Resumo Git e GitHub

Regis Pires Magalhães
regismagalhaes@ufc.br

UFC

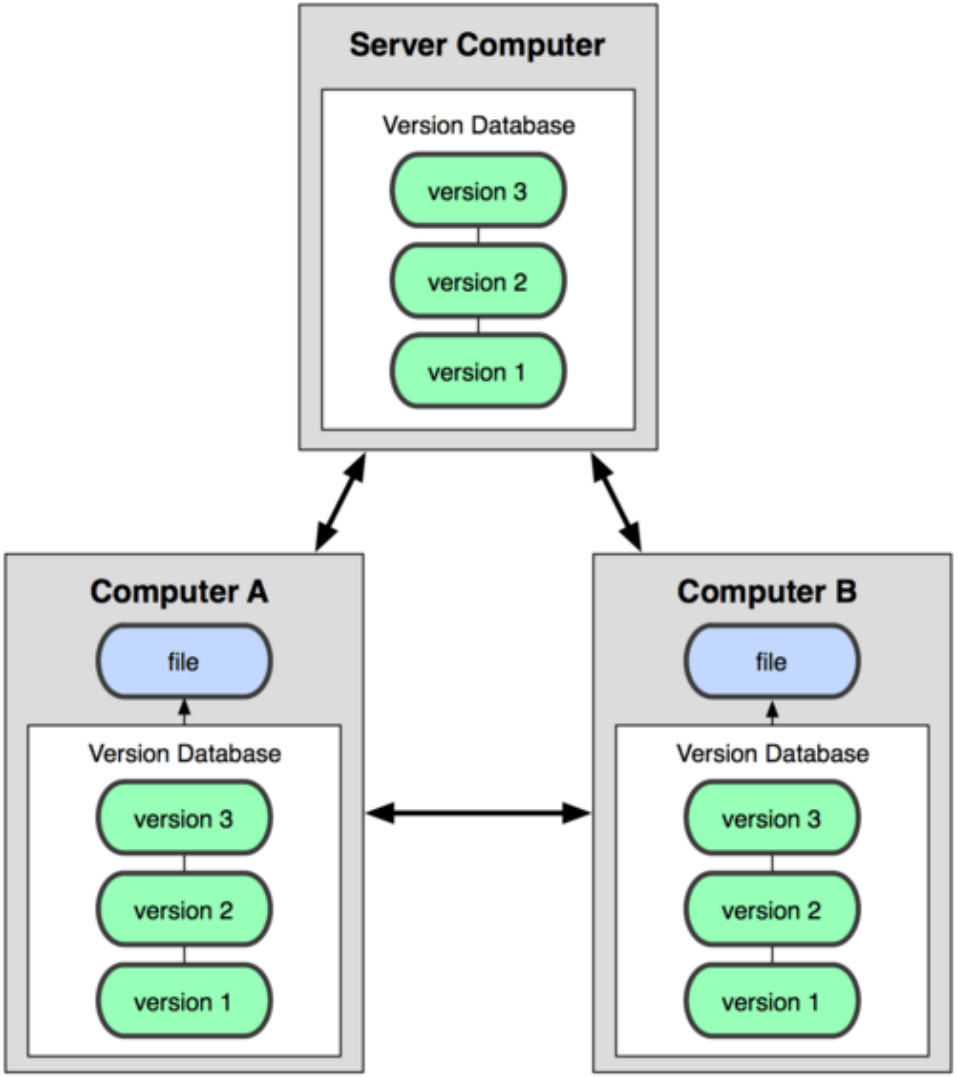
Sistema de Controle de Versão

- 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.
- Permite:
  - Reverter arquivos ou mesmo um projeto inteiro para um estado anterior.
  - Comparar mudanças feitas ao decorrer do tempo.
  - Ver quem foi o último a modificar algo que pode estar causando problemas, quem introduziu um bug, etc.

Sistemas de Controle de Versão Distribuídos

- Exemplos: Git, Mercurial, Bazaar e Darcs
  - Os clientes não apenas fazem cópias das últimas versões dos arquivos: eles são cópias completas do repositório.
  - Se um servidor falha, qualquer um dos repositórios dos clientes pode ser copiado de volta para o servidor para restaurá-lo.
  - Cada checkout (resgate) é na prática um backup completo de todos os dados.
  - Muitos desses sistemas lidam muito bem com o aspecto de ter vários repositórios remotos com os quais eles podem colaborar.
  - Permite que você estabeleça diferentes tipos de fluxos de trabalho que não são possíveis em sistemas centralizados.

Sistemas de Controle de Versão Distribuídos



Git

- Concebido em 2005 por Linus Torvalds para gerenciar o código do Linux.
- Incrivelmente rápido.
- Tem todo o histórico do projeto no seu disco local.
  - A maior parte das operações parece ser quase instantânea.
- Quase todas operações são locais
  - Há poucas coisas que você não possa fazer caso esteja offline.
- Git tem integridade
  - Arquivos com checksum SHA-1.

Git

Speed

+

Simple

+

Large Projects

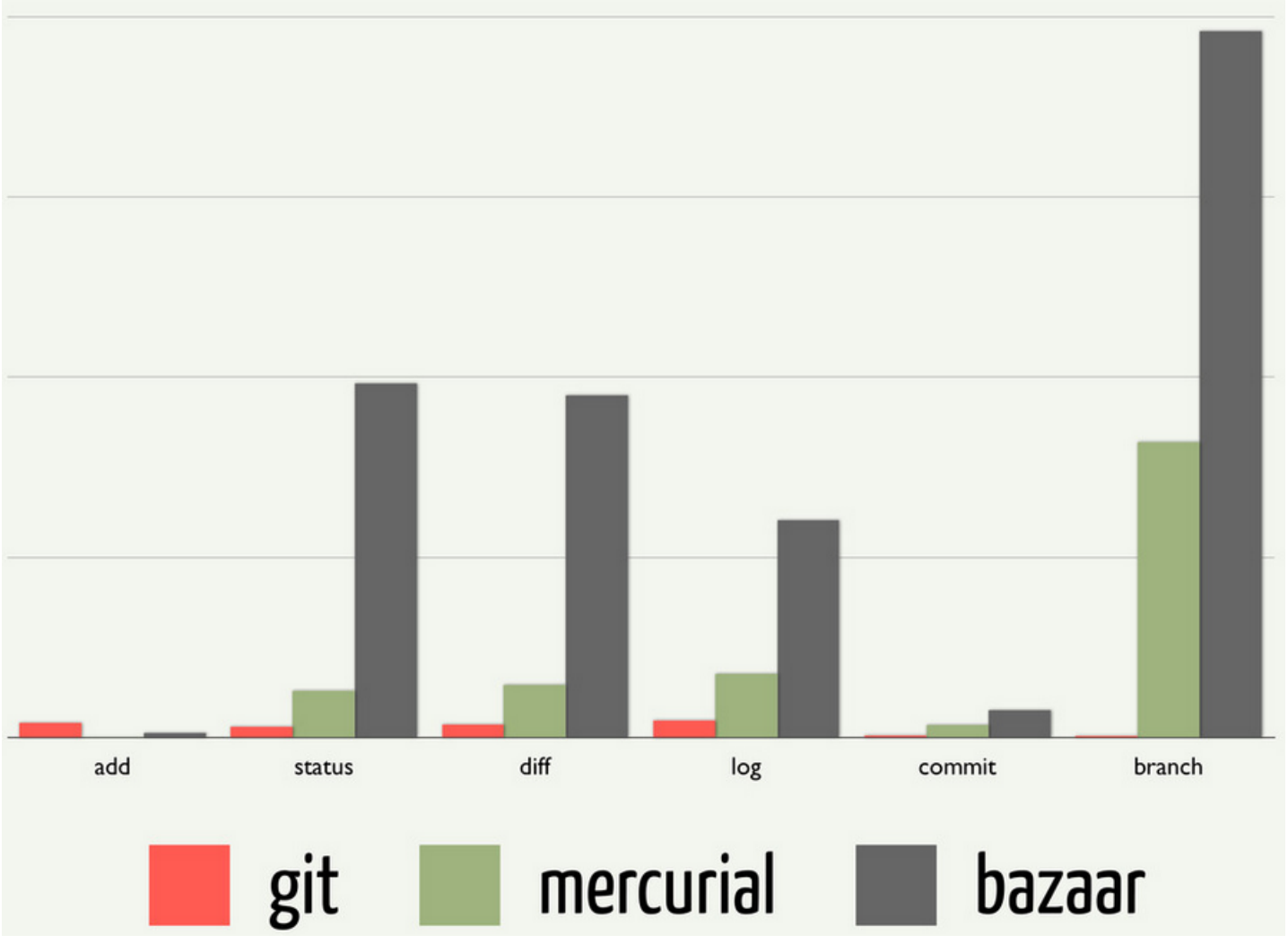
+

Fully Distributed

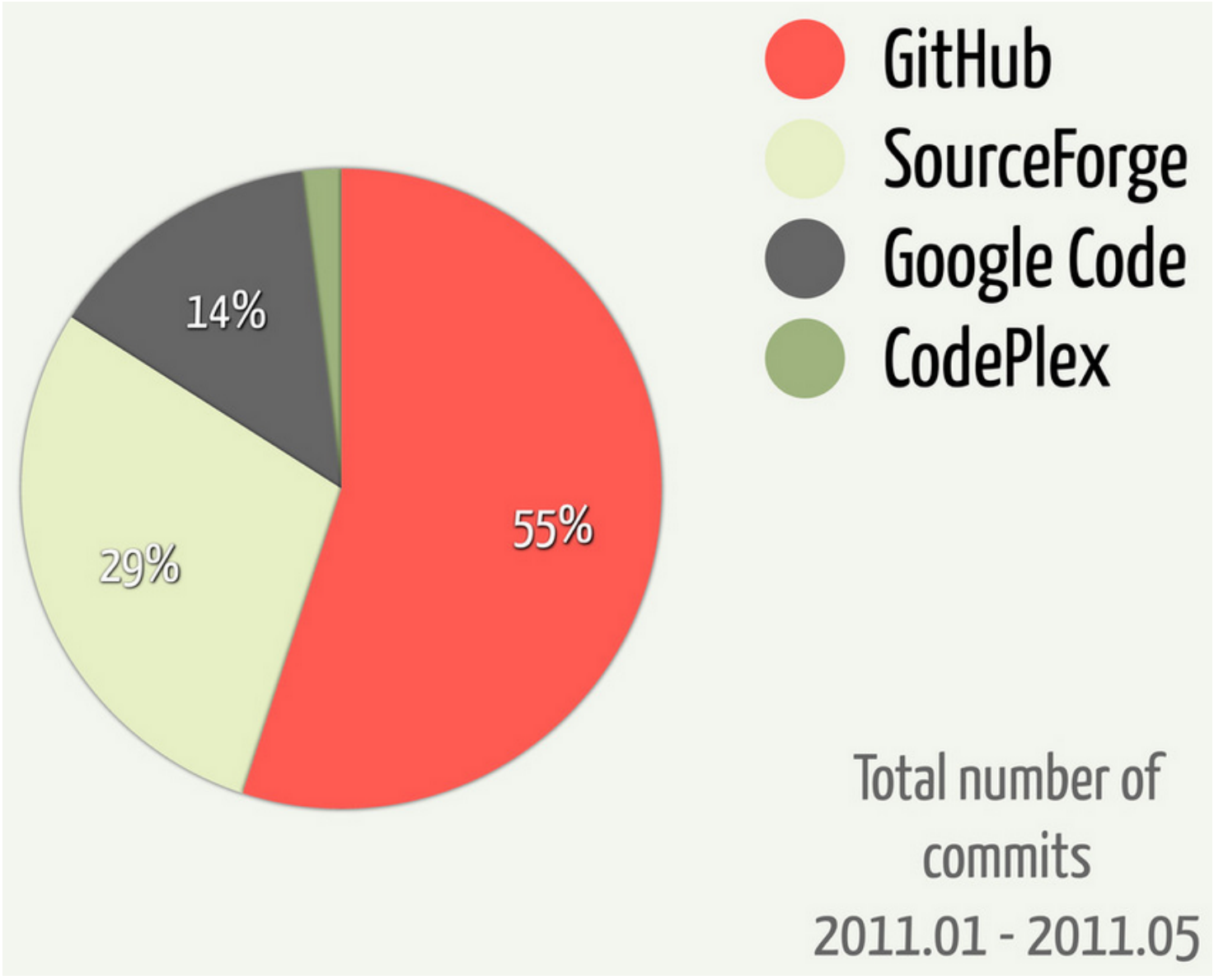
+

Non-linear Development

Git é rápido!!!



Git é muito usado

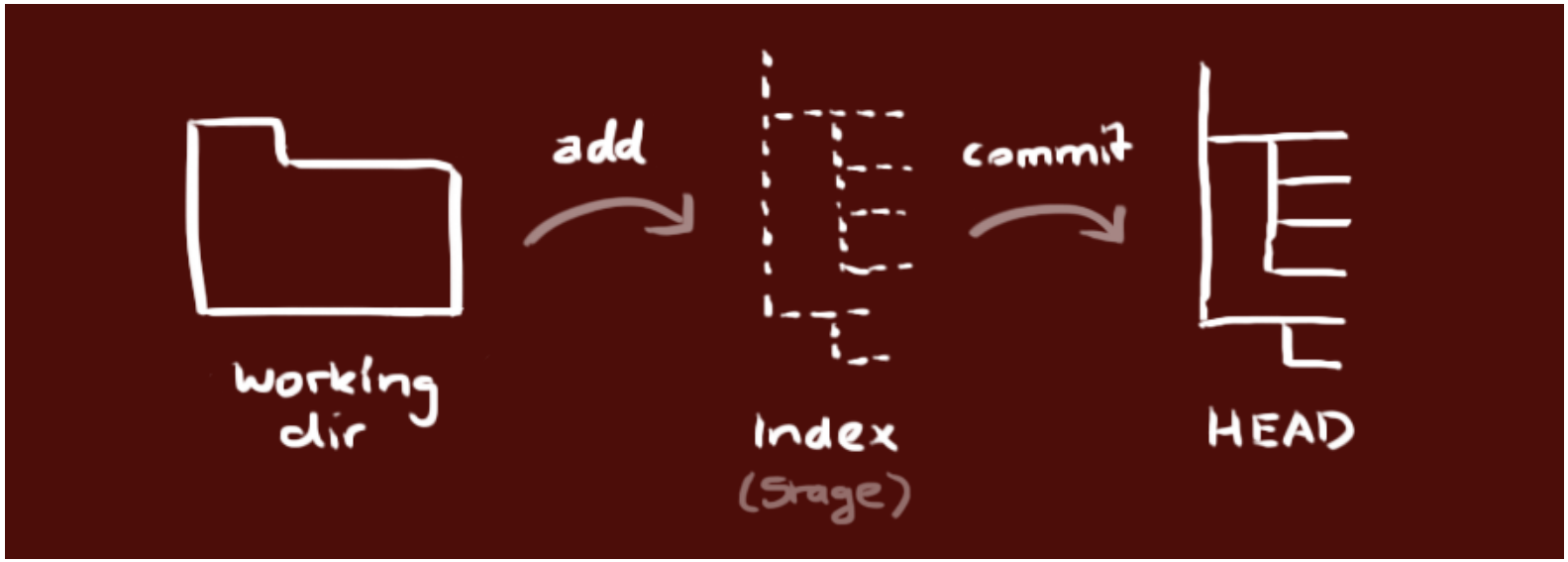


Os três estados

- Arquivos sempre estão em um dos três estados fundamentais:
  - modificado (modified)
    - arquivo que sofreu mudanças, mas que ainda não foi consolidado na base de dados.
  - preparado (staged)
    - quando você marca um arquivo modificado em sua versão corrente para que ele faça parte do snapshot do próximo commit (consolidação).
  - consolidado (committed)
    - Dados seguramente armazenados em sua base de dados local.

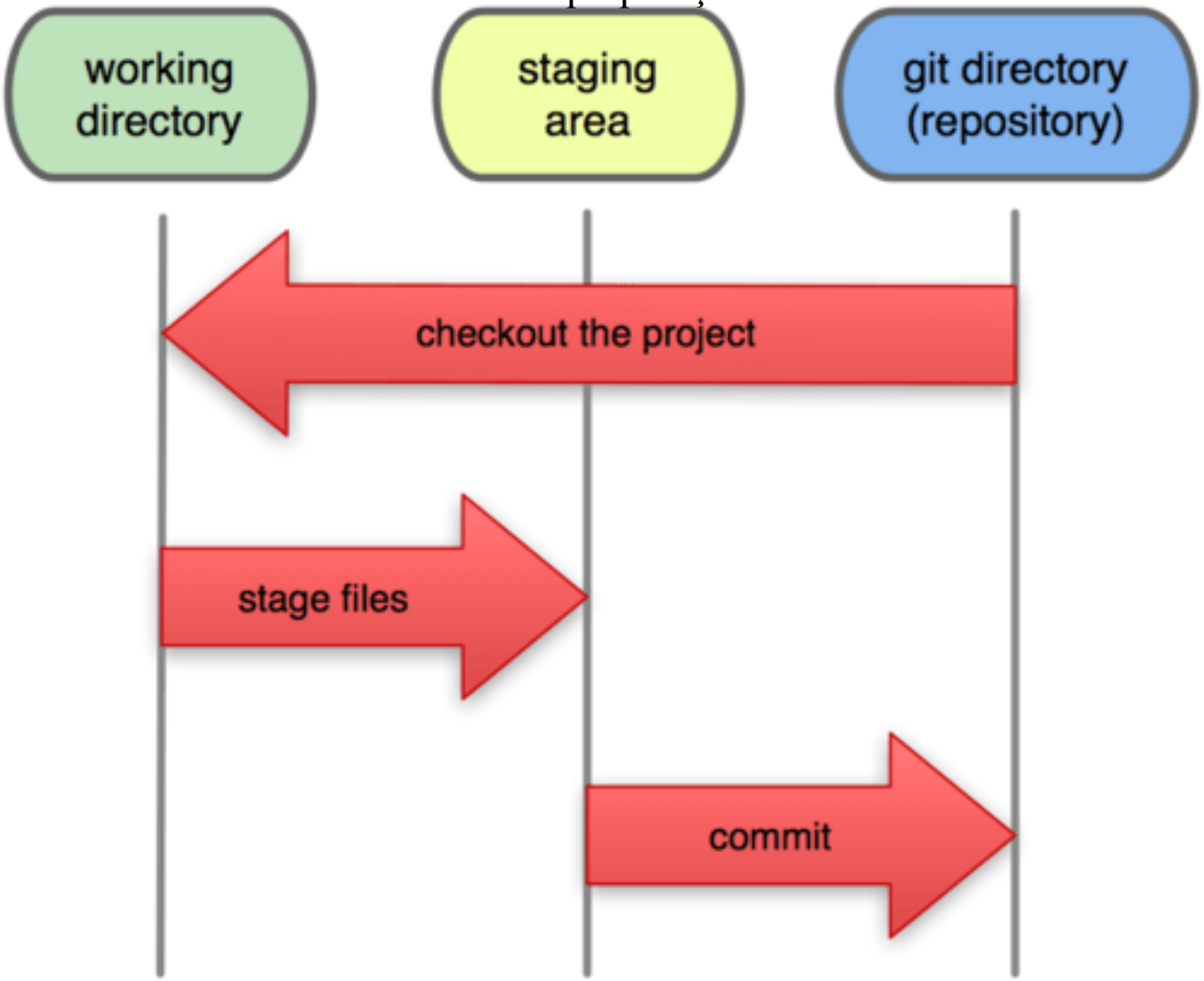
Repositório local

- Consiste em três "árvores" mantidas pelo git:
  - Working Directory
    - Contém os arquivos vigentes.
  - Index
    - Funciona como uma área temporária
  - HEAD
    - Aponta para o último *commit* (confirmação) que você fez
    - É uma referência para o último commit do branch atual



Local Operations

Área de preparação



Criar um novo repositório local

- `git init`

Criar um novo repositório a partir de um repositório já existente

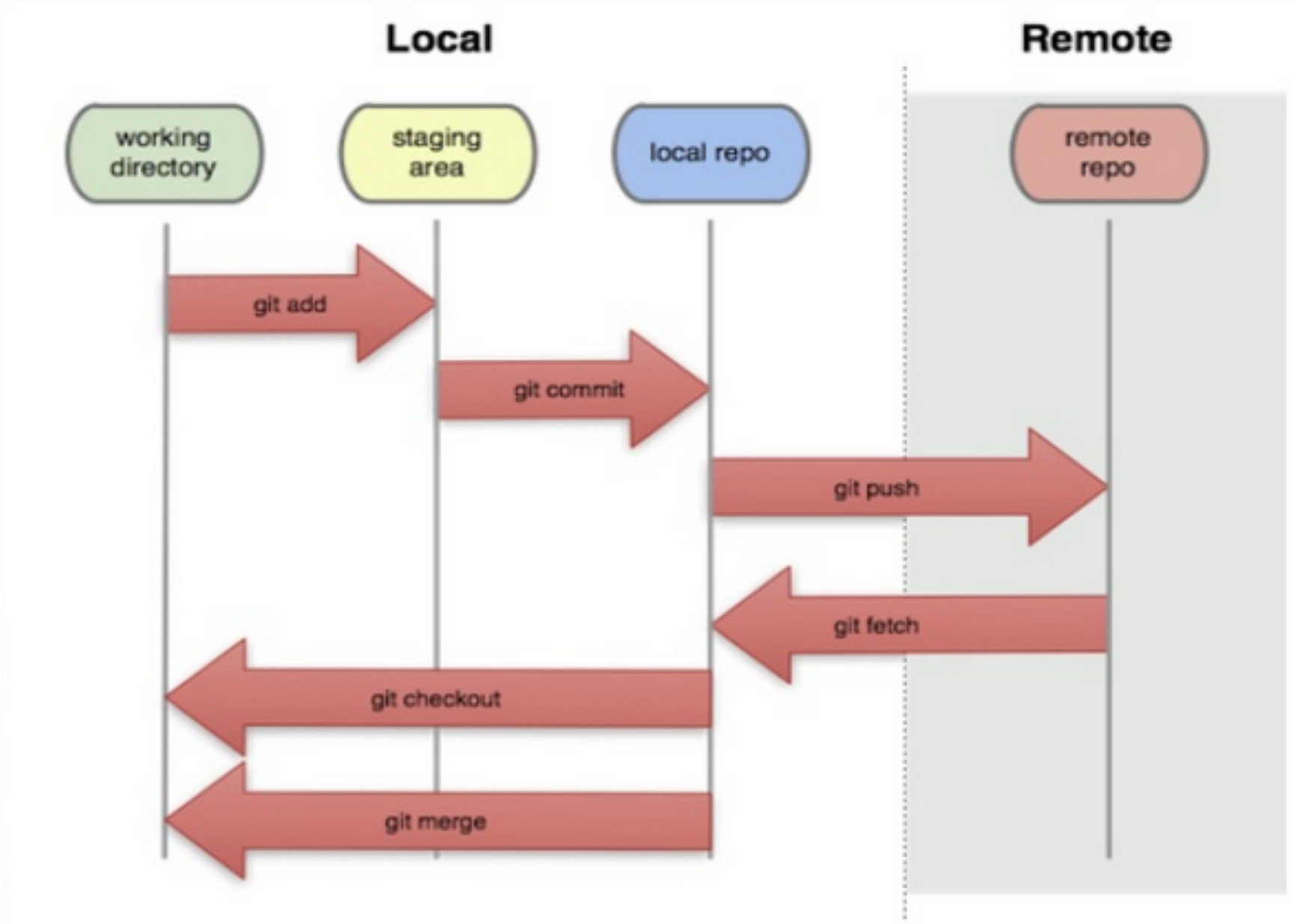
- Cópia de trabalho de um repositório local para outro repositório local
  - `git clone /caminho/para/o/repositório`
- Cópia de trabalho de um repositório remoto para um repositório local
  - `git clone usuário@servidor:/caminho/para/o/repositório`
  - Exemplo:
    - `git clone https://usuario@github.com/regispires/minicurso-git.git`
  - Se o usuário não for definido durante o clone, ele será requisitado sempre que for feita alguma operação remota que exija autenticação.

Conectando um repositório local a um repositório remoto

- Caso o repositório local não tenha sido criado a partir de um clone:
  - `git remote add origin <servidor>`

Alterando os dados de um repositório remoto já configurado

- `git remote set-url origin https://usuario@github.com/regispires/minicurso-git.git`


Local
Remote
working directory
staging area
local repo
remote repo
git add
git commit
git push
git fetch
git checkout
git merge

Adicionar e confirmar

- Você pode propor mudanças (adicioná-las ao **Index**) usando:
  - `git add <arquivo>`
  - `git add .`
  - `git add *`

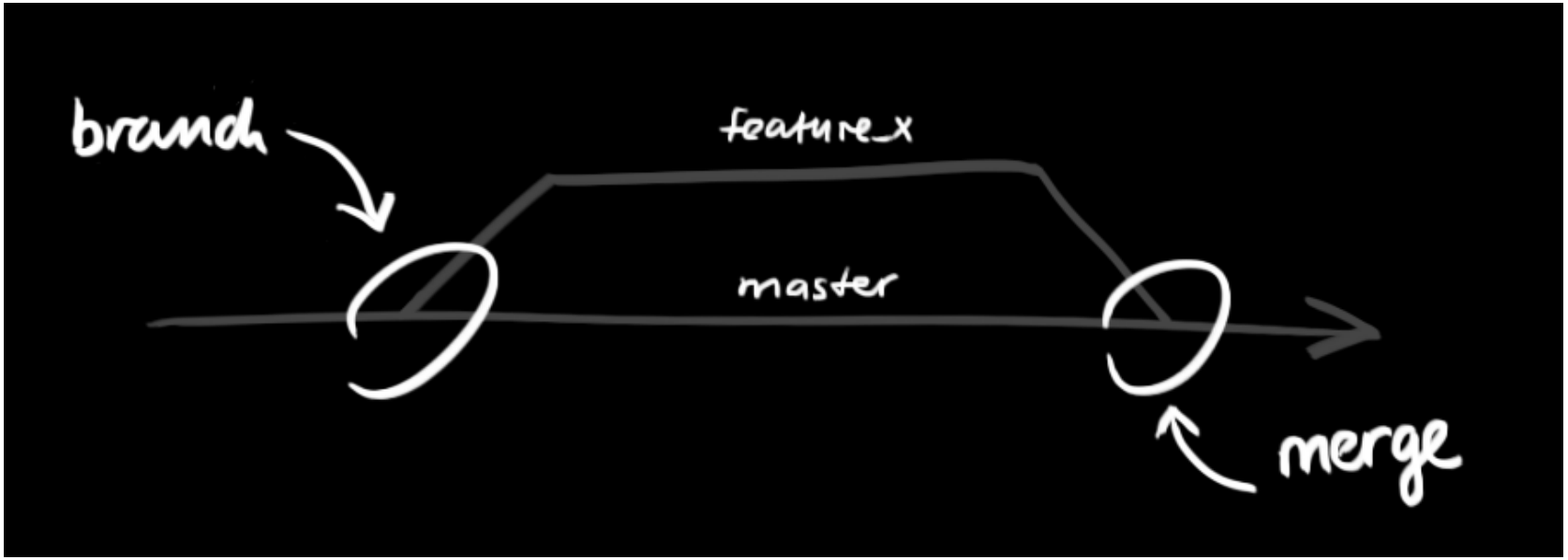
- Para confirmar (*commit*) estas mudanças, use:
  - `git commit -m "comentários das alterações"`
  - Agora o arquivo é enviado para o HEAD, mas ainda não para o repositório remoto.

Enviando alterações

- Alterações agora estão no HEAD da cópia de trabalho local.
- Para enviá-las ao repositório remoto:
  - `git push origin master`
    - Repositório remoto: origin
    - Branch remoto: master

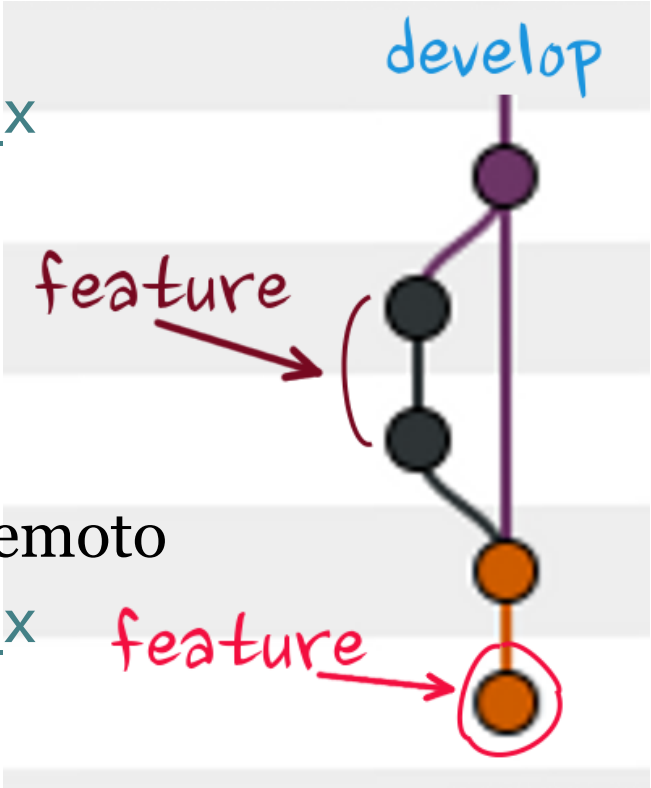
Ramificações - Branches

- Usados para desenvolver funcionalidades isoladas umas das outras.
- Branch master – branch padrão ao criar um repositório.
- Use branches durante o desenvolvimento e depois, mescle-os (merge) ao master.

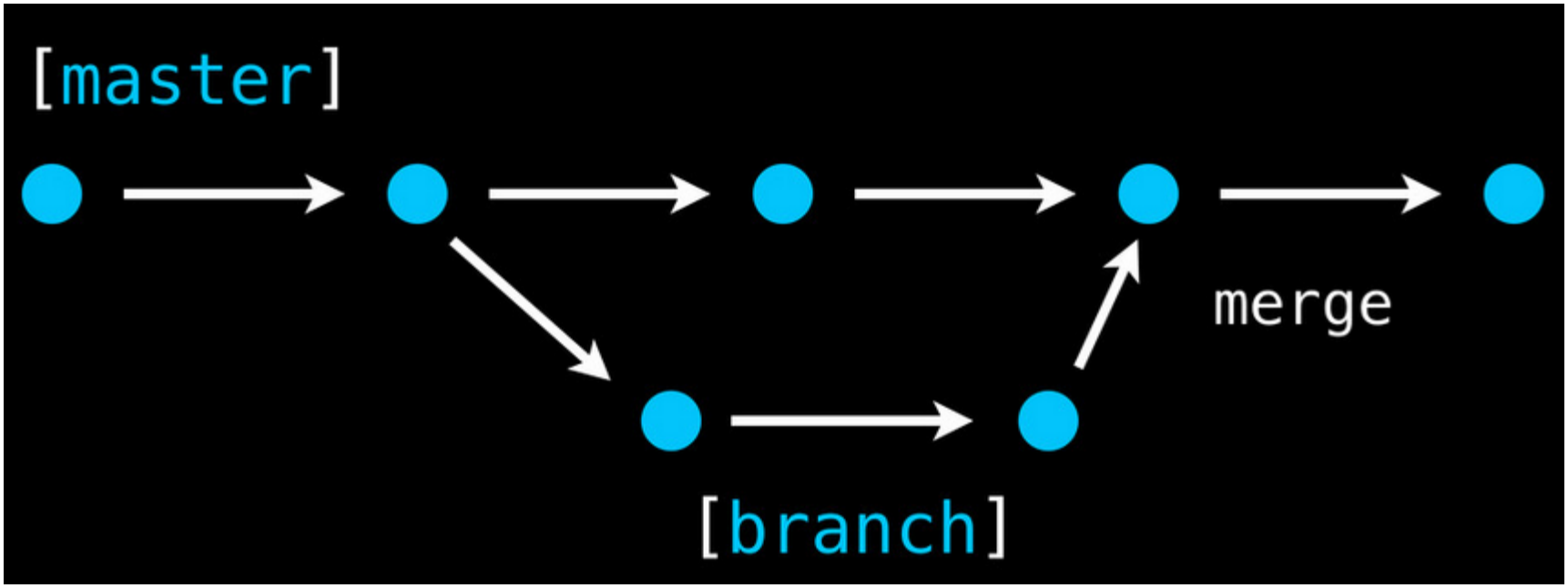


Ramificações - Branches

- Branch é uma ramificação no desenvolvimento.
- Uma técnica utilizada é fazer um branch para cada funcionalidade implementada
- Criar e selecionar um novo branch
  - `git checkout -b funcionalidade_x`
- Retornar ao branch master
  - `git checkout master`
- Remover o branch
  - `git branch -d funcionalidade_x`
- Enviando o novo branch ao repositório remoto
  - `git push origin funcionalidade_x`



Ramificações - Branches



Atualizar o repositório local

- Obter e mesclar (merge) alterações remotas
  - `git pull`
    - Faz um "`git fetch`" seguido de um "`git merge`"
  - `git pull --rebase`
    - Faz um "`git fetch`" seguido de um "`git rebase`"

Resolvendo conflitos

- Nem sempre é possível fazer o merge, pois podem existir conflitos.
- Em caso de conflitos, é preciso editar manualmente os arquivos:

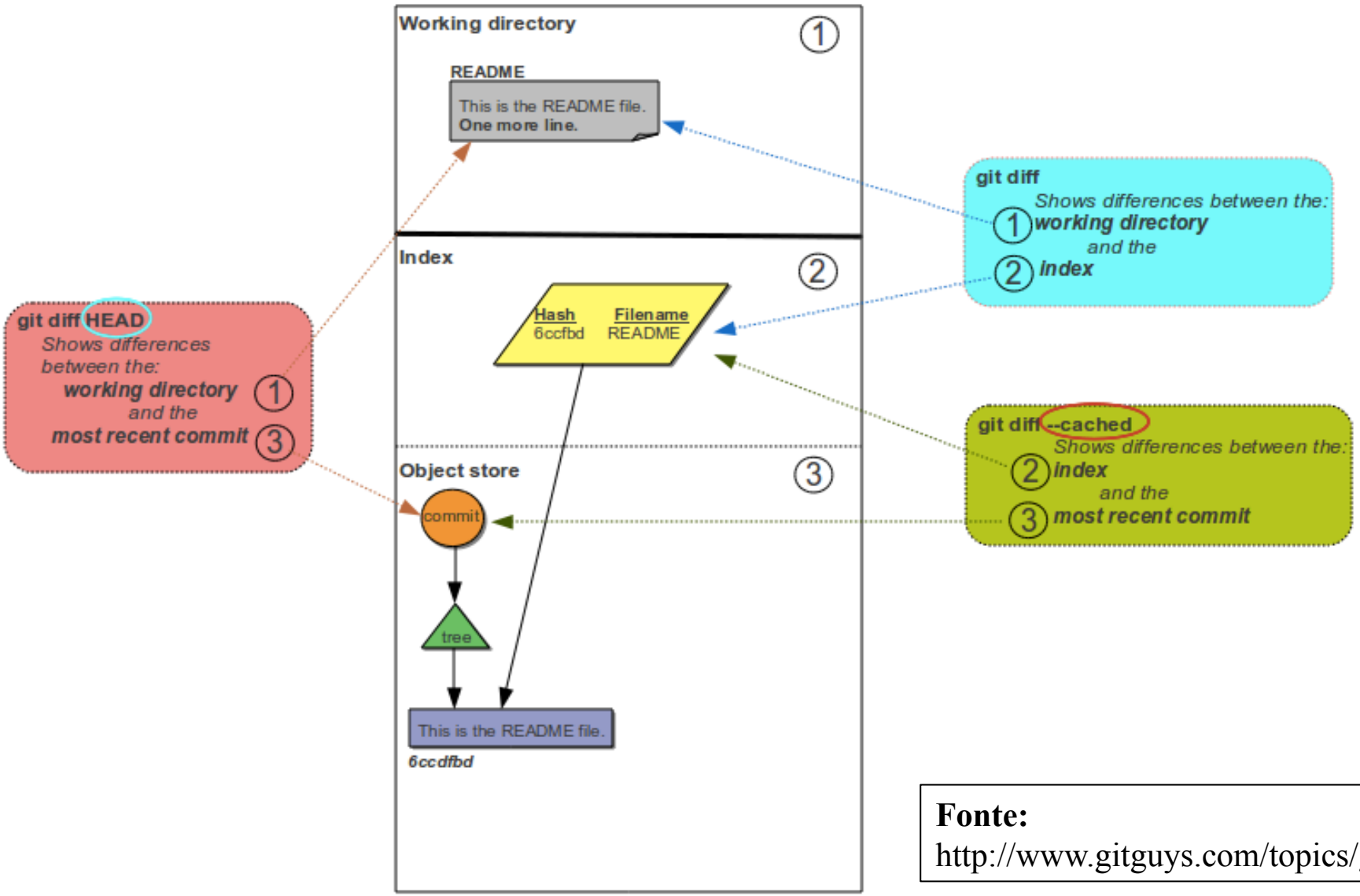
```
<<<<<<< HEAD
alteração remota
=======
alteração local
>>>>>>> modificação do arq1.txt
```

- Após a edição, é preciso marcar os arquivos como merged
  - `git add <arquivo>`

Visualizando diferenças entre branches

- `git diff <branch origem> <branch destino>`
- Bastante útil antes do merge.
- Exemplo:
  - `git diff master origin/master`

Visualizando diferenças entre branches



Fonte:
http://www.gitguys.com/topics/git-diff/

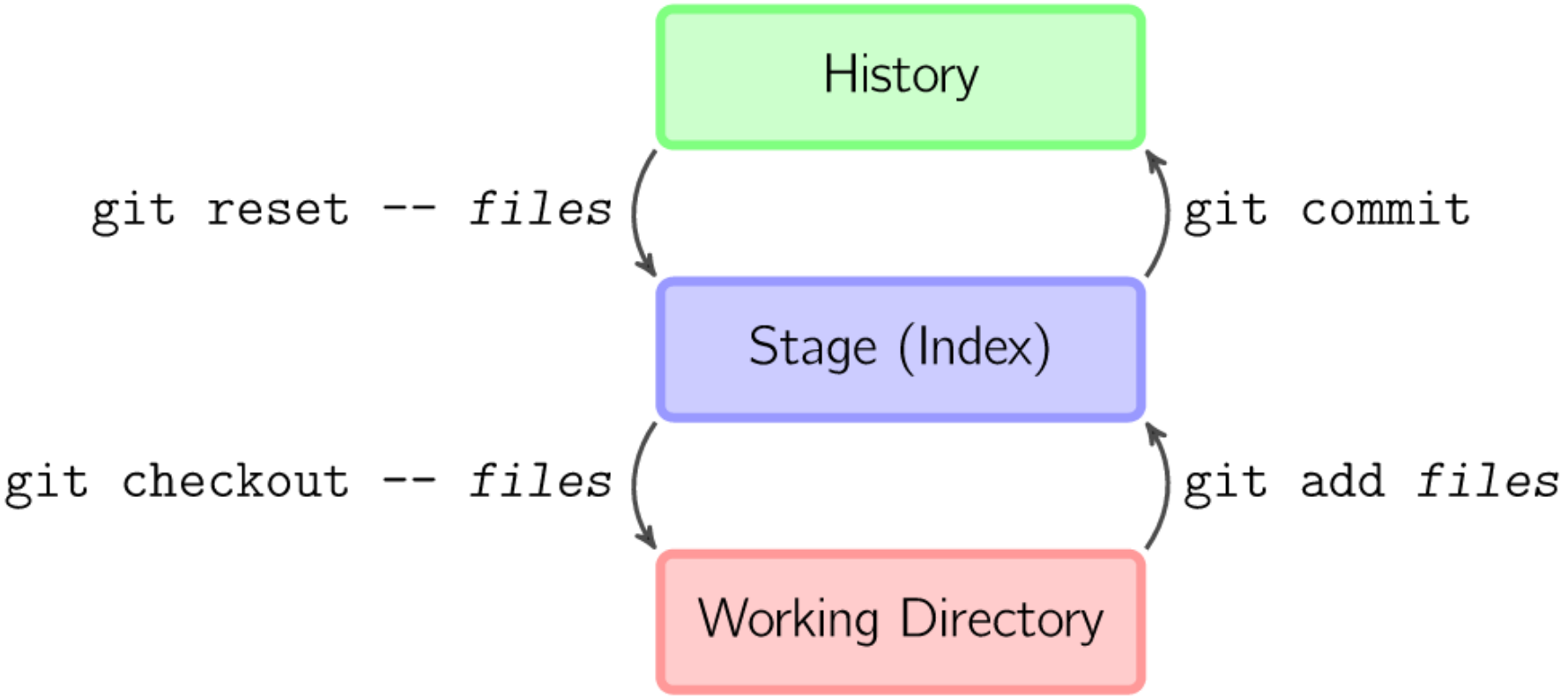
Sobrescrever alterações locais

- Usado em caso de alterações locais indevidas
- Substituir alterações na árvore de trabalho pelo conteúdo mais recente do HEAD.
  - `git checkout -- <arquivo>`
- Alterações já adicionadas ao index e novos arquivos são mantidos.

Sobrescrever alterações locais

- Para remover todas as alterações e commits locais:
  - `git fetch origin`
  - `git reset --hard origin/master`

Uso básico (simplificado)



Evitando o envio de alguns arquivos

- .gitignore
  - Um arquivo .gitignore pode ser criado para evitar o envio indesejado de alguns arquivos específicos para o repositório.

```
target/*
.settings/*
.classpath
.project
/target
*.class
*.jar
*.war
*.ear
```

Evitando modificações devido a diferença de quebra de linha entre diferentes plataformas

- Adicionar o arquivo .gitattributes

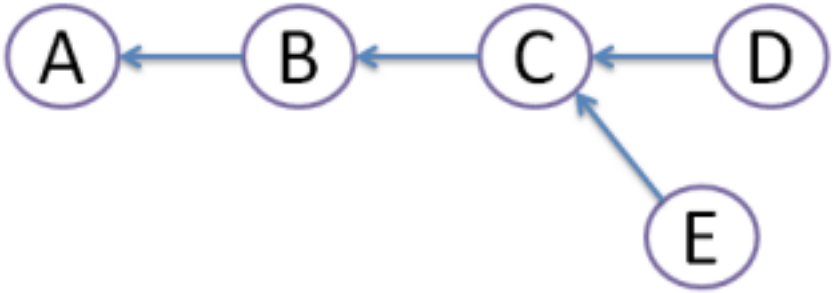
```
# Use LF file endings
* text eol=lf

# Definitively text files
*.xml text
*.java text
*.properties text
*.xhtml text
*.jsp text
*.jsf text
*.txt text
*.c text
*.h text

# Ensure those won't be messed up with
*.jpg binary
*.png binary
```

Diferença entre Merge e Rebase

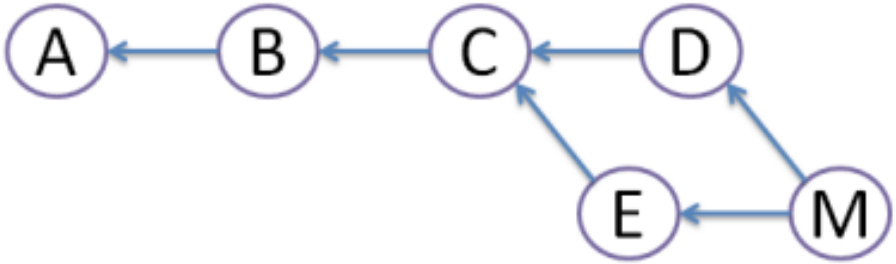
3 Commits

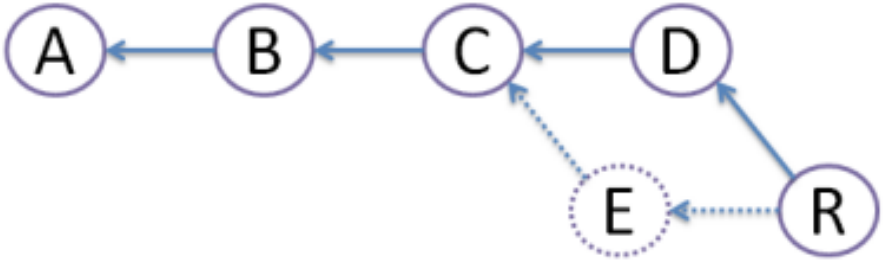


Commit D de João
e
Commit E de Maria

Merge

Rebase





Fluxo de trabalho para enviar e receber alterações

- O rebase analisa commit a commit.
  - Pega todo o histórico de commits e aplica um a um, do primeiro ao último.
  - Reduz a quantidade de conflitos a serem resolvidos de uma única vez ao final do desenvolvimento
    - Em caso de conflitos...
      - `git mergetool`
      - `git rebase --continue`

Obrigado!

Dúvidas, comentários, sugestões?

Regis Pires Magalhães
regismagalhaes@ufc.br

UFC