Minicurso – Git e GitHub

Victor Machado da Silva, MSc
victor.silva@professores.ibmec.edu.br

ibmec

Apresentação do curso

Aula 1: Introdução
- Apresentação
- O que é versionamento?
- Breve história do Git
- Primeiros conceitos
- Instalação e configuração do Git
- Criação de conta no GitHub
- Criando o primeiro repositório!

Aula 2: Conceitos básicos
- Sobre controle de versão
- Áreas de trabalho: diretório de trabalho, área de preparação, repositório
- Comandos essenciais: `init`, `add`, `commit`
- Criando o primeiro repositório local

Aula 3: Branches
- Introdução sobre branches
- Criando e alternando entre branches
- Comandos `branch`, `checkout`, `merge`
- Resolvendo conflitos de merge de forma básica
- Criando e mesclando branches simples

Aula 4: GitHub
- Sobre repositórios remotos
- Conectando um repositório local a um remoto
- Clonando repositórios remotos
- Comandos `push` e `pull`
- Clonando um repo, alterando e enviando de volta

Aula 5: Colaboração
- Pull Requests
- Criando um fork de um repositório
- Criando um PR e solicitando revisão
- Fusão de PRs
- Criando um PR e colaborando com colegas

Aula 6: Conflitos
- Sobre resolução de conflitos
- Causas comuns de conflitos
- Utilização de ferramentas para resolver conflitos
- Atividade prática: simulação de conflitos e suas resoluções

Aula 7: GitHub Issues
- Utilização do Issues para gerenciar tarefas
- Organização de um projeto com o GitHub Projects
- Labels e Milestones
- Criando issues e organizando um projeto

Aula 8: Git Ignore
- O que é e como usar o arquivo `.gitignore`
- Boas práticas de organização de repositórios
- Atividade prática: criando um arquivo `.gitignore` e organizando um repositório

Aula 9: Git Rebase
- Introdução ao `git rebase`
- Alterando o histórico de commits
- Comando `git cherry-pick`
- Atividade prática: reorganizando commits com `rebase` e `cherry-pick`

Aula 10: Git Flow
- Introdução a fluxos de trabalho como GitFlow e GitHub Flow
- Boas práticas para colaboração em projetos maiores
- Próximos passos na jornada do Git

Clique nas aulas para ir direto para o conteúdo!

Apresentação - Links interessantes

- Link para download do Git
- Documentação oficial
- Jogo “Learn git branching”
- Curso sobre versionamento no YouTube
- Vídeo avançado sobre estratégias de branching

Aula 1: Introdução

Introdução

Talvez você reconheça esse nome por causa de sites como *GitHub* ou *GitLab*. **Git** é um sistema *open-source* de controle de versionamento.

Versionar projetos é uma prática essencial no mundo profissional e, em particular, na área de tecnologia, para manter um **histórico de modificações** deles, ou poder reverter alguma modificação que possa ter comprometido o projeto inteiro, dentre outras funcionalidades que serão aprendidas na prática.

Os projetos são normalmente armazenados em **repositórios**.

O que é Controle de Versões?

Controle de Versões é um sistema de grava mudanças aplicadas em um arquivo ou um conjunto de arquivos ao longo do tempo, para que o usuário possa relembrar ou recuperar depois.

Na área de tecnologia o controle de versões já é algo consolidado, uma vez que inúmeras alterações são aplicadas em um software durante a sua implementação e manutenção. No entanto, adotar um sistema de controle de versões (VCS, da sigla em inglês) é algo recomendado para qualquer área.

Quando se quer adotar um VCS, normalmente o primeiro passo é trabalhar com um controle local, fazendo cópias dos arquivos e os renomeando com algum padrão (p.ex., incluindo a data ao final do nome do arquivo).

O Git veio para otimizar esse controle de versões, permitindo inúmeras alterações que seriam muito complicadas se fossem feitas manualmente.

Uma breve história do Git

O Git foi criado por Linus Torvalds em 2005 para auxiliar no desenvolvimento do kernel Linux. É um sistema de controle de versão distribuído, o que significa que cada membro da equipe possui uma cópia completa do repositório, permitindo trabalhar offline e facilitando a colaboração.

O sistema foi projetado de forma a atender aos seguintes requisitos:

- Velocidade
- Design simples
- Suporte forte para desenvolvimento não-linear, com milhares de atividades sendo realizadas em paralelo
- Completamente distribuído, permitindo o acesso de qualquer lugar
- Eficiente no suporte a grandes projetos

Primeiros conceitos

Repositório: É o local onde todas as versões do seu projeto são armazenadas. Pode ser local ou remoto (ou ambos).

Commit: Uma snapshot (imagem instantânea) de todas as alterações feitas no código em um determinado momento. Cada commit tem uma mensagem que descreve as mudanças.

Branch: É uma linha de desenvolvimento separada que permite trabalhar em recursos diferentes sem interferir no código principal.

Merge: É o processo de combinar as alterações de um branch em outro. Quando você deseja incorporar as alterações feitas em um branch em outro, você realiza uma mesclagem.

Configuração inicial

Faça a instalação usual do Git na sua máquina, baixando-o pelo site oficial. Considere todas as opções recomendadas durante a instalação. É recomendado instalar o Git em um diretório fácil de acessar (p.ex., na raiz do diretório C:\).

Em seguida, abra o prompt de comando (ou powershell, gitbash ou outro programa de terminal) e entre com as seguintes opções de configuração:

```
git config --global user.name "Seu Nome"
git config --global user.email "seu@email.com"
git config --global user.username "seu_username"
git config --global core.editor "code --wait"
git config --global core.autocrlf false
```

Criando o primeiro repo no GitHub

O GitHub é uma plataforma de hospedagem de código-fonte e colaboração. Criar uma conta no GitHub é essencial para compartilhar e colaborar em projetos.

Após criar uma conta em https://github.com/, faça o login e siga com as instruções abaixo para criar o primeiro repositório remoto:

- No canto superior direito, clique no ícone "+" e selecione "New Repository".
- Dê um nome ao repositório, adicione uma descrição.
- Escolha a opção “public” e marque a opção “add a README file”.
- Clique em "Create repository" para criar o repositório remoto.

Aula 1 – Links interessantes

Introdução ao Versionamento de Código: Artigo

História e Conceitos do Git: Vídeo

Git e GitHub para Iniciantes: Artigo

Git e GitHub: Guia Rápido de Configuração: Vídeo

GitHub: Guia de Criação de Repositórios: Artigo

Aula 2: Conceitos básicos

O que é Git?

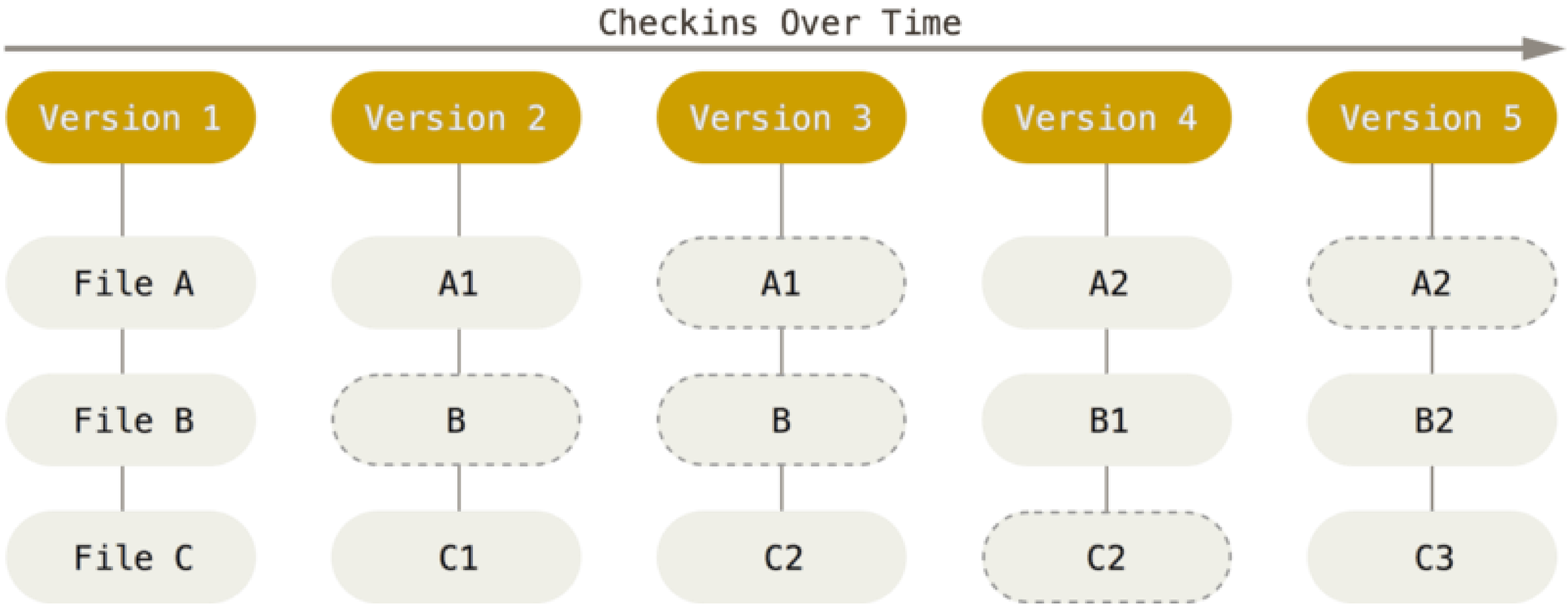
Git funciona pensando que os dados de um repositório compõem uma série de “fotografias” de um sistema de arquivos em miniatura.

No Git, a cada vez que você aplica uma alteração (ou *commit*), ou salva o estado do seu projeto, o Git basicamente tira uma “foto” de como os arquivos do repositório estão naquele momento e então ele armazena uma referência a essa foto.

Para ser eficiente, se os arquivos não foram alterados, o Git não altera os arquivos novamente, apenas um link para a última versão do arquivo armazenada.

Sendo assim, o Git trabalha os dados como um **fluxo de fotografias**.

O que é Git?


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

O que é Git?

A maior parte das operações no Git precisa apenas de arquivos e recursos locais para operarem, e normalmente nenhuma informação é necessária de outro computador na rede. Isso fornece uma velocidade de operação que outros VCS não possuem. Como cada usuário possui todo o histórico do projeto no computador, a maioria das operações aparenta ser quase instantânea.

Para todos os efeitos, na prática, o Git normalmente só acrescenta informações ao seu banco de dados, nunca removendo informações. É muito difícil, e não recomendado, gerar operações que removam informações, já que essas operações podem afetar o histórico do seu projeto.

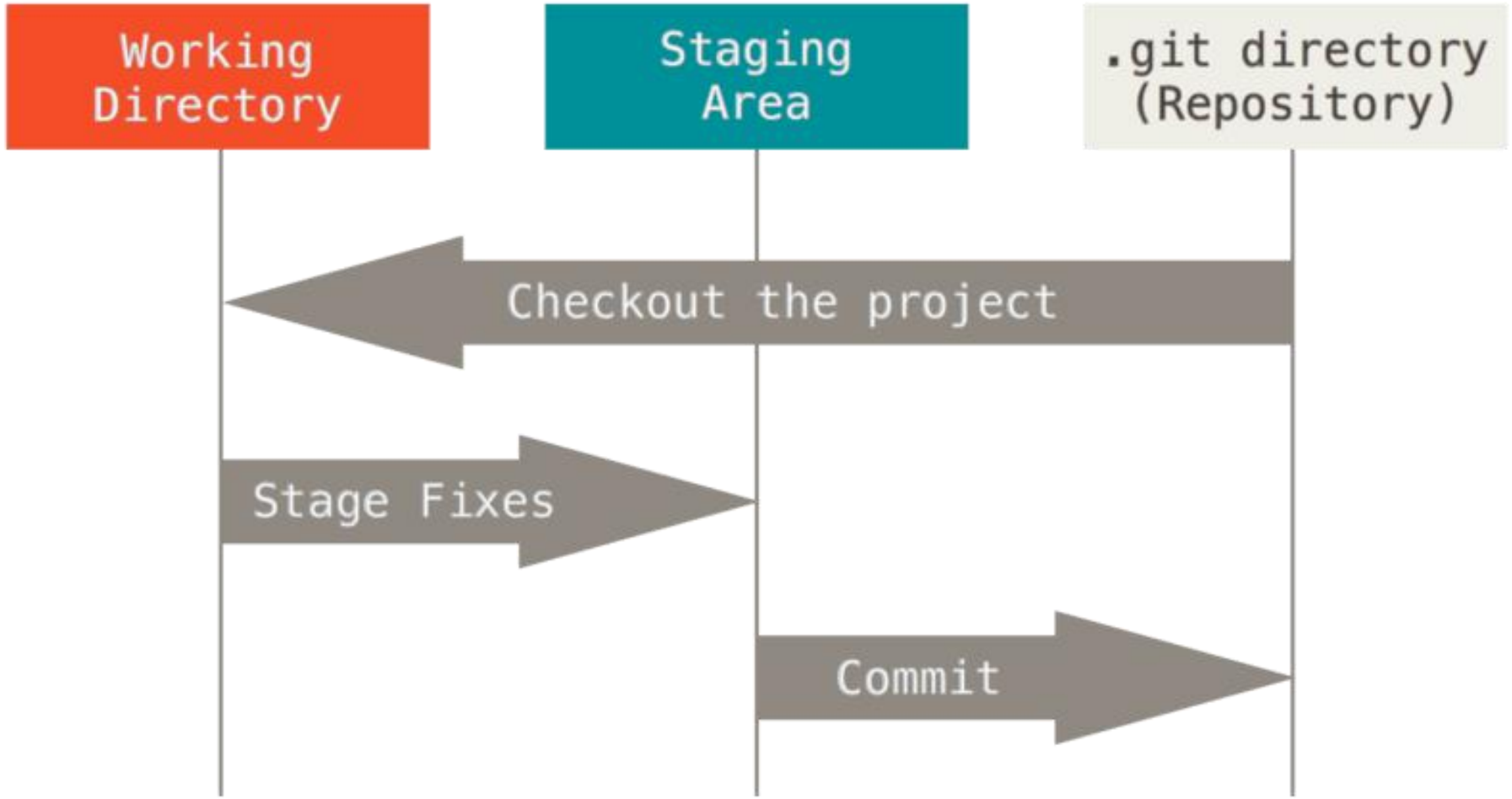
Os três estados

O Git tem três estados principais nos quais os arquivos de um repositório podem se encontrar: **modificado**, **preparado** e **“commitado”**:

- **Commitado** significa que os dados estão armazenados de forma segura em seu banco de dados local;
- **Modificado** significa que você alterou o arquivo, mas ainda não fez o commit no banco de dados;
- **Preparado** significa que você marcou a versão atual de um arquivo modificado para fazer parte do seu próximo commit.

Isso leva a três seções principais de um projeto Git: o diretório Git, o diretório de trabalho e área de preparo.

Os três estados



Comandos principais

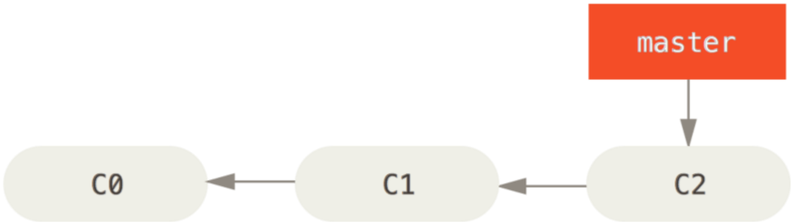
Comando	Descrição
git init	Inicializa um repositório git, sem nenhum commit
git status	Indica os status de arquivos modificados, adicionados ou removidos, além de arquivos preparados (staged)
git add <nome_arquivo>	Prepara o arquivo mencionado
git add -u	Prepara todos os arquivos modificados (porém não faz nada com arquivos novos)
git add .	Prepara todos os arquivos (incluindo arquivos novos)
git commit	Faz um commit dos arquivos preparados, sem uma mensagem de commit
git commit -m “mensagem”	Faz um commit dos arquivos preparados, incluindo a mensagem de commit definida
git restore <arquivo>	Desfaz modificações do arquivo que não foi preparado
git restore --staged <arquivo>	Desfaz a preparação do arquivo (porém mantém modificações)

Aula 3: Branches

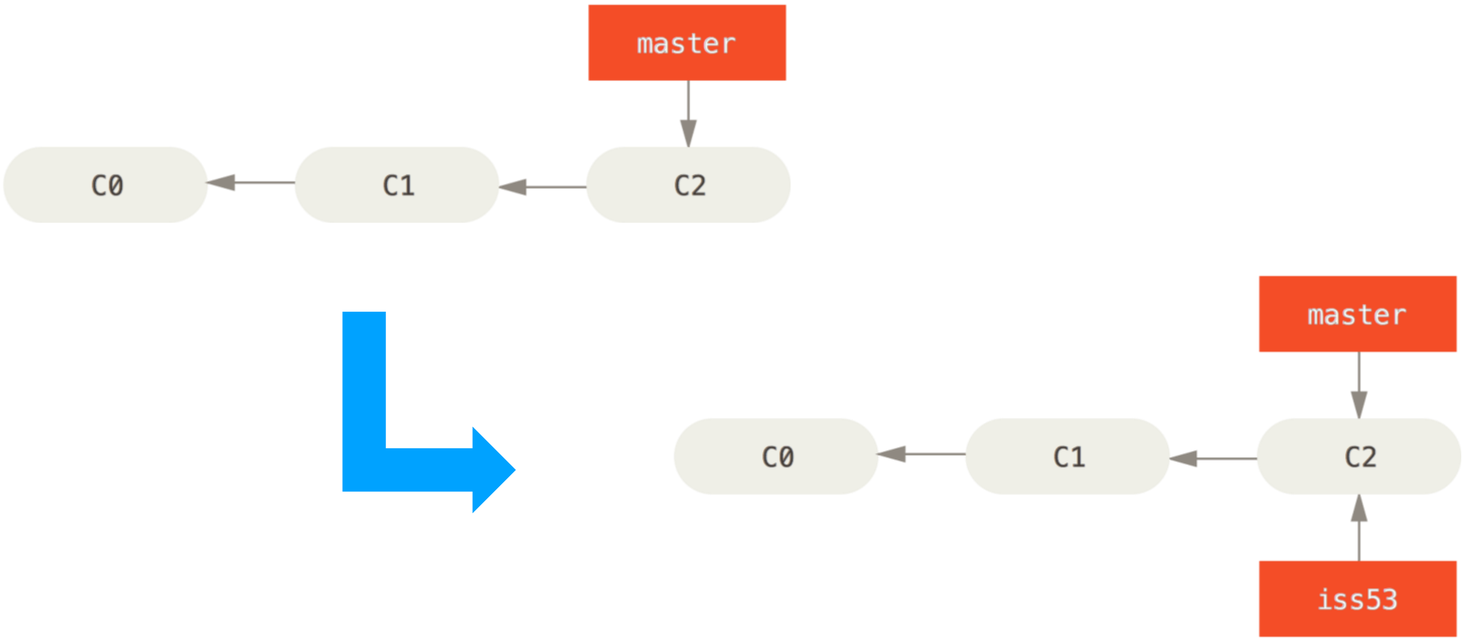
Entendendo os conceitos

Branches (ramificações) são uma característica fundamental no Git que permite criar caminhos de desenvolvimento separados. Cada branch representa uma linha independente de desenvolvimento dentro do repositório, permitindo que você trabalhe em diferentes recursos ou correções de bugs sem interferir no código principal. Isso é especialmente útil em projetos colaborativos, onde várias pessoas podem estar trabalhando em funcionalidades diferentes ao mesmo tempo.

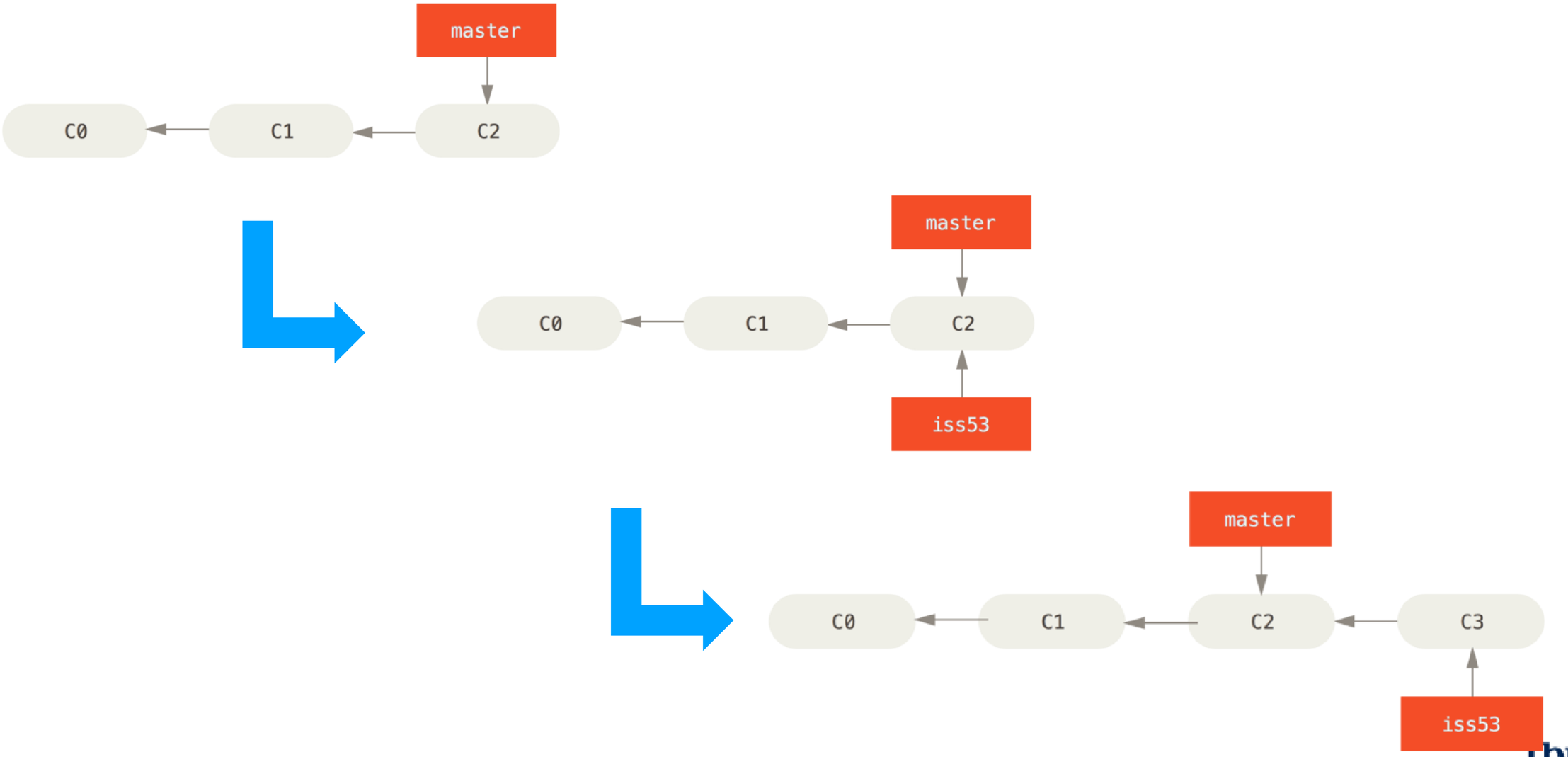
Branches no Git



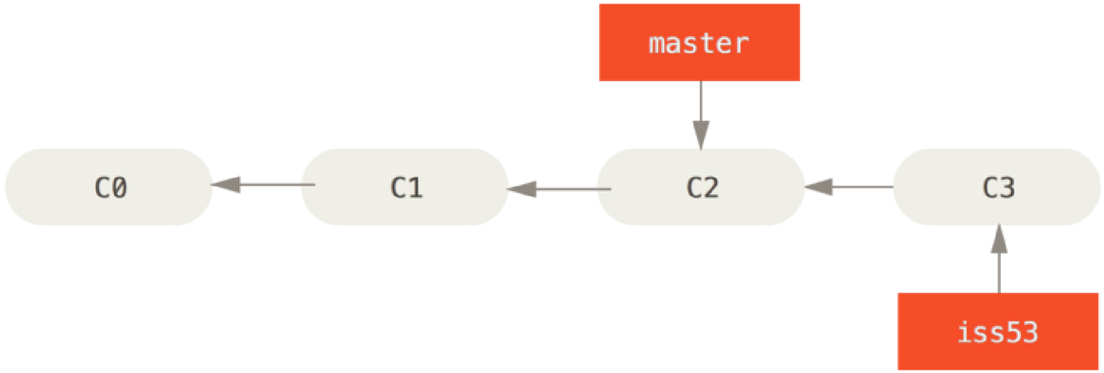
Branches no Git



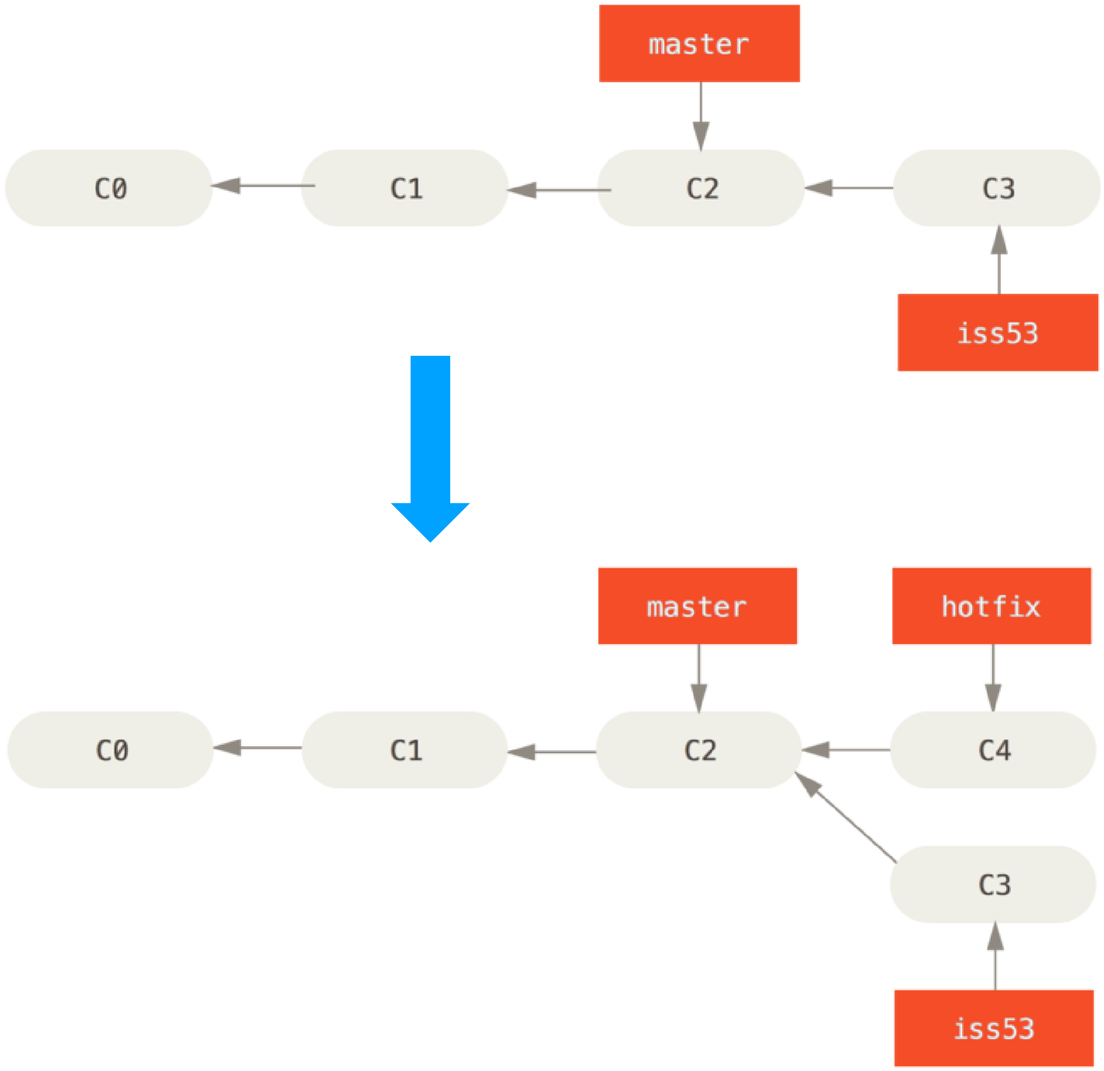
Branches no Git



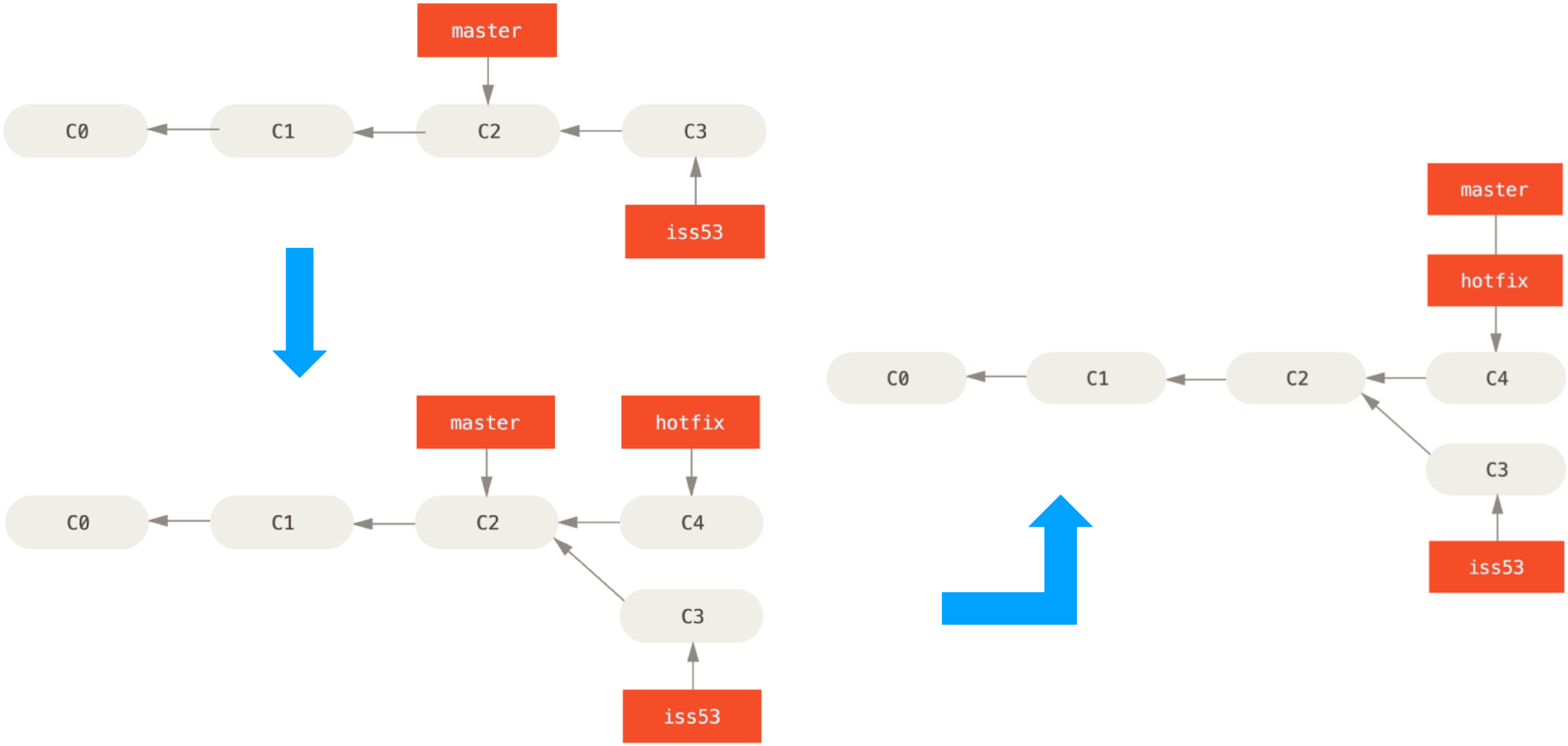
Branches no Git



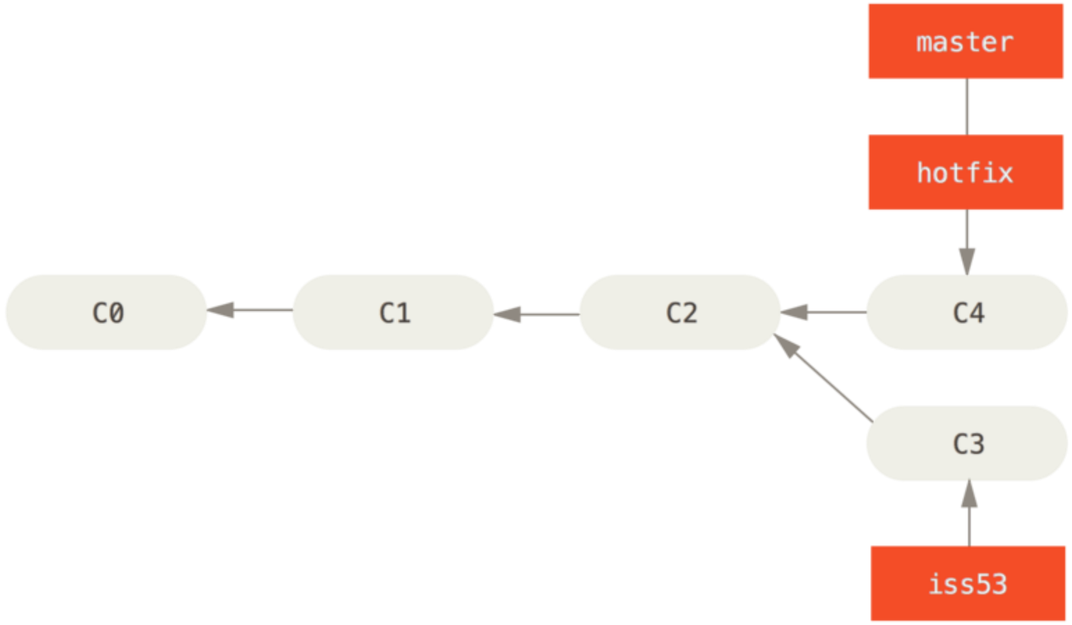
Branches no Git



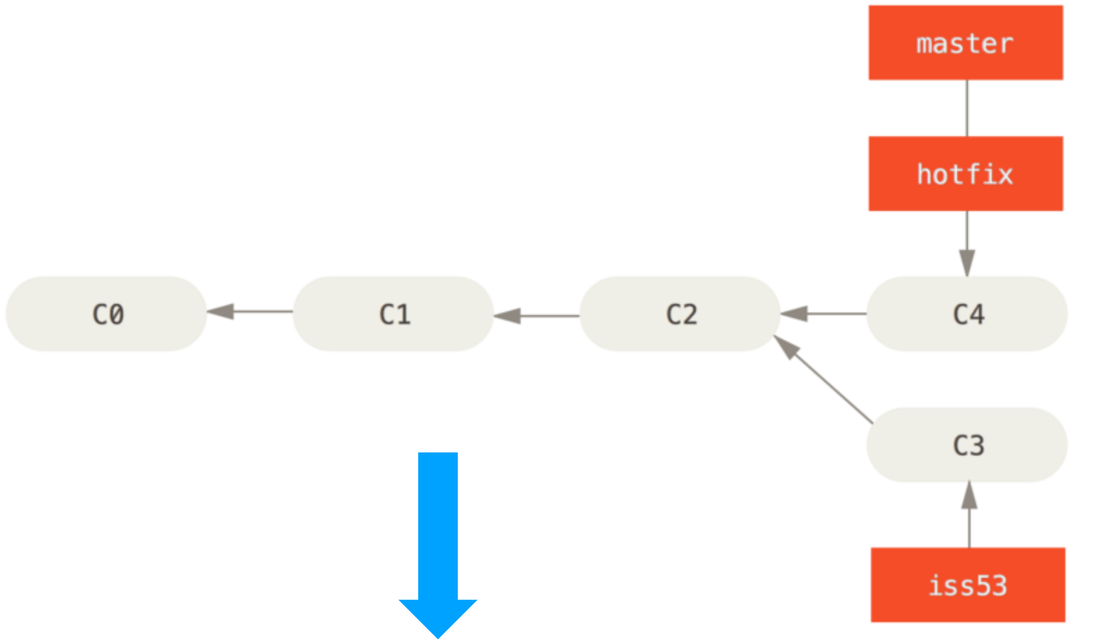
Branches no Git

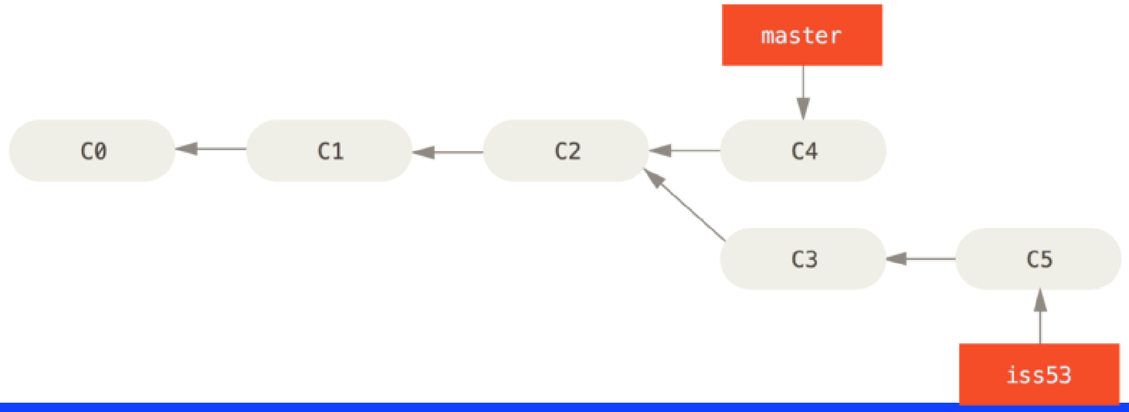


Branches no Git

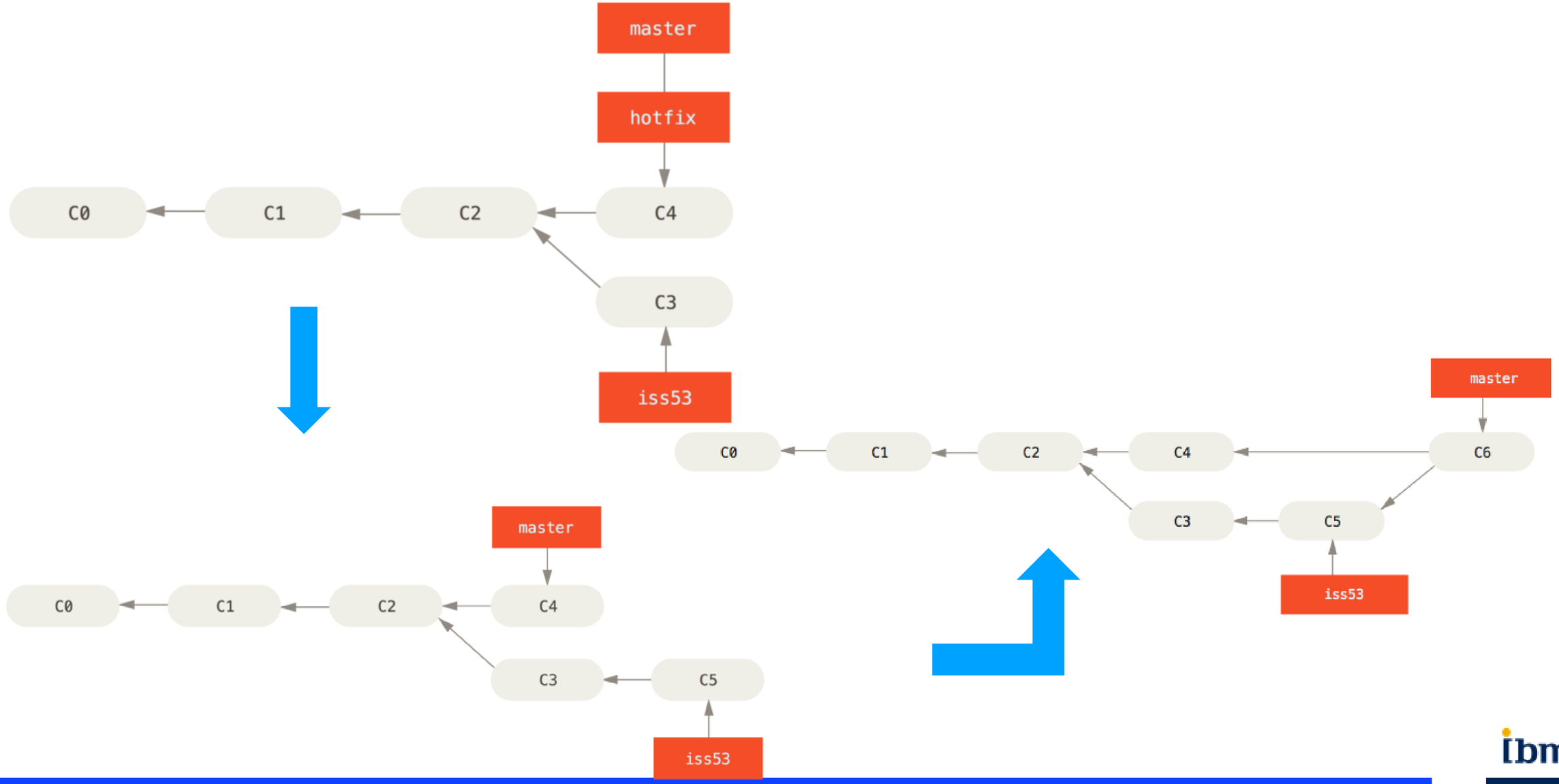


Branches no Git





Branches no Git



Criando e alternando entre branches

Comando	Descrição
git branch <nome_branch>	Cria um novo branch com o nome indicado
git branch	Lista todos os branches no repositório, destacando o branch atual com um asterisco (*)
git branch -d <nome_branch>	Exclui um branch que já foi mesclado a outro branch
git branch -D <nome_branch>	Força a exclusão de um branch, mesmo que ele tenha alterações não mescladas
git branch -m <antigo> <novo>	Renomeia um branch
git checkout <nome_branch>	Alterna entre branches. Move o HEAD (ponteiro atual) para um branch específico
git merge <nome_branch>	Combina as alterações do branch listado no branch atual

Resolvendo conflitos de merge

Conflitos de merge ocorrem quando o Git não consegue determinar automaticamente como combinar as alterações de dois branches. Isso pode acontecer quando você e outra pessoa fizeram alterações na mesma parte do código.

Para resolver os conflitos manualmente:

- O Git marcará as áreas com conflito no código-fonte com marcadores especiais.
- Você deve editar o código manualmente, removendo os marcadores e decidindo qual alteração manter.
- Após a resolução do conflito, você deve fazer um novo commit para finalizar o merge.

Fluxo de trabalho básico

- Crie um novo branch com `git branch nome-do-branch`.
- Alterne para o novo branch com `git checkout nome-do-branch`.
- Faça suas alterações no código.
- Faça um commit no novo branch.
- Volte para o branch principal (por exemplo, "master") com `git checkout master`.
- Use `git merge nome-do-branch` para mesclar as alterações do novo branch no branch principal.
- Resolva conflitos, se houver.
- Faça um novo commit após resolver conflitos.
- O novo branch pode ser excluído após a mesclagem, se desejado, usando `git branch -d nome-do-branch`.

IBMEC.BR

/IBMEC

IBMEC

@IBMEC_OFICIAL

@IBMEC

Ibmec