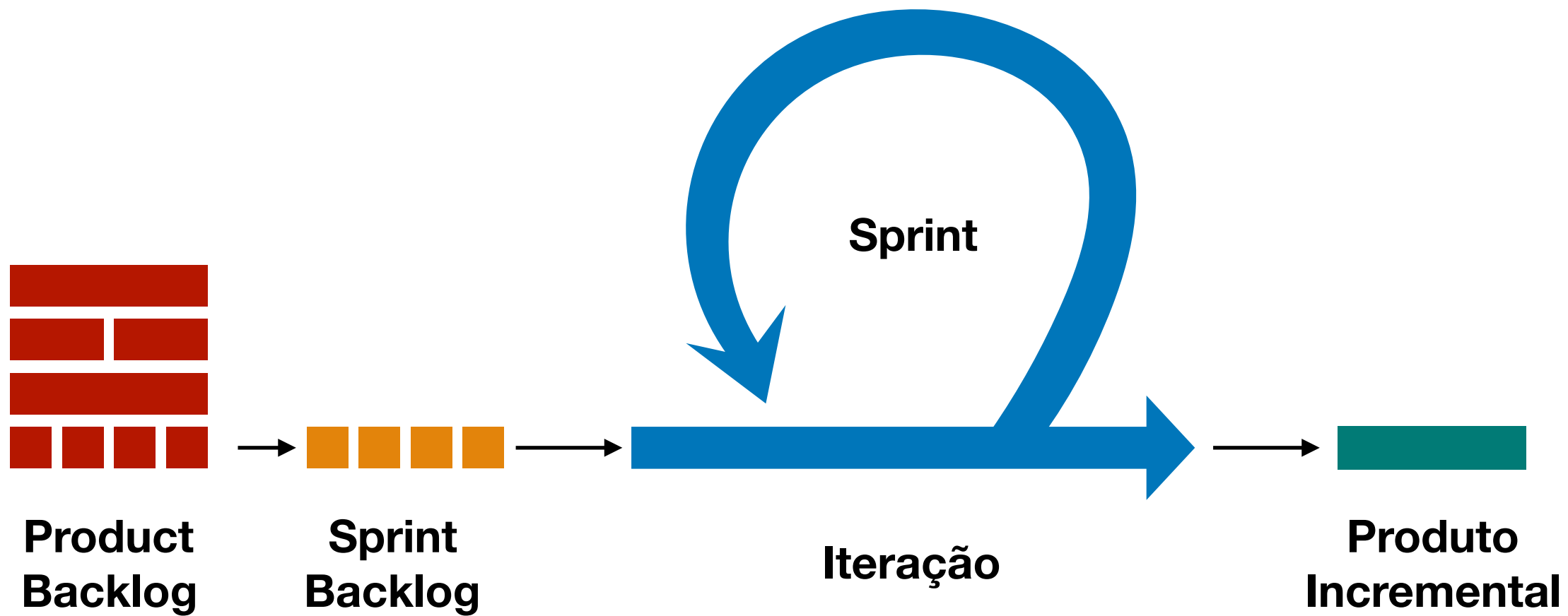


Atividade

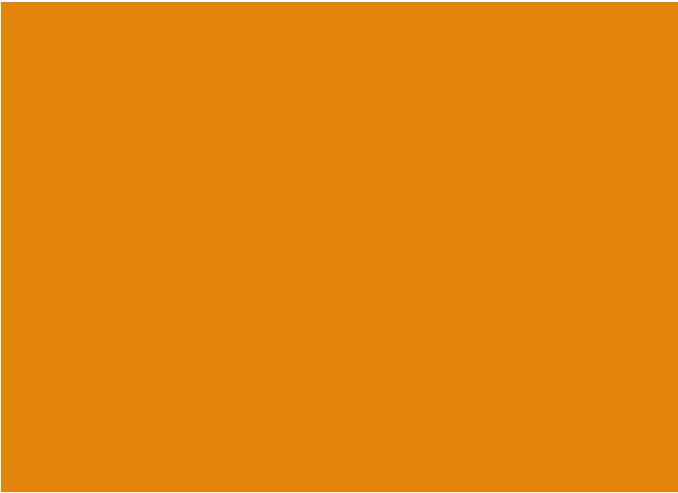
- Vamos nos dividir em grupos de 3 pessoas
- Cada pessoa vai criar um parágrafo de um início de história
 - Fantasia Medieval em uma cidade
 - Descreva a cidade (1 parágrafo)
 - Descreva o herói (1 parágrafo)
 - Descreva o vilão (1 parágrafo)
- Merge dos 3 parágrafos.

Discussão

- A história fez sentido?
- Poderíamos ter melhorado o processo? Como?



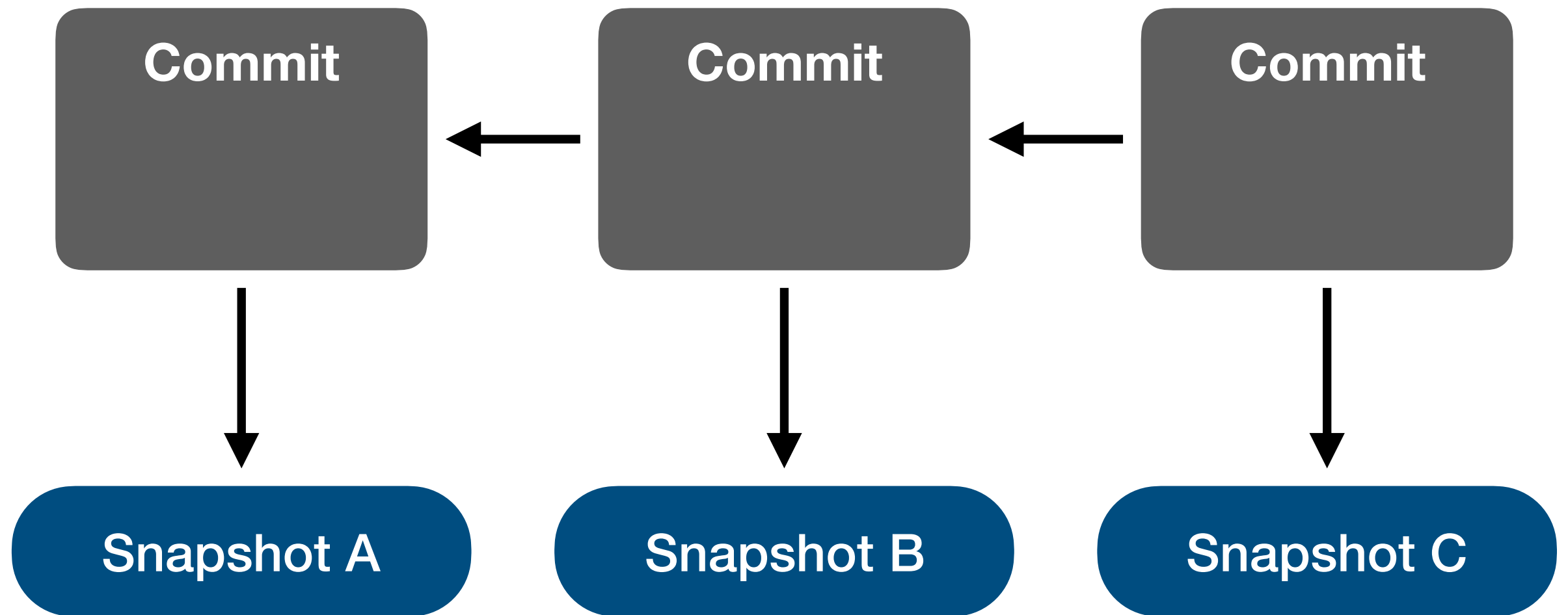
História



Tarefas

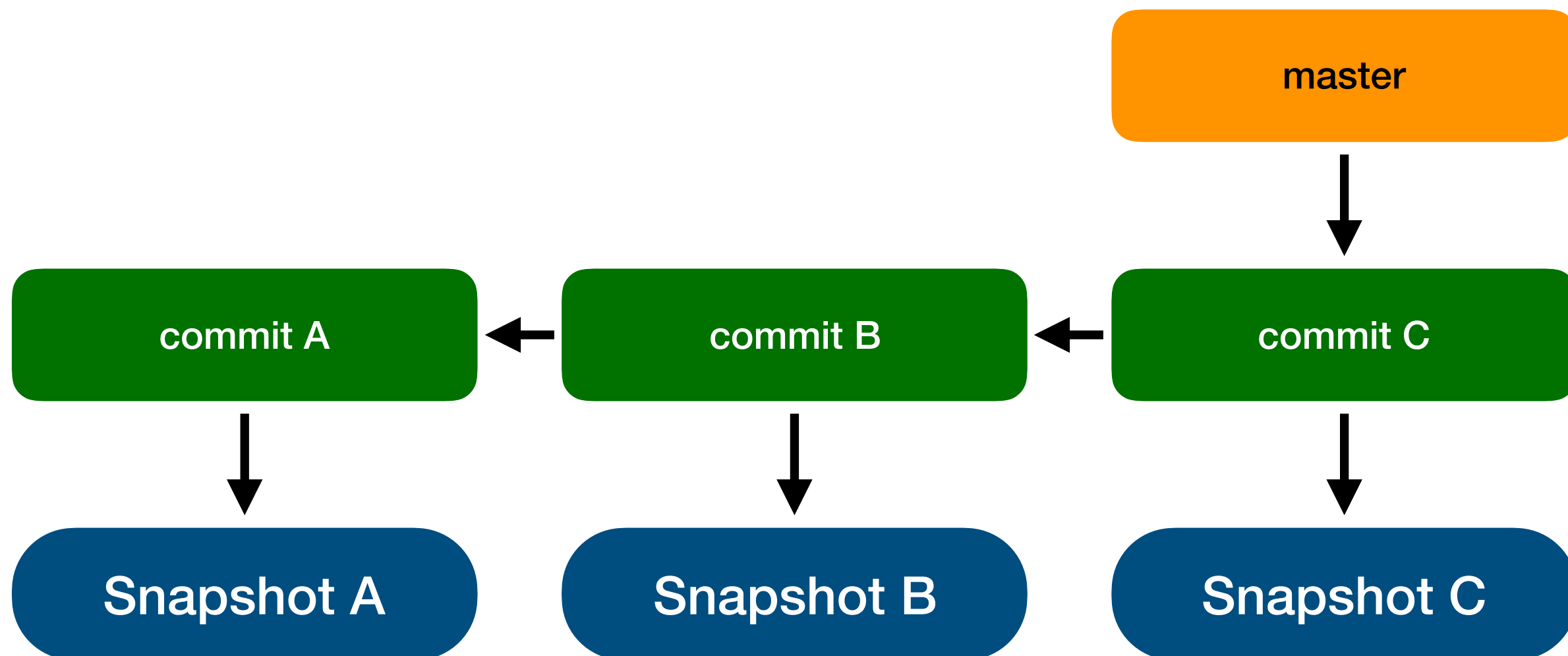
A decorative graphic consisting of a 3x3 grid of teal squares. The grid is formed by orange lines that intersect to create the squares. The lines are of uniform thickness and the squares are of uniform size.

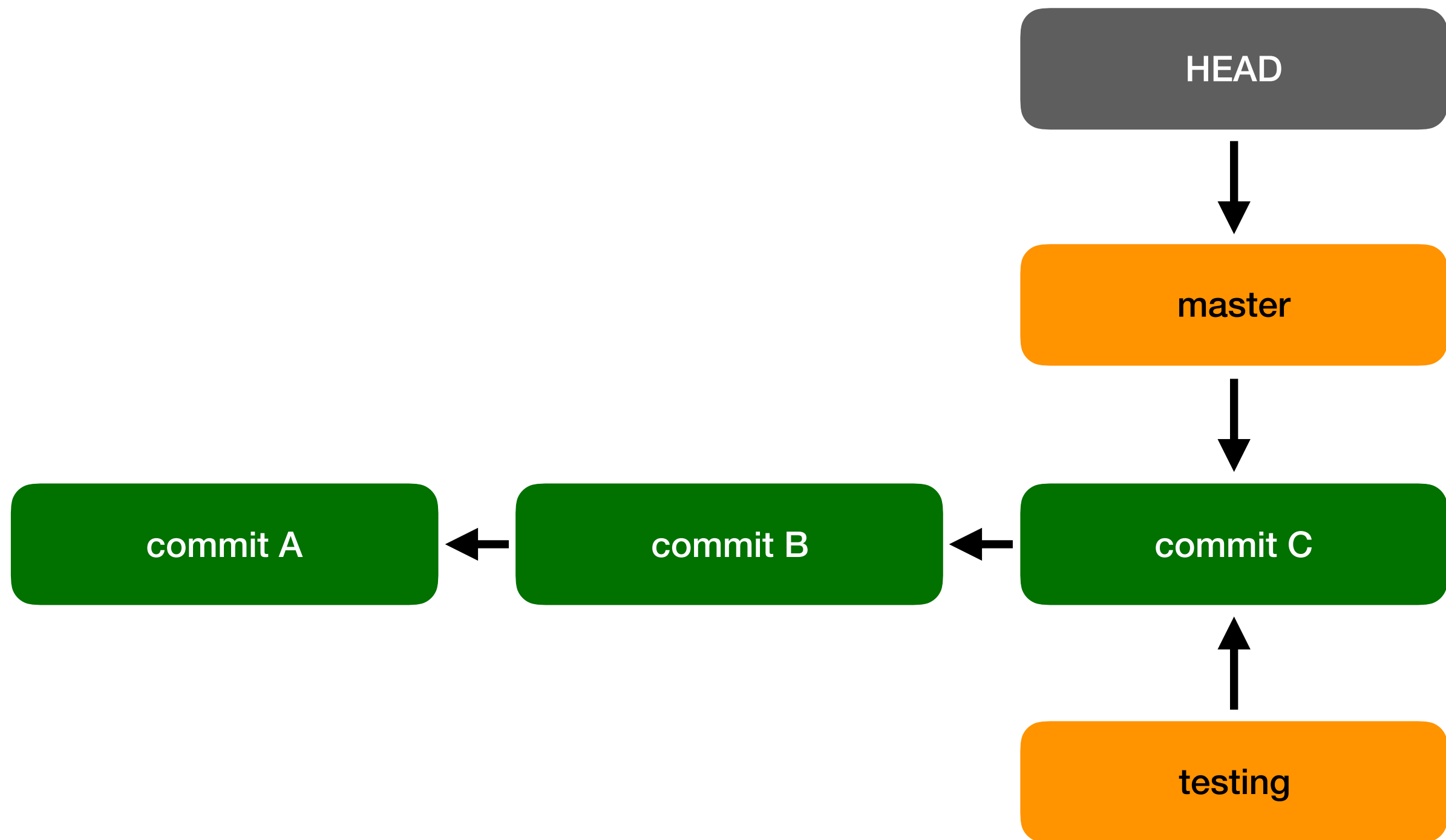
Branches

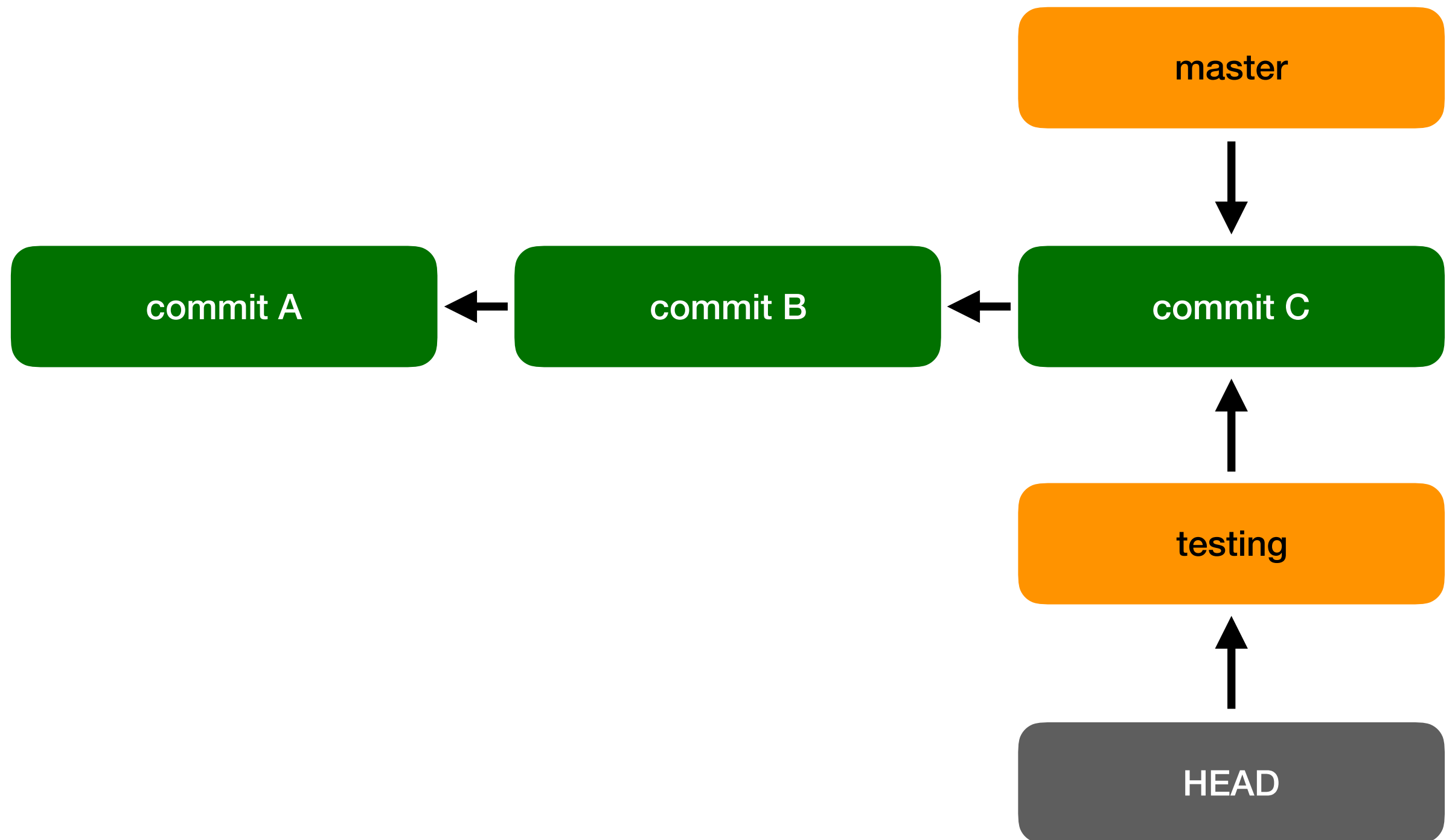


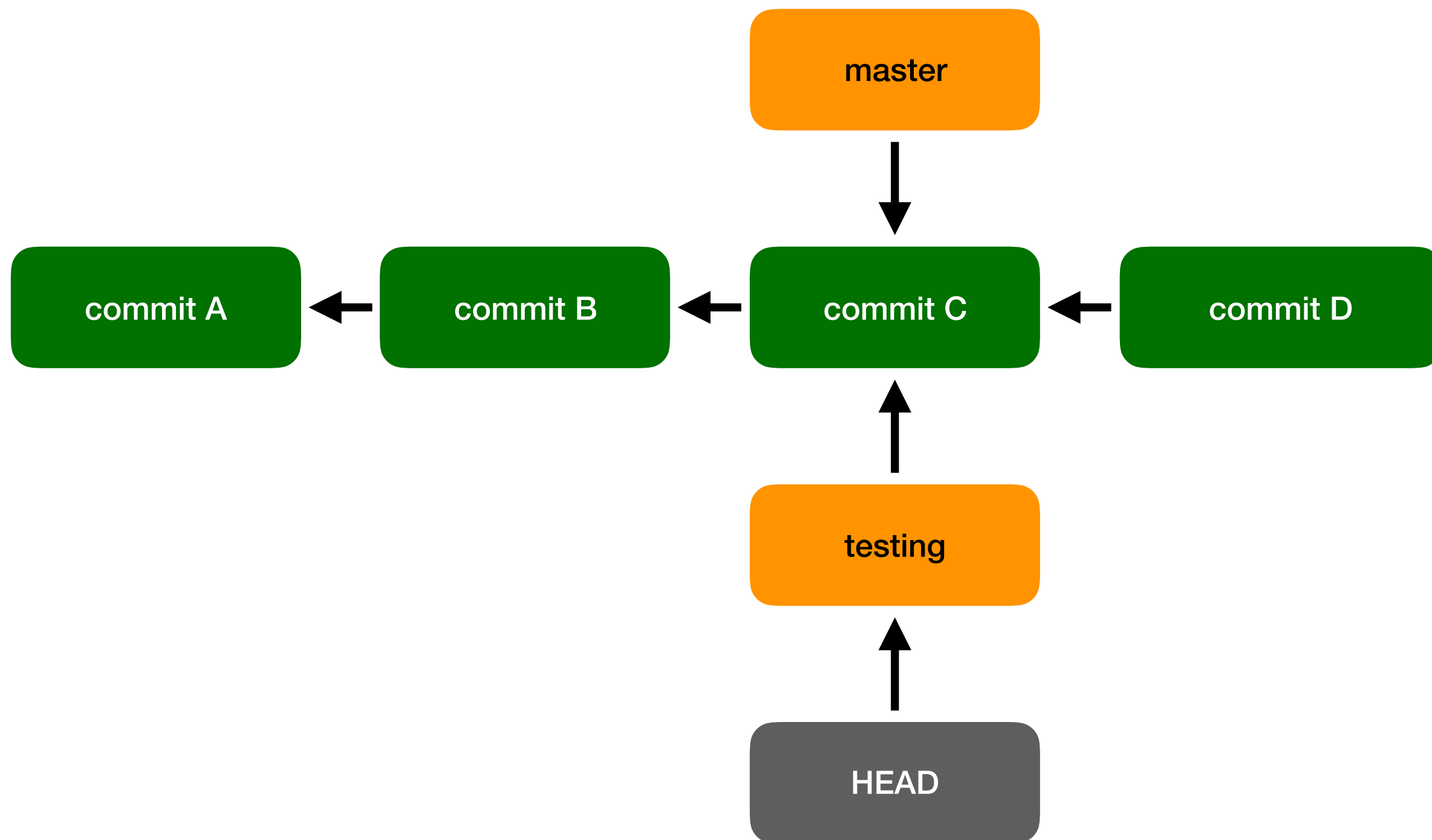
Branch

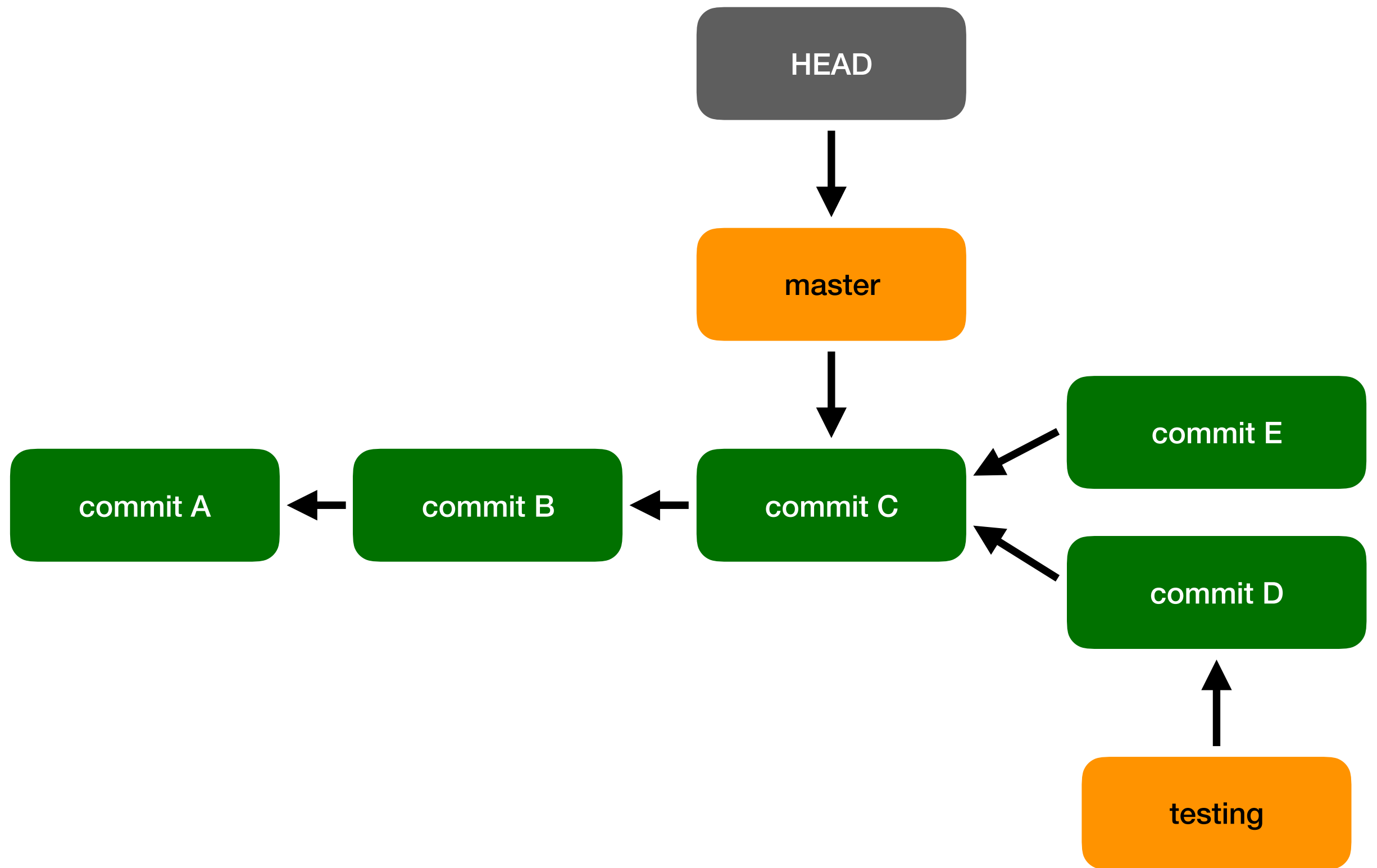
- Um ponteiro (leve) que pode ser movido entre os commits
- Um repositório inicia com o branch **master**





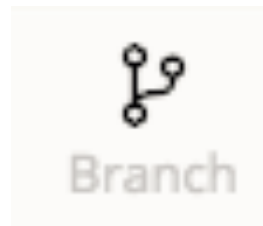






Comandos

- **git branch <nome>**



- Cria um novo branch com o nome dado

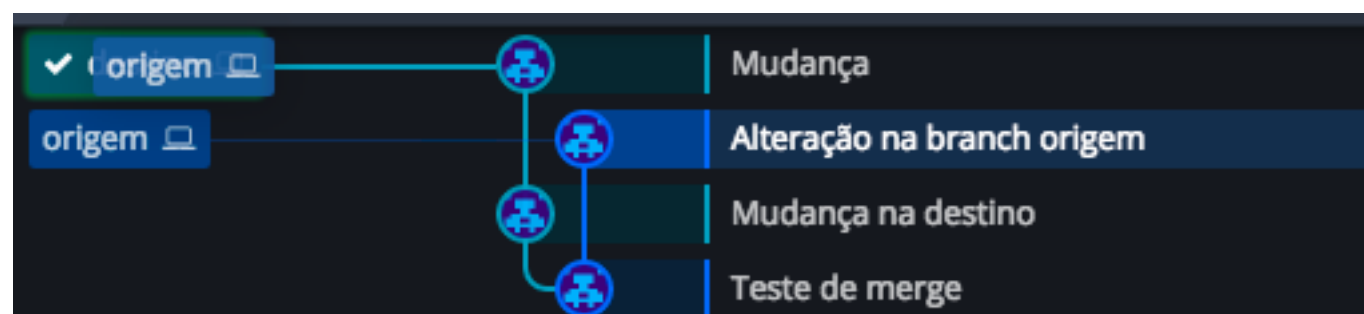
- **git checkout <nome>**

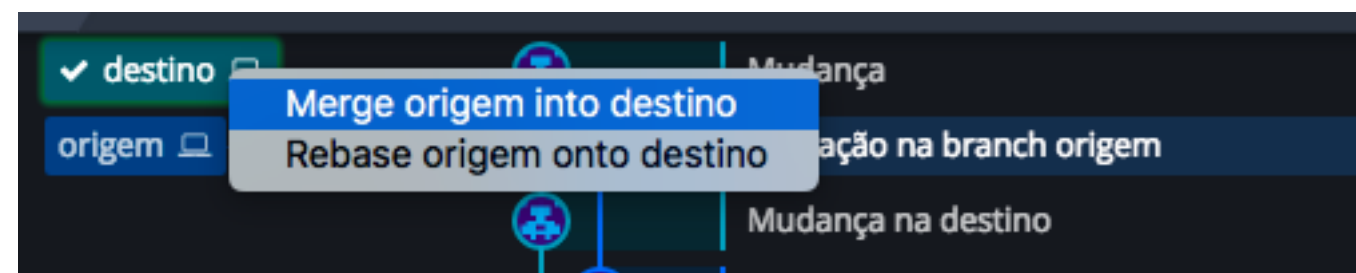
- Troca seu branch atual (HEAD) para o branch desejado

No GitKraken

- Ir para o branch de onde iremos integrar o código
- Arrastar o branch para o branch destino
- Ele fará o processo de merge
- Resolver conflitos (se necessário)





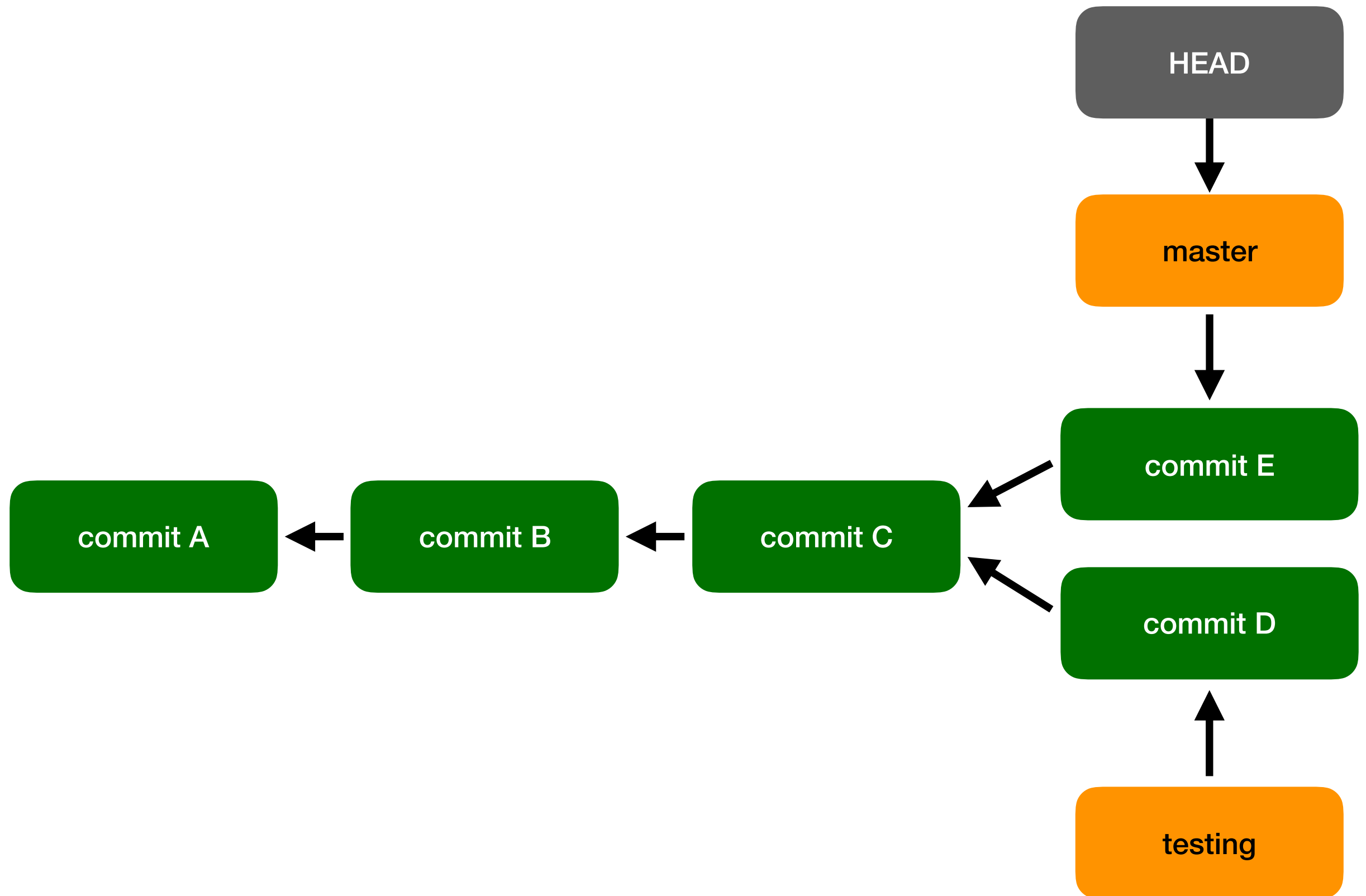


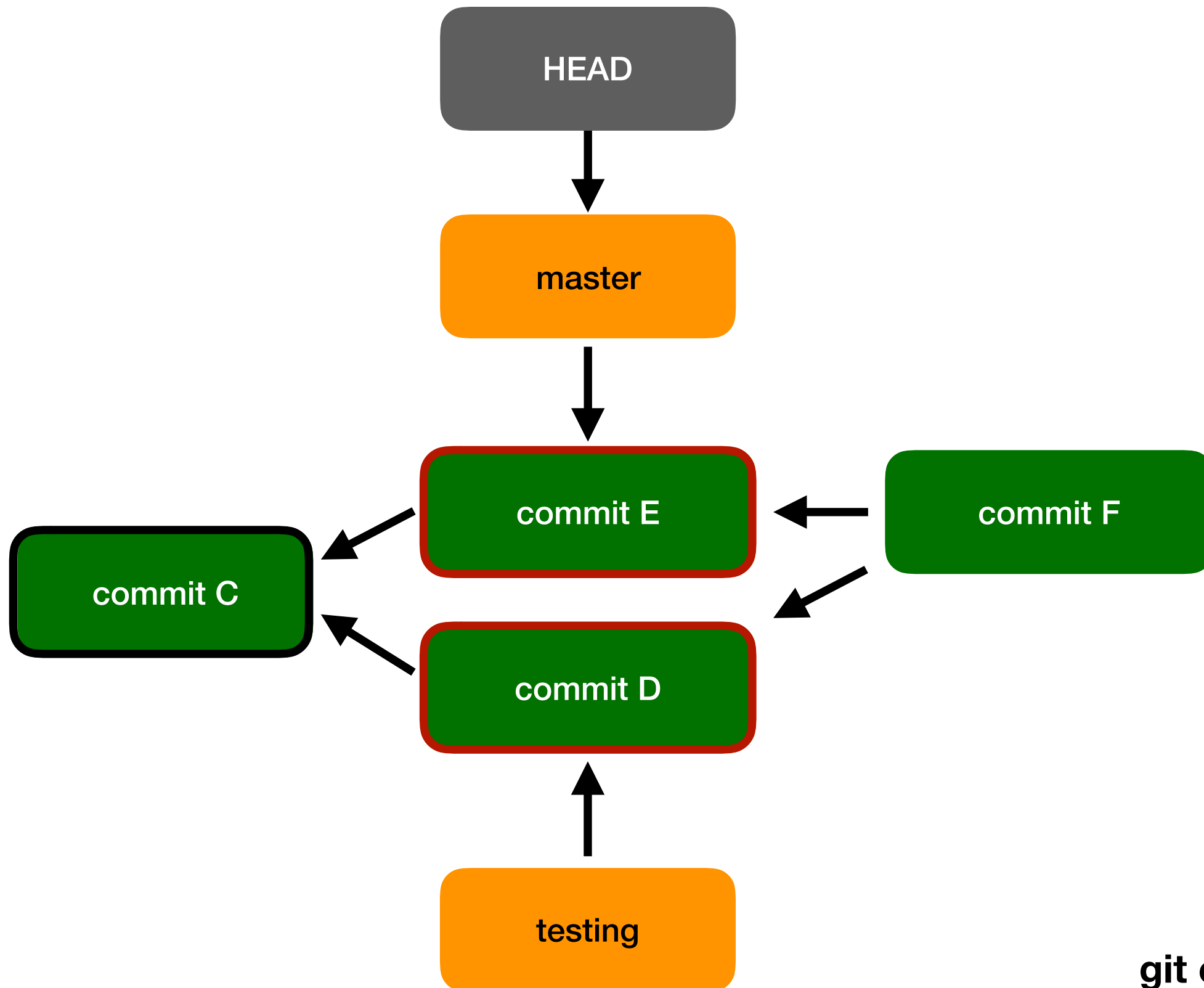
Atividade

- Criar 4 branches para a história
 - Branch para integrar a história (integracao-#grupo)
 - Branch para cada usuário (RA)

Atividade

- Escrever cada um dos membros (1 parágrafo) em seu branch
 - Como o herói e o vilão se conheceram?
 - O confronto
 - O encerramento da história





git checkout master
git merge testing
git commit

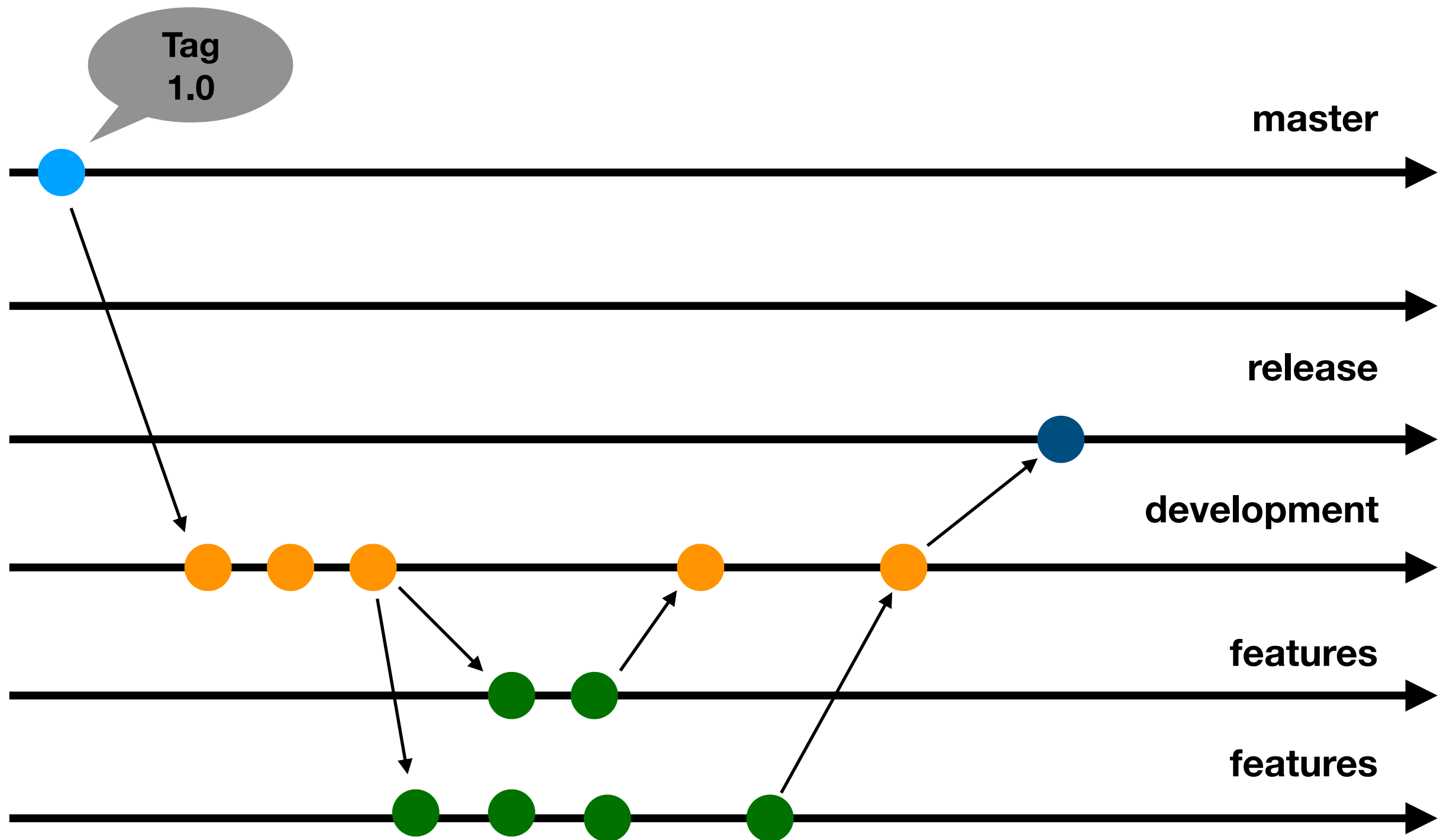
Atividade

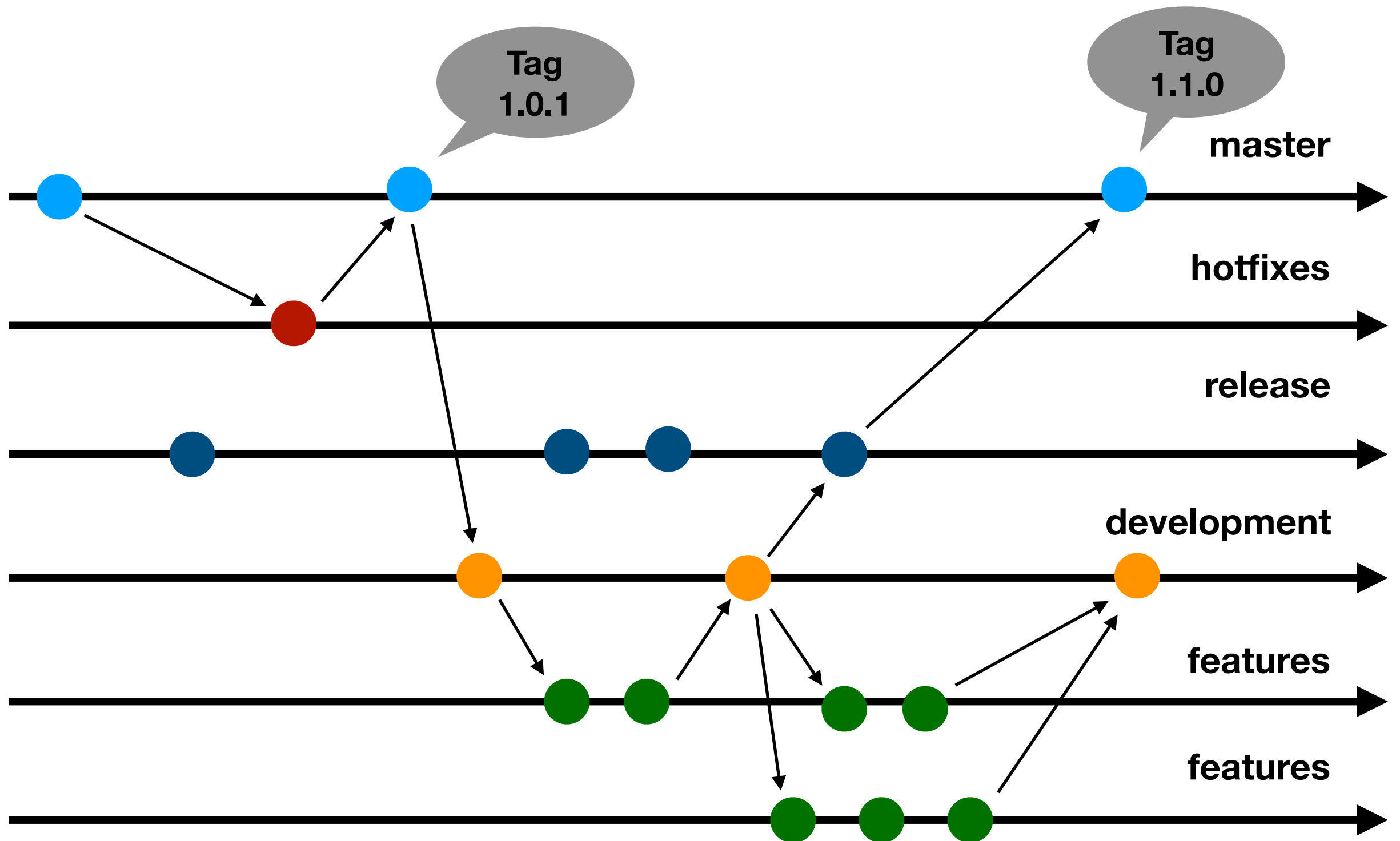
- Vamos fazer o merge dos branches?
- Gerar uma história integrada!

Política de branches

Política de branches

- Gerenciar paralelismo da equipe
- Gerência de configuração e versões
- Responsabilidades
- Controle de qualidade





Comandos

- **git tag -l <wildcard (ex: 1.1.*)>**
 - lista as tags (que sigam o padrão, se definido)
- **git tag -a <TAG> -m <mensagem>**
 - Cria uma tag com o nome TAG e mensagem associada

Comandos

- **git show <TAG>**
 - informações da TAG
- **git checkout <TAG>**
 - faz o checkout do commit associado com esta TAG

Pull Request

Pull Request

- Notifica outros de mudanças feitas em um repositório
 - Discutir alterações e rever as mudanças
 - Só então merge é feito na branch destino

Pull Request

- GitHub - Regras de proteção de branches
 - Baseado em padrão de nomes
 - Permissões
 - Aprovações

Options

Collaborators

Branches

Webhooks

Integrations & services

Deploy keys

Moderation

Interaction limits

Branch protection rule

Apply rule to

history

Rule settings

Protect matching branches

Disables force-pushes to all matching branches and prevents them from being deleted.

☐ **Require pull request reviews before merging**

When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.

☐ **Require status checks to pass before merging**

Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☐ **Require signed commits**

Commits pushed to matching branches must have verified signatures.

☐ **Include administrators**

Enforce all configured restrictions for administrators.

Create

Workshop sobre Git - Semana Integrada PUCCAMP

3 commits

3 branches

0 releases

1 contributor

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).




base: **historias** ▼



compare: **145643** ▼

There isn't anything to compare.

historias is up to date with all commits from **145643**. Try [switching the base](#) for your comparison.

 Showing **0 changed files** with **0 additions** and **0 deletions**.

Unified

Split

No commit comments for this range

Stash

Stash

- Trabalho incompleto em uma branch
- Necessidade de mudar para outra branch
- Stash
 - Empilha as mudanças feitas em uma branch
 - Permite fazer checkout em outra branch e trabalhar nela
 - Voltar ao trabalho anterior

Comandos

- **git stash**

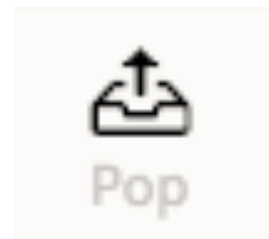


- Empilha os trabalhos em andamento e volta ao último commit

- **git stash list**

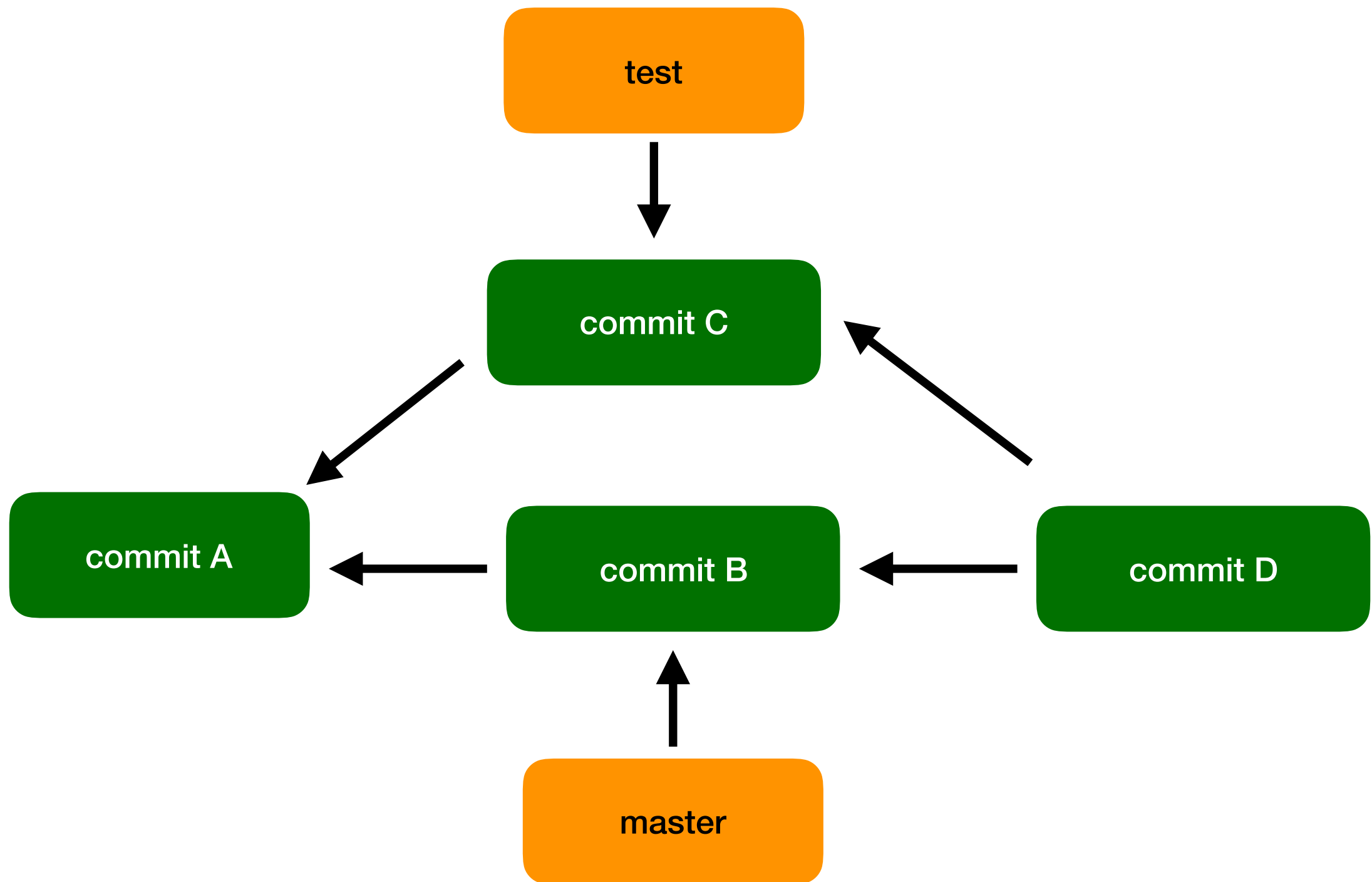
- Mostra a pilha de mudanças pendentes

- **git stash apply <stash@{n}>**



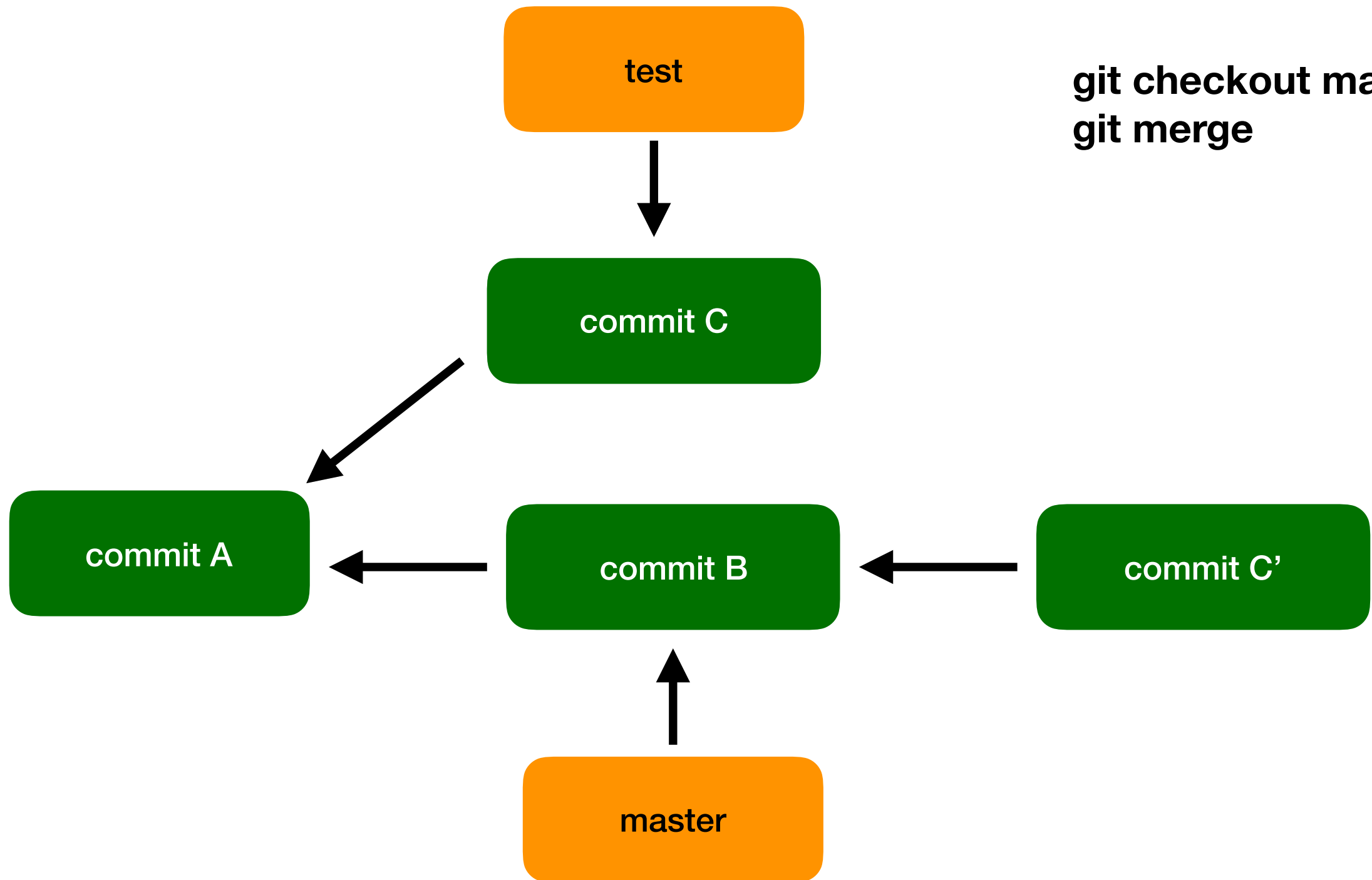
- Aplica a última mudança ou a (n-ésima) caso seja passado este parâmetro (topo = 0)

Rebase



git checkout test
git rebase master

git checkout master
git merge



Rebase

- Faz um *patch* das mudanças no outro branch
- Lineariza a história do código
 - Mesmo desenvolvido em paralelo
 - Mais simples de acompanhar
 - Inclusão mais limpa em um branch remoto.

Obrigado!