Introdução ao Git

Author: Eduardo R. D’Avila <erdavila@gmail.com>
Date: Tue Mar 26 19:00:00 2013 -0300

Tópicos

Introdução

Commits

Branches* e *Tags

Interação entre Repositórios

Comandos e Opções Diversas

Por que usar um SCM?

Permite:

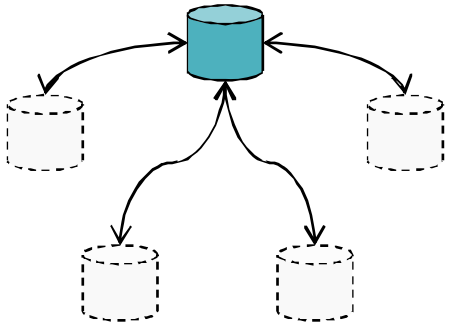
- restaurar versões anteriores
- juntar alterações feitas concorrentemente
- identificar o que foi alterado em cada edição
- identificar qual alteração introduziu um *bug*
- identificar quem fez cada alteração e quando
- marcar versões
- analisar como o projeto evolui

Centralizado x Distribuído

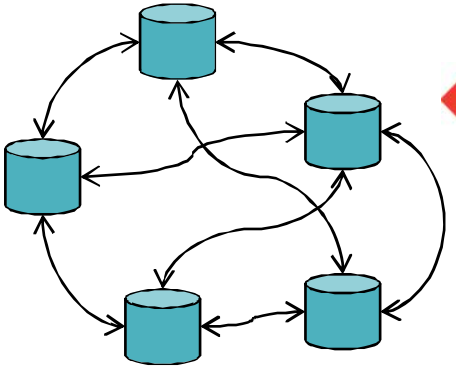
Centralizado

CVS

SUBVERSION



Distribuído



git

mercurial

Git é Distribuído

- Todo diretório de trabalho é um repositório completo
  - Contém todo histórico de alterações
- Principais operações são locais
  - Sem dependência de um servidor
- Somente operações de “sincronização” necessitam acesso a outros repositórios

Repositórios

Repositório do desenvolvedor

- Um diretório usado para se **trabalhar** no projeto
- Contém arquivos do projeto disponíveis para serem editados
- Dados do Git no diretório `.git` na raiz do projeto

Repositório para ser compartilhado (*bare*)

- Um diretório usado para **compartilhar** o projeto
- Sem arquivos do projeto disponíveis para serem editados
- Dados do Git diretamente na raiz do projeto

Criação de repositório:

- `git init [--bare] <diretório>`

Configuração inicial

Configurar nome e e-mail do desenvolvedor:

- `git config --global user.name "Nome do Desenvolvedor"`
- `git config --global user.email username@domínio.com`

Recomendado: habilitar cores

- `git config --global color.ui true`

Commits

Commit

Um *commit* representa um estado do projeto

Contém também autor, data, descrição

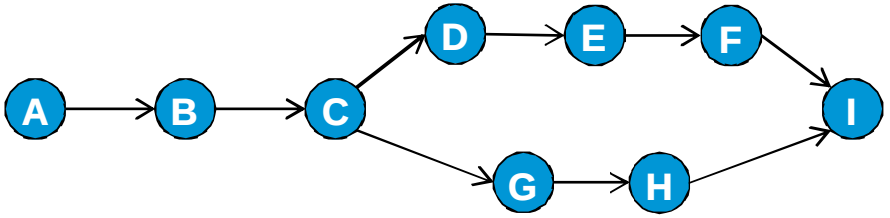
Também chamado de “revisão”

Cada *commit* tem um ancestral

Exceções:

- O primeiro *commit* do projeto
- *Commits* resultantes de *merge*

A cadeia de *commits* forma o histórico de alterações do projeto



Commit

Como fazer um *commit*

1. Usar o comando git-add para **marcar** os arquivos a serem incluídos no **próximo** *commit*:

- `git add <arquivo>... <diretório>...  # Inclui arquivos recursivamente`

2. Efetuar o *commit*:

- `git commit [-m "descrição"]`

Visualizar o histórico de *commits*

A partir do mais recente

- `git log [<parâmetros>...]`

```
commit 2874505bdb80ee8790296428576d977c28708e7f
Author: Eduardo R. D'Avila <erdavila@gmail.com>
Date:   Sat Jul 28 13:48:56 2012 -0300

    Force loading order of test scripts
```

Commits são identificados por *hashes* SHA1 (40 caracteres)

- Gerados pelo Git
- Só é necessário usar caracteres o suficiente para não ser ambíguo, a partir de 4 caracteres

Commit

Index / Staging Area / Cache

É um estágio pré-*commit* – representa o estado que se tornará o próximo *commit*

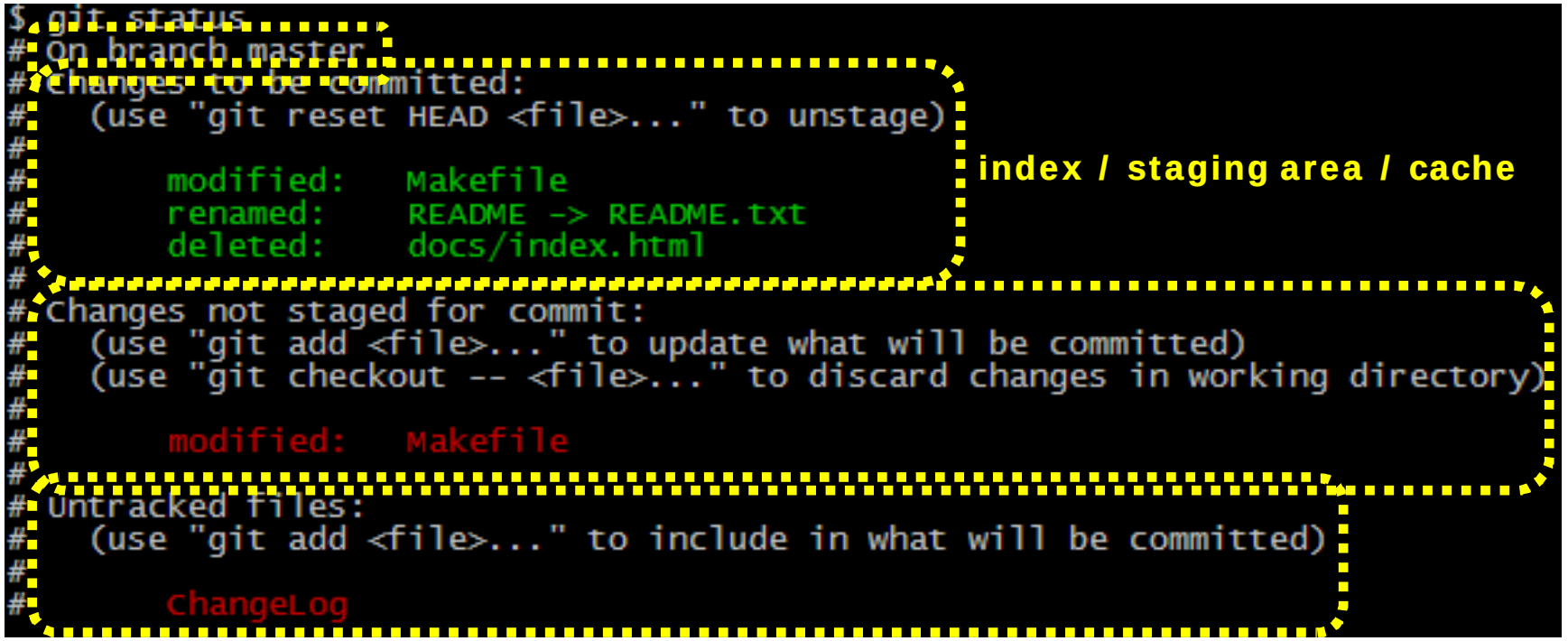
Para identificar arquivos alterados desde o último *commit*:

```
git status
```

Para fazer *commit* diretamente todos os arquivos alterados:

```
git commit -a
```

Comando status



Arquivos versionados mas não-alterados não aparecem na saída do comando status

Comandos Básicos

Marcar o que vai ser incluído no próximo *commit* (adicionar ao *index*)

Arquivos novos ou alterados:

- `git add <arquivos>... <diretórios>...`

Remover do versionamento e apagar:

- `git rm <arquivos>...`
- `git rm –r <diretórios>... # Remove arquivos recursivamente`

Remover do versionamento sem apagar:

- `git rm --cached <arquivos>...`

Renomear ou mover arquivo:

- `git mv <nome antes> <nome depois>...`

Verificar o estado

- `git status`

Efetuar o *commit*

- `git commit`

Branches e Tags

Branches

Branches são como “realidades alternativas”

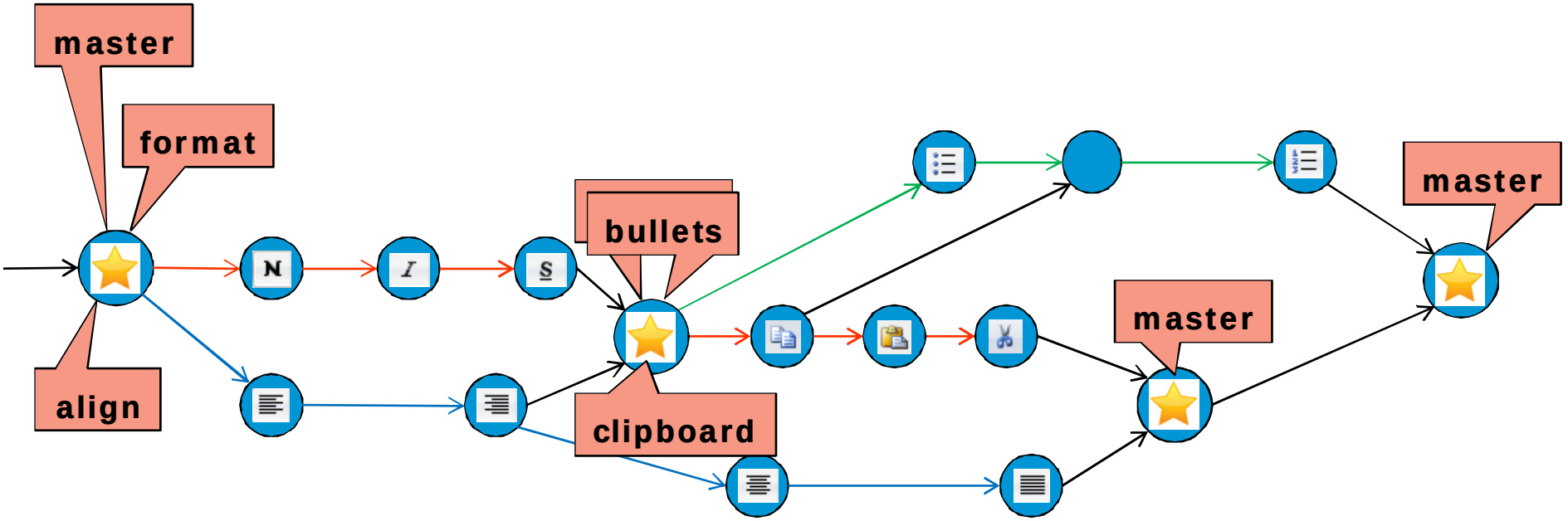
Cada *branch* tem um nome

Branch `master` é criado por padrão

- Não tem nada de especial
  - Pode ser renomeado
  - Pode ser removido
  - Pode ser recriado

Branches

São implementados como referências a *commits*



O nome do *branch* pode ser usado no lugar do *commit* em comandos

Operações com *branches*

Criar

- `git branch <branch> [<commit>]` `# somente cria`
- `git checkout -b <branch> [<commit>]` `# cria e alterna para o branch`

Listar

- `git branch`

O branch atual é marcado com "*"

Alternar

- `git checkout <branch>`

Merge

- `git merge <outro branch>`

Excluir

- `git branch -d <branch>` `# somente se foi "mergeado" no branch atual`
- `git branch -D <branch>`

Remove somente a referência. Os *commits* continuam no histórico.

Merge

Junta ao *branch* atual as alterações de outro *branch*

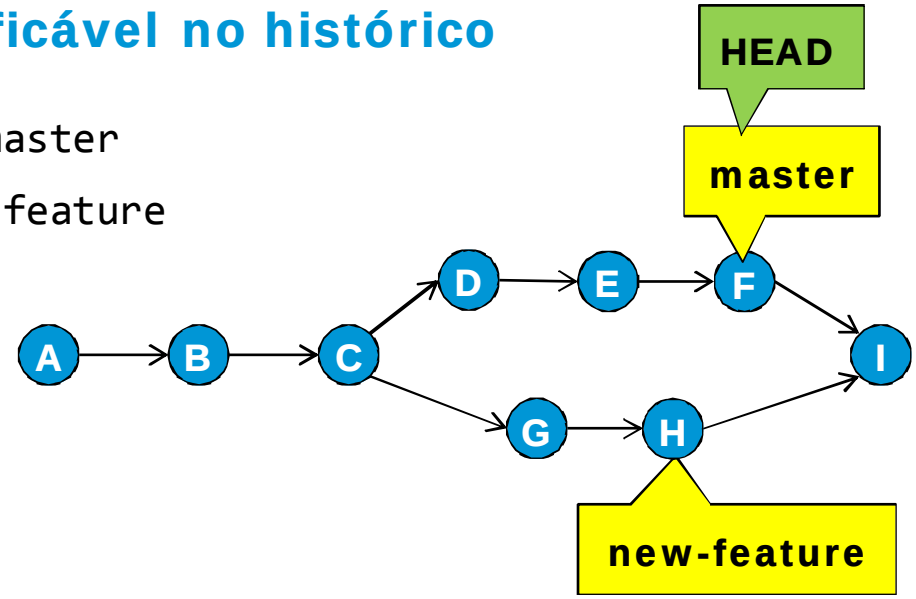
Um novo *commit* é criado

A não ser que:

- Resulte em um "*fast-forward*"
- Haja algum conflito!

Facilmente identificável no histórico

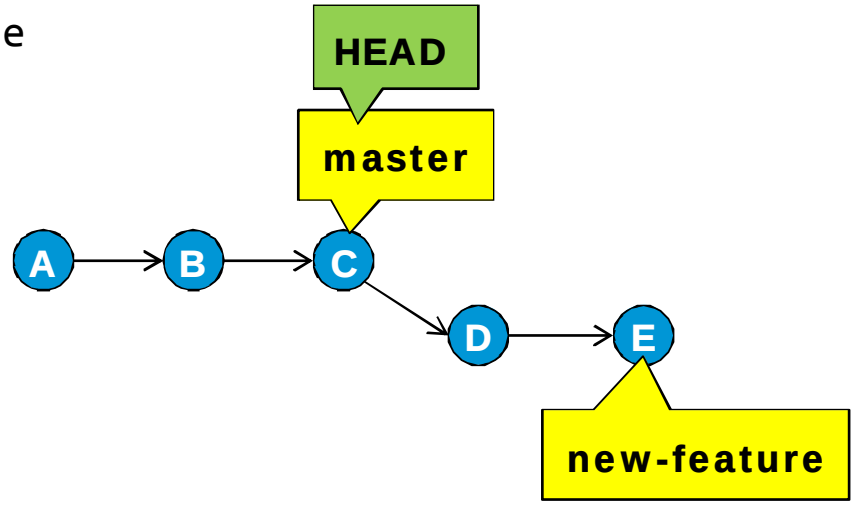
```
$ git checkout master
$ git merge new-feature
```



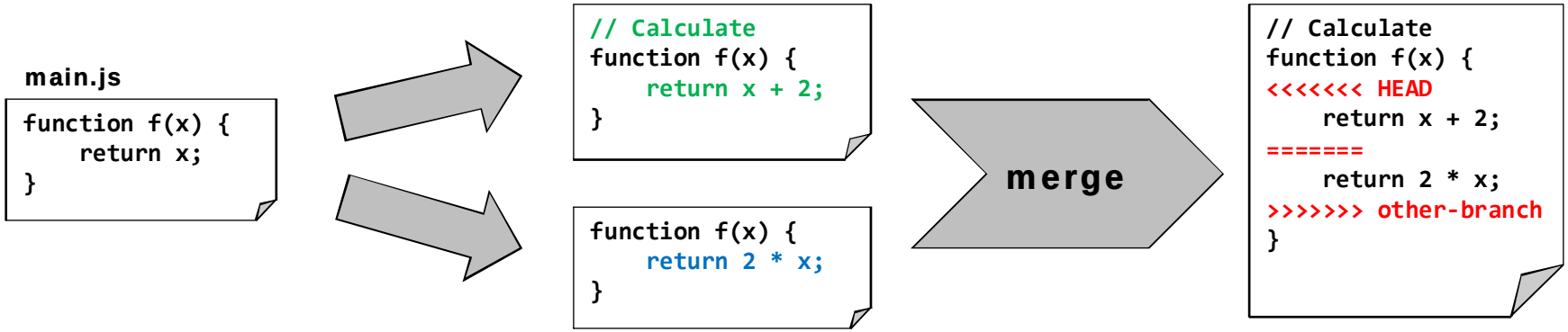
Fast-forward

Simplesmente avança o *branch* atual para alcançar o outro *branch*

```
$ git checkout master
$ git merge new-feature
```



Merge com conflito




```
$ git merge other-branch
Auto-merging main.js
CONFLICT (content): Merge conflict in main.js
Automatic merge failed; fix conflicts and then commit the result.

$ git status
# On branch master
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to mark resolution)
#
#       both modified:      main.js
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Merge com conflito - resolução

O que fazer?

- Editar arquivos **resolvendo** os conflitos
- `git add <arquivos>` `# Marca a resolução dos conflitos`
- `git commit` `# Finaliza o merge`

ou cancelar:

- `git merge --abort`

Ferramentas de merge externas

- Git suporta por padrão: araxis, bc3, diffuse, ecmerge, emerge, gvimdiff, kdiff3, meld, opendiff, p4merge, tkdiff, tortoisemerge, vimdiff, xxdiff
- Outras ferramentas podem ser suportadas através de configurações adicionais
- `git mergetool [--tool=<ferramenta>] [<arquivo>]`

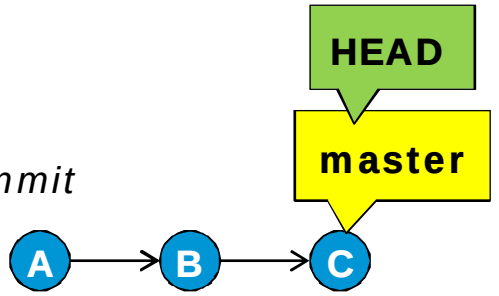
HEAD

HEAD é o estado do histórico sobre o qual se está trabalhando

Implementado como referência a um *branch* (o *branch* atual) ou diretamente a um *commit*

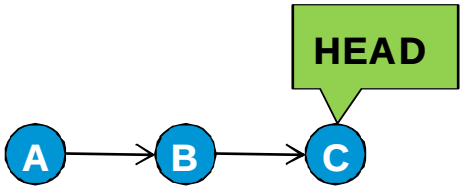
`git checkout <branch>`

- HEAD referencia o *branch*, que por sua vez referencia o *commit*
- HEAD é o *commit* mais recente do *branch*
- Ao fazer um *commit*, o *branch* avança



`git checkout <commit ou branch remoto ou tag>`

- HEAD referencia diretamente um *commit*
- Repositório fica em "*detached HEAD*"
- Ao fazer um *commit*, HEAD avança
- Cuidado! *Garbage collector* pode eliminar *commits* sem referência



Tags

Marcam uma versão do projeto

São implementadas como referências a *commits*

Não podem ser alteradas

Podem ser assinadas com GPG

Listar

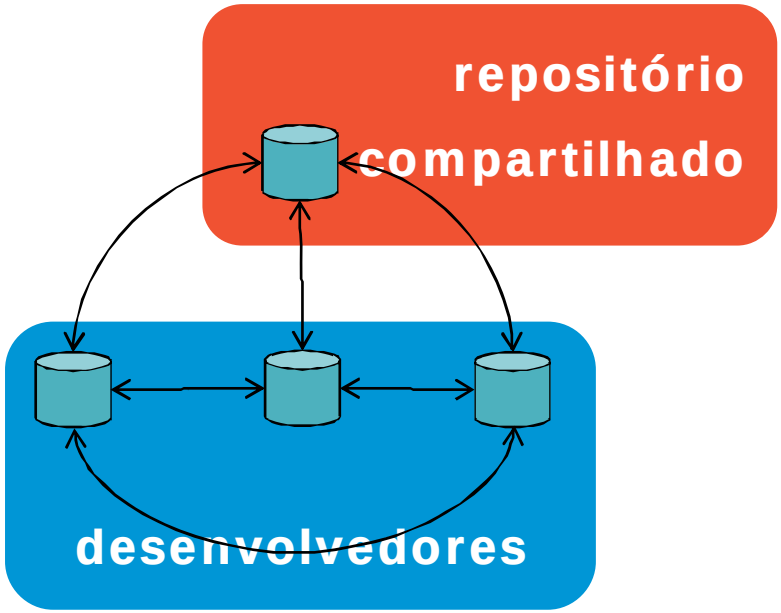
- `git tag`

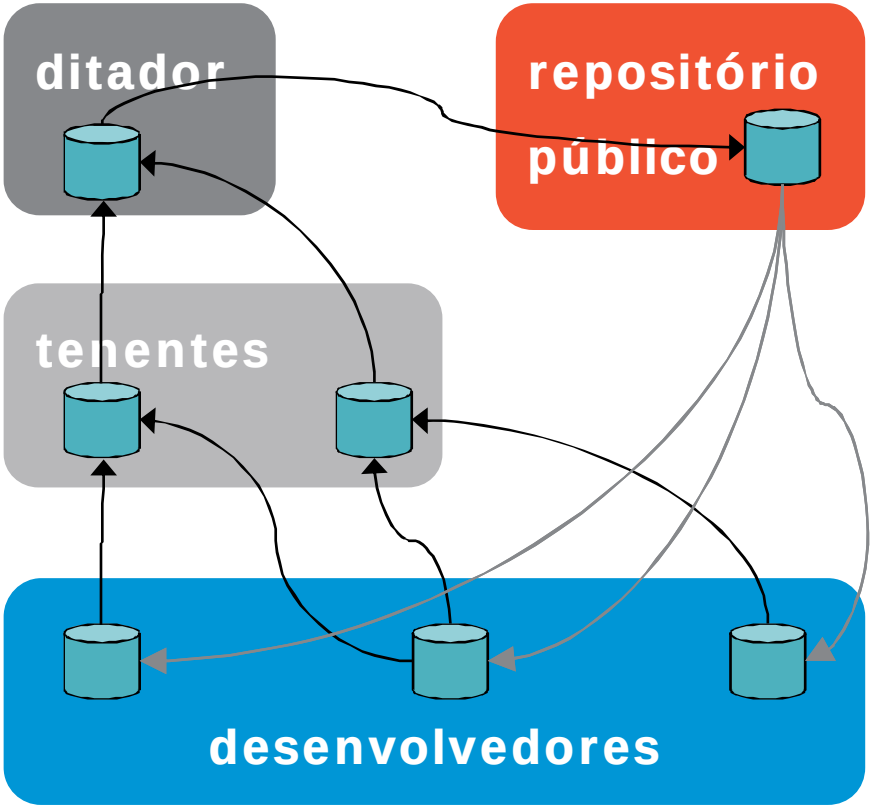
Criar

- `git tag <nome> [<commit>]`

Interação entre Repositórios

Workflows


repositório
compartilhado
desenvolvedores


ditador
repositório
público
tenentes
desenvolvedores

Repositórios Remotos

Identificados por uma URL ou caminho:

Sistema de arquivos local:

- /git/projeto.git
- ../outro-projeto
- file:///home/fulano/projeto

SSH (Git deve estar instalado no servidor):

- ssh://usuario@servidor/caminho
- usuario@servidor:caminho

Outros protocolos:

- git://servidor/caminho – Serviço do Git no servidor
- http[s]://servidor/caminho
- ftp[s]://servidor/caminho
- rsync://servidor/caminho

Nomes de Repositórios Remotos

Nomes podem ser usados para referenciar repositórios remotos

Definir um nome para um repositório remoto em um repositório **existente**:

- `git remote add <nome> <url ou caminho>`

Múltiplos repositórios podem ser referenciados – afinal Git é distribuído!

Listar repositórios remotos:

- `git remote –v`

Clonagem de repositório

- `git clone [--bare] <url ou caminho> [<diretório>]`

Automaticamente define o nome “origin” para referenciar o repositório clonado

Branches Remotos

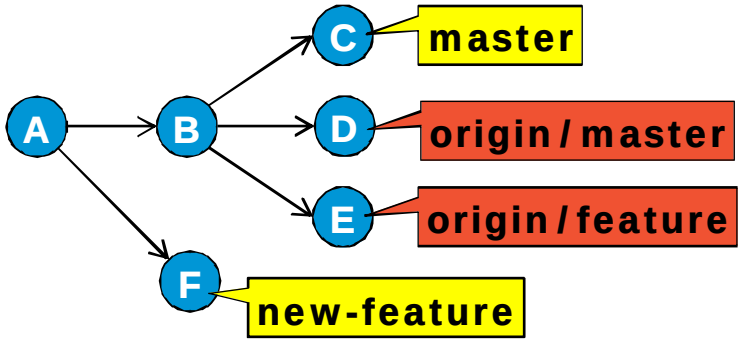
Cada repositório tem seu conjunto de *branches*

Branches remotos identificam o estado de um *branch* em um repositório remoto

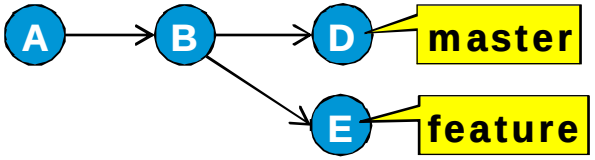
Identificados por "`<repositório>/<branch>`" ou "`remotes/<repositório>/<branch>`"

Repositório X

Usa o nome "origin" para referenciar o repositório Y



Repositório Y



Listar *branches* remotos

- `git branch -r` `# Lista somente branches remotos`
- `git branch -a` `# Lista branches locais e remotos (o atual é marcado com "*")`

Operações com repositórios remotos

Somente duas **operações básicas envolvem comunicação com repositório remoto**

`git fetch [--tags] <repositório>`

- Atualiza *branches* e *tags* (e seus *commits*) do repositório remoto
- **Não toca** nos *branches* locais

`git push [--tags] <repositório> <branch local>:<branch remoto>`

- Atualiza (ou cria) o *branch* do repositório remoto com o estado do *branch* local
- Só funciona se for um "*fast-forward*" (pois não há como fazer um *merge* remoto)

Extra:

`git pull <repositório> <branch remoto>`

- Executa um **fetch** do *branch* remoto e faz **merge** com o ***branch* atual**

Branches “seguidos” (*tracked*)

Cada *branch* local pode estar associado a um *branch* remoto

Criar um *branch* a partir de um *branch* remoto faz a associação automaticamente:

- `git branch <novo branch local> <repositório>/<branch remoto>`
- `git checkout -b <novo branch local> <repositório>/<branch remoto>`

Associar *branch* já existente:

- `git branch --set-upstream <branch local> <repositório>/<branch remoto>`
- `git branch --set-upstream-to <repositório>/<branch remoto> <branch local>`

A partir da versão 1.8.0

Operações simplificadas no *branch* atual:

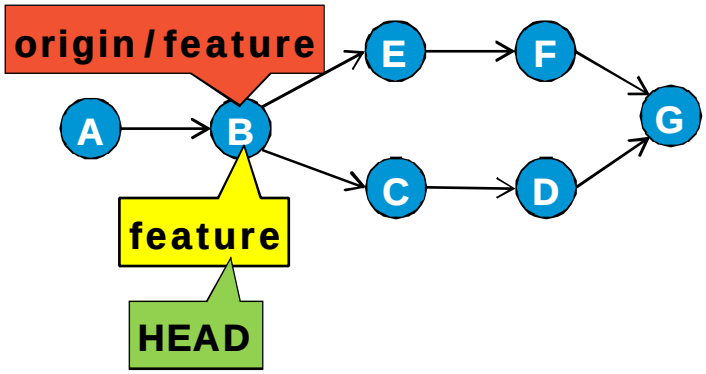
- `git pull` `# Sem mais parâmetros!`
- `git push` `# Sem mais parâmetros!`

Listar associações:

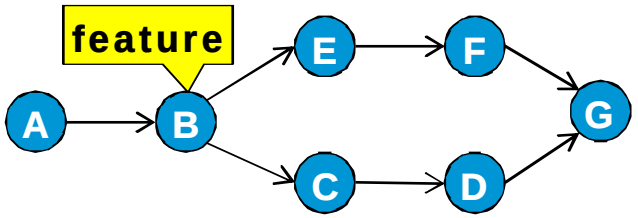
- `git branch -v -v` `# Mesmo parâmetro duas vezes`

Exemplo

Local



origin




```
$ git fetch origin
$ git checkout -b feature origin/feature
$ git add <arquivos> ; git commit
$ git push
ERRO!
$ git pull    # fetch + merge
$ git push
```

Comandos e Opções Diversas

Comandos

Git tem mais de 130 (!) comandos

Os comandos são classificados como:

- "porcelain" – Comandos de alto nível. Para serem usados por humanos
- "plumbing" – Comandos de baixo nível. Para serem usados por scripts

Em comandos que esperam um *commit* como parâmetro, pode-se usar qualquer item que referencie o *commit*:

- *branch*
- *tag*
- HEAD
- *branch* remoto

Ajuda

- `man git-<comando>`
- `git help <comando>`

Comando `stash`

Guarda temporariamente alterações pendentes

Útil quando se começa uma tarefa e precisa-se alternar para outra tarefa no Git

Funciona como uma pilha:

- `stash@{0}` (ou simplesmente "`stash`") – no topo da pilha. É o default para `pop`, `apply` e `drop`
- `stash@{1}`
- `stash@{2}`
- …

Comandos

```
• git stash [save ["descrição"]]          # cria novo stash no topo da pilha
• git stash pop [--index] [<stash-id>]    # aplica e remove stash da pilha
• git stash apply [--index] [<stash-id>]  # aplica stash
• git stash list                          # lista todos os stashes
• git stash drop [<stash-id>]             # descarta stash da pilha sem aplicá-lo
• git stash clear                         # descarta todos os stashes
• git show <stash-id>                     # mostra alterações do stash
```

O parâmetro `--index` faz restaurar o que estava no *index* quando o *stash* foi salvo

A aplicação de um *stash* em um outro *commit* é sujeita a conflitos

Comando `log`

Lista *commits* em ordem inversa (o mais recente primeiro)

Provê **muitas** opções de visualização do histórico

- `git log [<commits ...>]` – Histórico a partir dos *commits* ou *branches* especificados
- `git log [<commit1>]..[<commit2>]` – Histórico do que o `<commit2>` tem de novo em relação ao `<commit1>`
- `git log [--follow] -- <arquivo ou diretório>` – Lista somente *commits* que incluem alterações no `<arquivo ou diretório>`. O parâmetro `--follow` força a identificação de renomeação e cópias de arquivo.

Algumas opções:

- `-S <string>` – Lista somente *commits* que removeram ou incluíram `<string>` em algum arquivo
- `--stat` – Inclui lista de arquivos que foram alterados em cada *commit* e estatísticas
- `-p [-C]` – Inclui as alterações de cada arquivo (*diffs*) e, opcionalmente, detecta renomeações e cópias
- `--oneline` – Mostra somente *hash* e mensagem de cada *commit*
- `--graph` – Mostra grafo de *branches* e *merges*
- `--decorate` – Identifica outras referências (branches e tags) no histórico

Comandos `show` / `diff` / `blame`

`show` – Exibe informações de um *commit* e/ou alterações e conteúdo de arquivos

- `git show [<commit>]` – Mostra dados do *commit* e suas alterações de arquivo
- `git show <commit> -- <arquivo>` – Mostra **alterações** feitas no ARQUIVO no *commit* especificado
- `git show <commit>:<arquivo>` – Mostra **conteúdo** do ARQUIVO no *commit* especificado

`diff` - Mostra diferenças em conteúdos de arquivos

- `git diff` – Mostra alterações que ainda **não estão** no *index*
- `git diff --cached` – Mostra alterações que **estão** no *index*
- `git diff <commit>` – Mostra diferenças entre o estado atual e o *commit*
- `git diff <commit1> <commit2>` – Mostra diferenças entre os *commits*
- `git diff -- <arquivo>` – Seleciona arquivo para mostrar diferenças

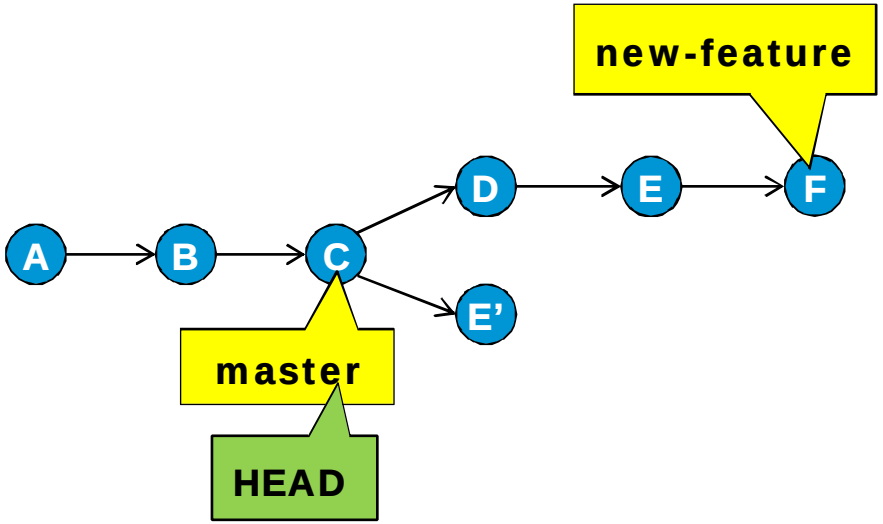
`blame` – Exibe informações de quando cada linha de um arquivo foi alterada e quem fez a alteração

- `git blame <arquivo>`

Comando `cherry-pick`

Re-aplica alterações de um único *commit*

```
$ git checkout master
$ git cherry-pick E
```

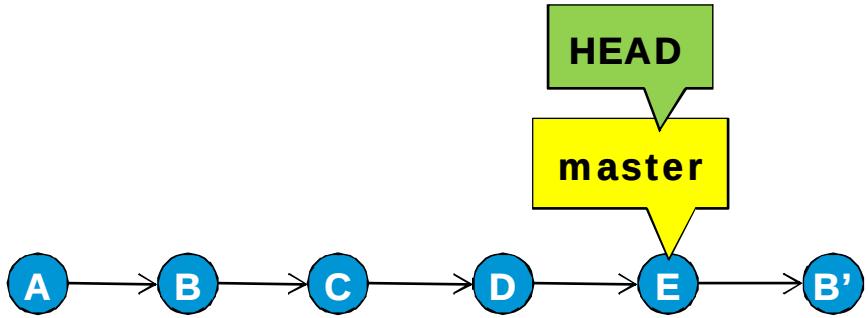


Pode ser necessário resolver conflitos

Comando `revert`

Inverte as alterações feitas por um *commit*

```
$ git checkout master
$ git revert B
```



Pode ser necessário resolver conflitos

Operações em linhas de arquivos

O parâmetro `-p` em alguns comandos permite selecionar interativamente **partes de arquivos**

Seleciona partes a serem adicionadas ao *index* (e posteriormente *commitadas*)

- `git add -p ARQUIVO`

Seleciona partes a serem removidas do *index*

- `git reset -p ARQUIVO`

Seleciona partes alteradas a serem descartadas

- `git checkout -p ARQUIVO`

Ferramentas gráficas

`gitk`

Correspondente ao comando `git log`, mas mais visual e navegável

`git gui`

Provê funcionalidades de *commit*, *merge*, gerência de *branches* e de repositórios remotos, entre outras opções

`EGit/JGit`

Plugin para Eclipse

`SourceTree`

http://www.sourcetreeapp.com/

Perguntas?
Dúvidas?
Sugestões?
Críticas?
Opiniões?

Links

Pro Git

http://git-scm.com/book

Git Cheatsheet (stash x work-dir x index x local x remote)

http://ndpsoftware.com/git-cheatsheet.html

A Visual Git Reference

http://marklodato.github.com/visual-git-guide/index-en.html

git ready

http://gitready.com/