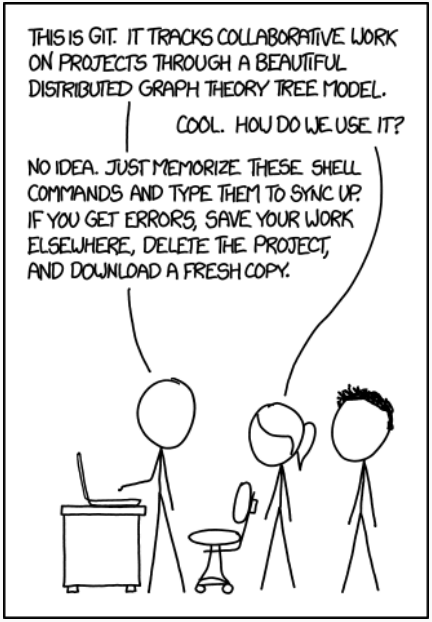
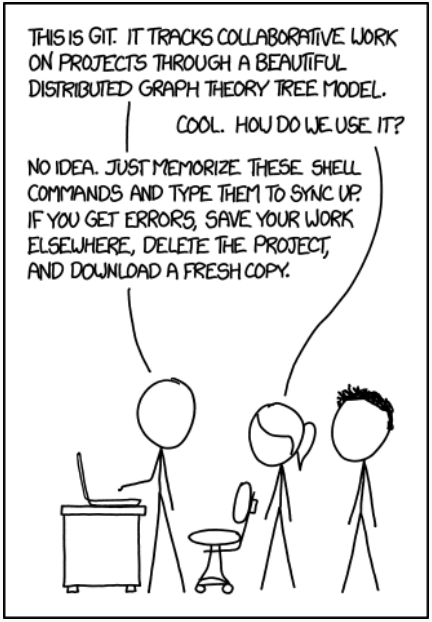
Parte II

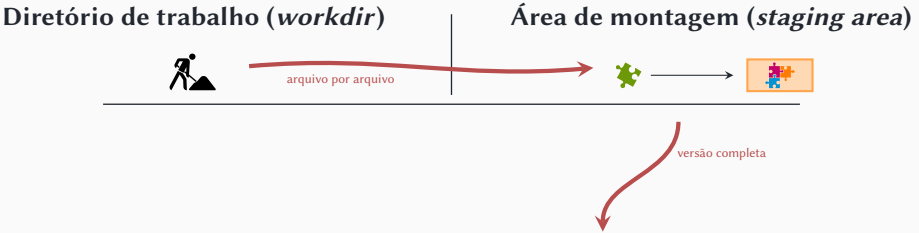
Entendendo o git

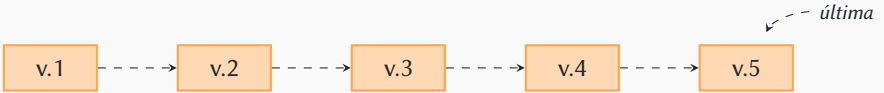

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.



If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of "It's really pretty simple, just think of branches as..." and eventually you'll learn the commands that will fix everything.

(from xkcd.com/1597)


Diretório de trabalho (*workdir*)
Área de montagem (*staging area*)
arquivo por arquivo
versão completa


última
v.1
v.2
v.3
v.4
v.5
tempo

Quem sai na foto?

- **Cada nova versão no histórico é uma *cópia* da área de montagem**

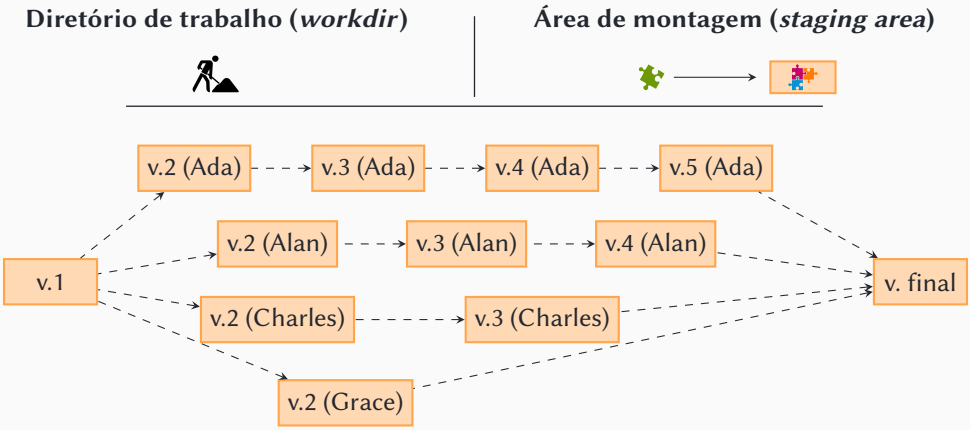
  portanto...
- **A área de montagem nunca está vazia!**
  - Ela sempre tem uma cópia da "última foto", ou seja, da última versão
  - Não é preciso fazer **`git add`** com arquivos que não mudaram, só o que efetivamente mudou
- **E como remover um arquivo da área de montagem?**
- **`git rm --cached arquivo`**
  - » *(**`--cached`** é sinônimo de **`--staged`**)*
  - » *(**`git rm arquivo`** apaga da área de trabalho também)*
  - **`git rm`** **não é** o contrário de **`git add`**!
    - » *O contrário de **`git add arquivo`** é **`git restore --staged arquivo`***
      (copia o arquivo da última versão no repositório para a área de montagem)
      *(antigamente, **`git reset arquivo`**)*
    - » ***`git restore arquivo`*** *apaga as alterações da área de trabalho*
      (copia o arquivo da área de montagem para a área de trabalho)
      *(antigamente, **`git checkout arquivo`**)*

De onde vêm esses IDs de cada *revision*?

Por que não é v.1 → v.2 → v.3 etc.?

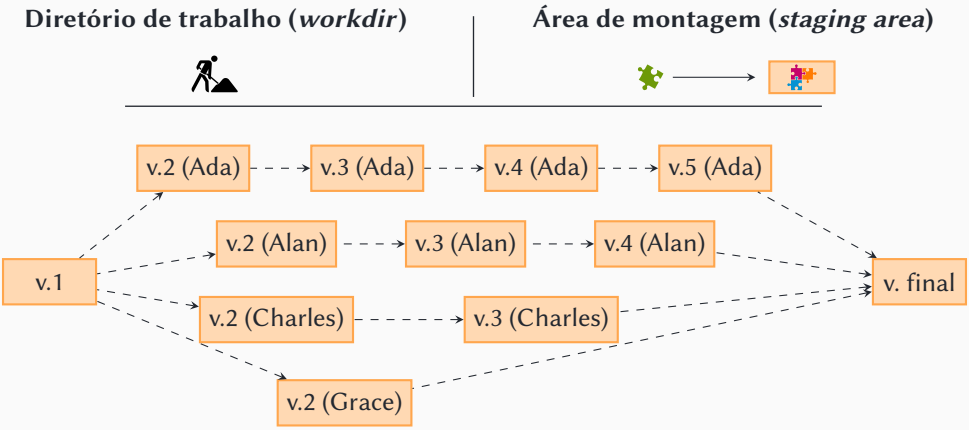
Multiverso: o mundo não é linear

“Cada um faz sua parte e depois a gente junta tudo”



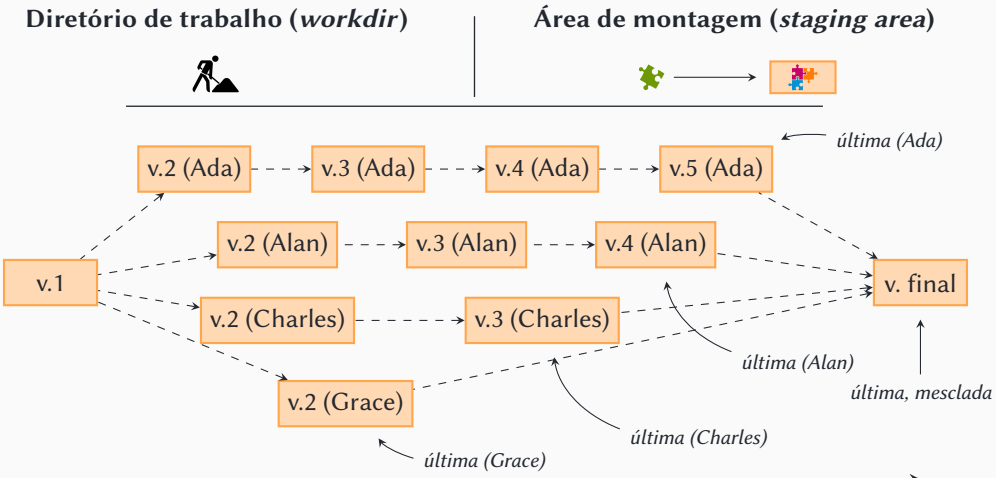
“Cada um faz sua parte e depois a gente junta tudo”

- Cada um vê uma linha do tempo diferente!



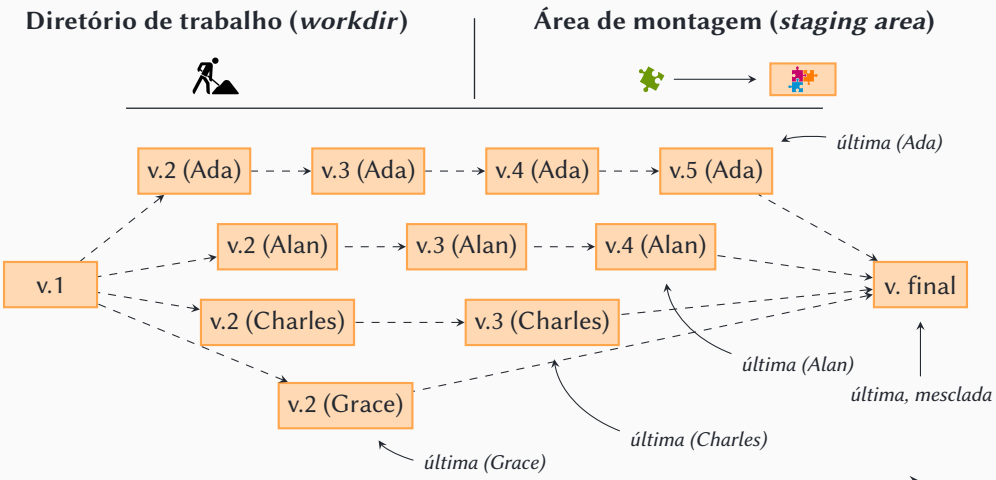
“Cada um faz sua parte e depois a gente junta tudo”

- Cada um vê uma linha do tempo diferente!
- Cada um vê uma “última” versão (o “presente”) diferente!



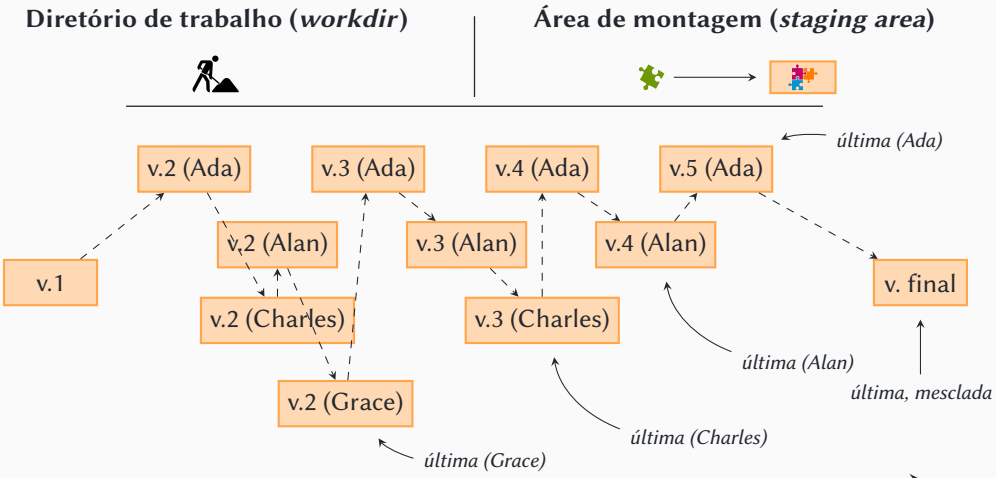
“Cada um faz sua parte e depois a gente junta tudo”

- Como ordenar as versões?
  - » *Versão + nome?*



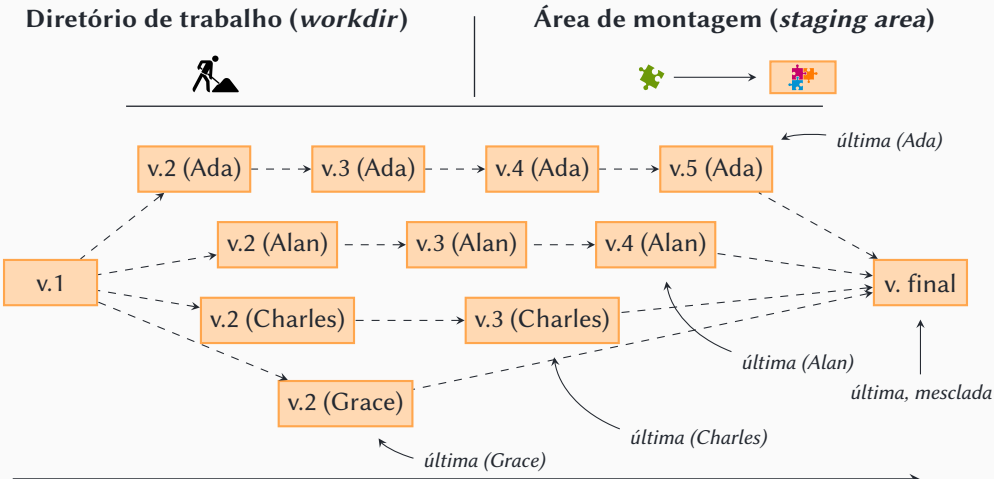
“Cada um faz sua parte e depois a gente junta tudo”

- Como ordenar as versões?
  - » *Data?*

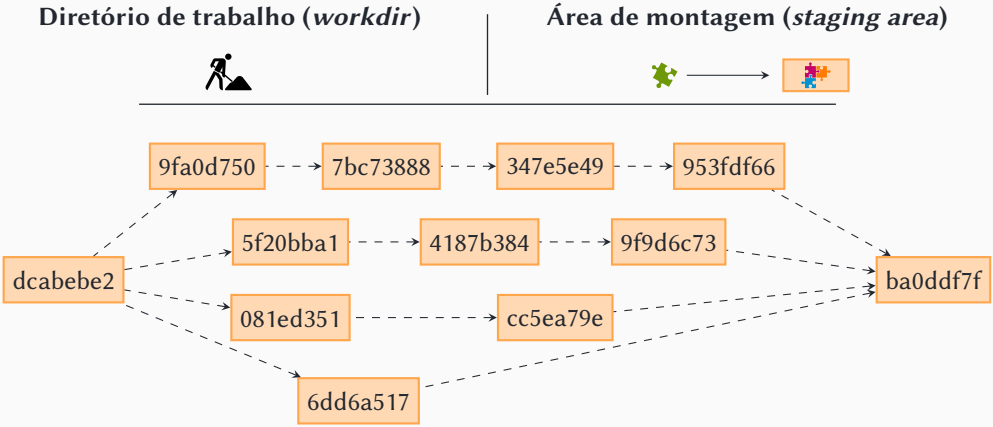


Não é possível inferir uma ordem entre as versões!

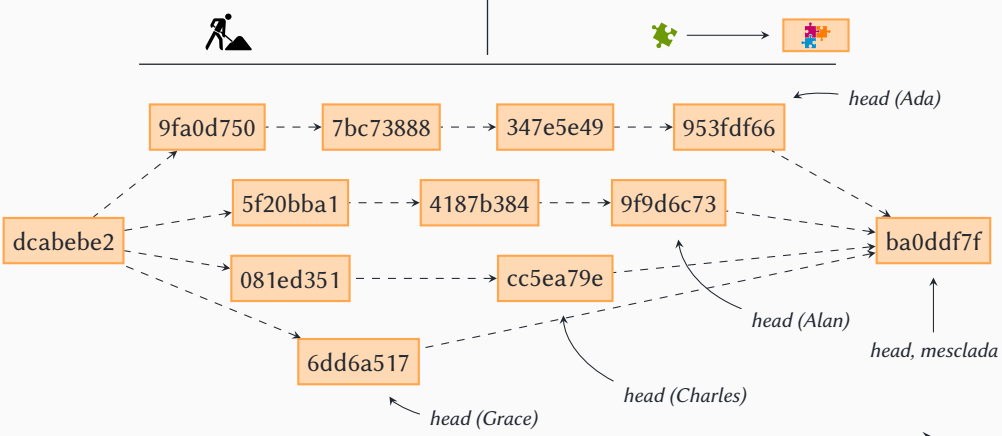
- É preciso explicitar a sequência
- IDs não são confiáveis (podem ser repetidos, significado indefinido)



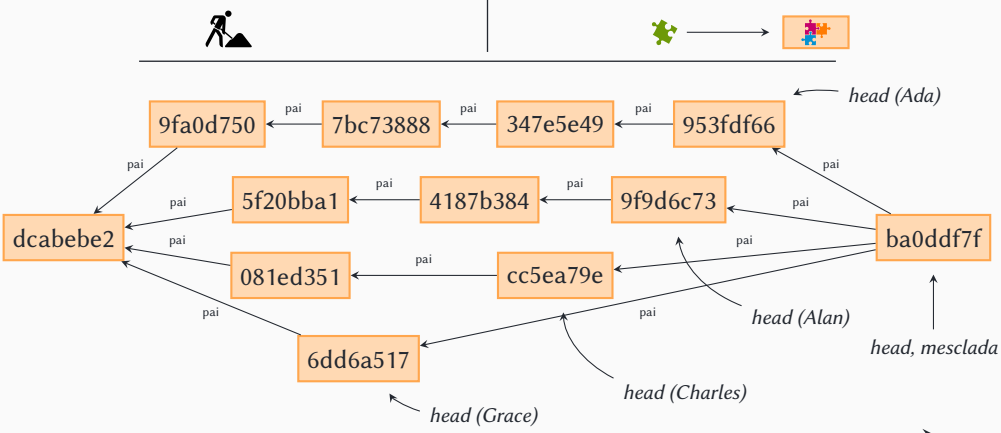
- **IDs aleatórios (não exatamente)**
  - Unicidade



- **IDs aleatórios (não exatamente)**
  - Unicidade
- **Cada "última" versão → *head***



- **IDs aleatórios (não exatamente)**
  - Unicidade
- **Cada "última" versão → *head***
- **Não se pode ver o futuro**
  - Antepassados, não descendentes
    - » `git log` *só percorre a árvore genealógica*



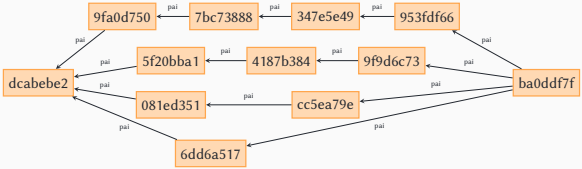
O que é o git

Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)

Depósito de versões encadeadas

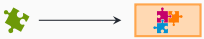
(grafo de versões)



O que é o git

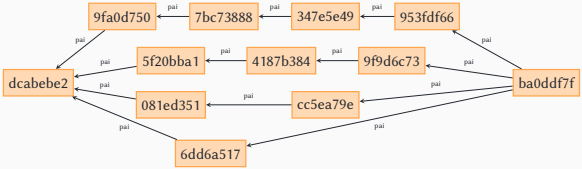
Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)



Depósito de versões encadeadas

(grafo de versões)



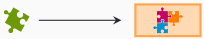
Ramos/branches

(referências são atualizadas automaticamente)

ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	ba0ddf7f

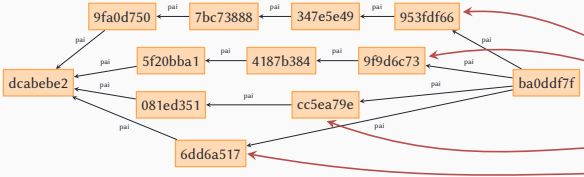
Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)



Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

(referências são atualizadas automaticamente)

ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	ba0ddf7f

O que é o git

Tudo o que git faz é

- **Criar novas *revisions* (com a informação sobre os antepassados)**
  - (O que envolve copiar arquivos de/para o repositório, o *workdir* e a *staging area*)
- **Manipular a tabela de *heads***

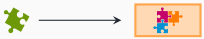
Todos os demais recursos são baseados nesses

Manipulando a tabela de *heads*

- `git commit --amend`

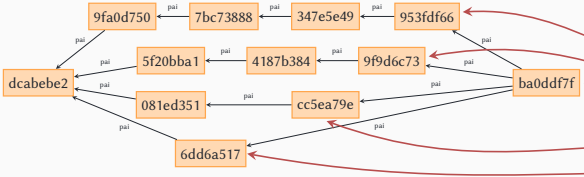
Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)



Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

(referências são atualizadas automaticamente)

ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	ba0ddf7f

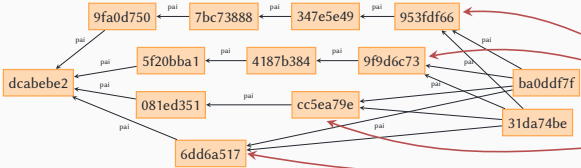
Manipulando a tabela de *heads*

- **`git commit --amend`**
  - » *Cria uma nova revision similar à anterior*

Diretório de trabalho (*workdir*) | **Área de montagem (*staging area*)**

Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

(referências são atualizadas automaticamente)

ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	ba0ddf7f

Manipulando a tabela de *heads*

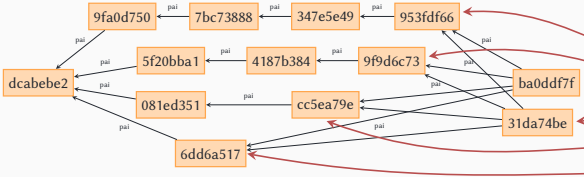
- **`git commit --amend`**
  - » *Cria uma nova revision similar à anterior*
  - » *Modifica a tabela de heads*

Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)

Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

(referências são atualizadas automaticamente)

ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	31da74be

Viajando no multiverso

Viajando no multiverso

- **Cada "universo" (incluindo o histórico) é um ramo (*branch*)**
  - Identificado pela *head* correspondente (e mais nada!)
    - » *A partir da head, podemos percorrer todo o histórico*
    - » *(tip seria um nome muito melhor!)*

- **`git branch`**
  - Lista os ramos ("universos"), ou seja, as *heads*
  - O repositório "nasce" com um ramo pronto: **`main`** (antigamente, **`master`**)

- **`git branch <NOME>`**
  - Cria um novo ramo (uma nova *head*) a partir do ramo atual
  - No momento da criação, os dois ramos/*heads* apontam para a mesma *revision*

Viajando no multiverso: oncotô?

- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?

Viajando no multiverso: oncotô?

- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?
- **Oncotô? (O que é "aqui/agora"?)**

Viajando no multiverso: oncotô?

- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?
- **Oncotô? (O que é "aqui/agora"?)**
- **`HEAD`**

Viajando no multiverso: oncotô?

- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?
- **Oncotô? (O que é "aqui/agora"?)**
- **`HEAD`**
  - Sim, isso é muito idiota!

Viajando no multiverso: oncotô?

- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?
- **Oncotô? (O que é "aqui/agora"?)**
- **`HEAD`**
  - Sim, isso é muito idiota!
    - » *(pense que existem várias cabeças, mas só um cabeção :-p)*

Viajando no multiverso: oncotô?

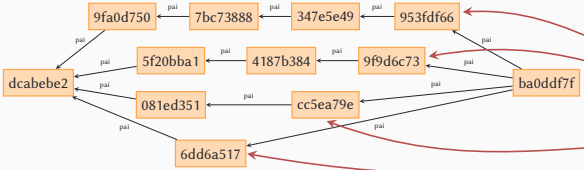
- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?
- **Oncotô? (O que é "aqui/agora"?)**
- **`HEAD`**
  - Sim, isso é muito idiota!
    - » *(pense que existem várias cabeças, mas só um cabeção :-p)*
- **`HEAD` normalmente não aponta para um `ID`, mas para o nome do *branch***
  - Só assim para podermos atualizar o *branch* quando fazemos **`commit`**

Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)

Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

(referências são atualizadas automaticamente)

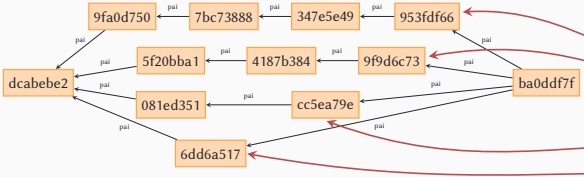
ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	ba0ddf7f

Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)

Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

(referências são atualizadas automaticamente)

ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	ba0ddf7f
HEAD	main

Viajando no multiverso: HEAD

- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?
- **Oncotô? (O que é "aqui/agora"?)**
- **`HEAD`**

Viajando no multiverso: HEAD

- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?
- **Oncotô? (O que é "aqui/agora"?)**
- **`HEAD`**
- **Como atualiza `HEAD`?**

Viajando no multiverso: HEAD

- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?
- **Oncotô? (O que é "aqui/agora"?)**
- **`HEAD`**
- **Como atualiza `HEAD`?**
  - **`git switch grace`** (ou **`git checkout grace`**)
    1. *Atualiza o workdir e a staging area*
    2. *Faz* **`HEAD`** $\rightarrow$ **`grace`**

Viajando no multiverso: HEAD

- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?
- **Oncotô? (O que é "aqui/agora"?)**
- **`HEAD`**
- **Como atualiza `HEAD`?**
  - **`git switch grace`** (ou **`git checkout grace`**)
    1. *Atualiza o workdir e a staging area*
    2. *Faz* **`HEAD`** $\rightarrow$ **`grace`**
- **Lição de casa**
  - **`git switch --detach ID`** (ou **`git checkout ID`**) quando **`ID`** não é um *branch*
    - » *O que acontece com* **`HEAD`***?*
    - » *Será que "detached HEAD" é útil para algo além de "dar uma olhada"?*
    - » *Como sair do estado "detached HEAD"?*

Manipulação de *branches* X restauração de arquivos

- **Você acrescentou indevidamente uma funcionalidade que só funciona no Linux**
  - Opção 1: Cria um novo *branch* diferente (“multiplataforma”) sem essa alteração
  - Opção 2: Continua “em frente”: cria uma nova *revision* que “des-modifica” o que foi feito
    - » *Não é uma “viagem no multiverso”, é uma restauração de arquivos a um estado anterior (pense na restauração de um vaso quebrado)*

Manipulação de *branches* X restauração de arquivos

- **Você acrescentou indevidamente uma funcionalidade que só funciona no Linux**
  - Opção 1: Cria um novo *branch* diferente (“multiplataforma”) sem essa alteração
  - Opção 2: Continua “em frente”: cria uma nova *revision* que “des-modifica” o que foi feito
    - » *Não é uma “viagem no multiverso”, é uma restauração de arquivos a um estado anterior (pense na restauração de um vaso quebrado)*

- **Manipulações de *branches* operam no nível das *revisions* como um todo**
  - O próximo **`git commit`** cria um novo *branch* (**`git branch`**, **`git switch`**, **`git commit`**)
    - » *Ou reescreve à força a história do branch atual, em geral uma má ideia (**`git reset`**)*

Manipulação de *branches* X restauração de arquivos

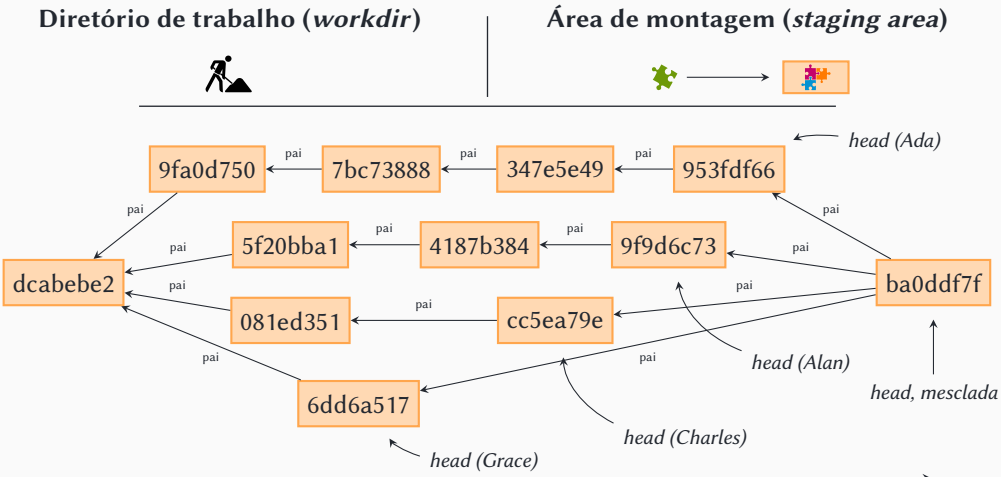
- **Você acrescentou indevidamente uma funcionalidade que só funciona no Linux**
  - Opção 1: Cria um novo *branch* diferente ("multiplataforma") sem essa alteração
  - Opção 2: Continua "em frente": cria uma nova *revision* que "des-modifica" o que foi feito
    - » *Não é uma "viagem no multiverso", é uma restauração de arquivos a um estado anterior (pense na restauração de um vaso quebrado)*

- **Manipulações de *branches* operam no nível das *revisions* como um todo**
  - O próximo **`git commit`** cria um novo *branch* (**`git branch`**, **`git switch`**, **`git commit`**)
    - » *Ou reescreve à força a história do branch atual, em geral uma má ideia (**`git reset`**)*

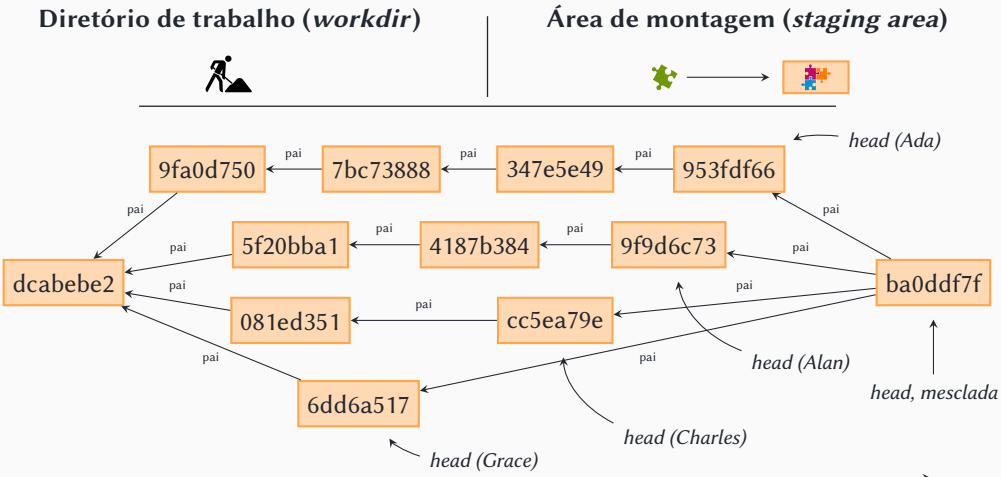
- **Restaurações operam no nível de cada arquivo individualmente (mesmo que incluam todos os arquivos de uma vez)**
  - **`git restore --source ID arquivo`**: copia **`arquivo`** da *revision* **`ID`** para a área de trabalho
  - O próximo **`git commit`** continua a história normalmente
  - É possível inclusive fazer outras modificações no arquivo antes de **`git commit`**

Versões sintéticas e mesclas

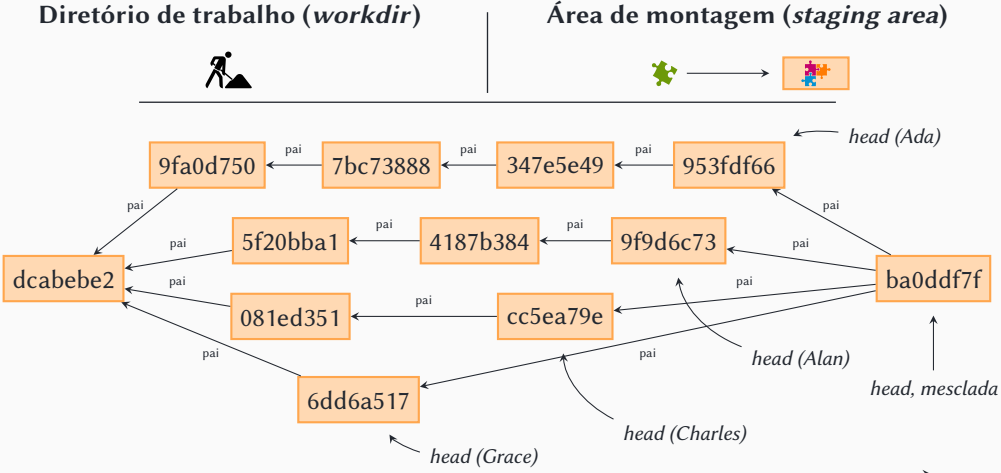
- Quem criou a *revision* `ba0ddf7f`?



- Quem criou a *revision* `ba0ddf7f`?
  - Ninguém!



- **Quem criou a *revision* `ba0ddf7f`?**
  - **Ninguém!**
    - » *Ok, alguém criou: o git.*



Versões sintéticas

- **Sabemos que git é capaz de comparar *revisions* (`git diff`)**
- **git também é capaz de *sintetizar revisions* a partir das diferenças**
  - Identifica as diferenças entre duas *revisions* **`A`** e **`B`**
  - Coloca uma *outra revision* **`C`** na área de trabalho e *staging area*
  - Aplica essas diferenças sobre a área de trabalho (*revision* **`C`**)
  - Faz **`commit`**, criando uma nova *revision* **`D`**
    - » *No branch atual*
  - Resultado: a diferença entre **`A`** e **`B`** é igual à diferença entre **`C`** e **`D`**
    - » *Mas **`C`** e **`D`** são diferentes de **`A`** e **`B`***

ATENÇÃO

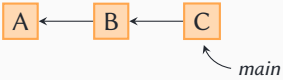
git usa a área de trabalho como “rascunho”
durante o processamento para gerar *revisions* sintéticas!

portanto, a área de trabalho precisa estar “limpa” antes de começar
(área de trabalho = *staging area* = `HEAD`)

Só assim ele pode, ao terminar, “limpar” tudo
e voltar a área de trabalho ao estado anterior

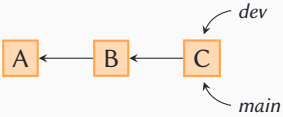
Versões sintéticas

- **Vários comandos geram *revisions* sintéticas**
  - **`git cherry-pick`**
  - **`git pull`**
  - **`git am`**
  - **`git revert`**
  - ...
- **Alguns aplicam as mudanças apenas na área de trabalho atual**
  - Nesses casos, git não faz **`commit`** automaticamente
  - Nesses casos, a área de trabalho *não* precisa estar limpa
    - » ***`git stash pop`***
    - » ***`git switch`*** — *surpresa!*
- **As versões sintéticas mais comuns (e importantes) são *mesclas***
  - **`git merge`** (óbvio)
  - **`git rebase`**



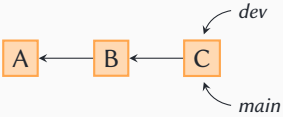
HEAD → **main**

`git branch dev`



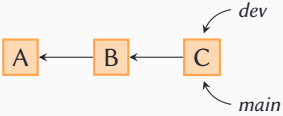
`HEAD` → `main`

```
git branch dev
git switch dev
```



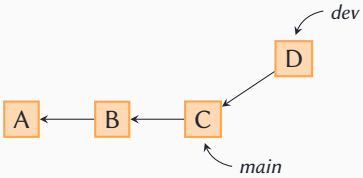
HEAD $\rightarrow$ ~~**main**~~ **dev**

```
git branch dev
git switch dev
```
[...trabalha...]



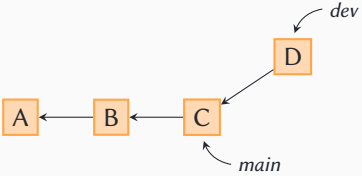
HEAD $\longrightarrow$ **dev**

```
git branch dev
git switch dev
[...trabalha...]
git commit
```

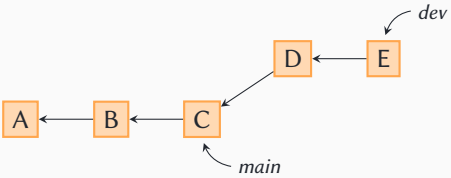


HEAD → **dev**

```
git branch dev
git switch dev
```
[...trabalha...]
```
git commit
```
[...trabalha...]

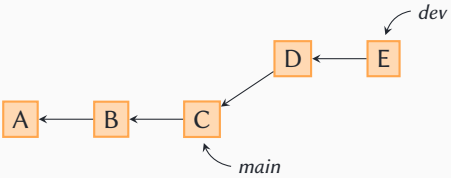


HEAD → **dev**



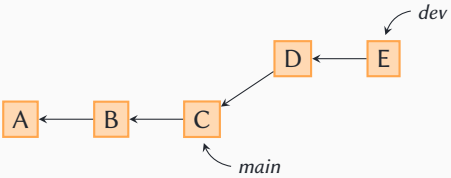
HEAD → **dev**

```
git branch dev
git switch dev
[...trabalha...]
git commit
[...trabalha...]
git commit
```



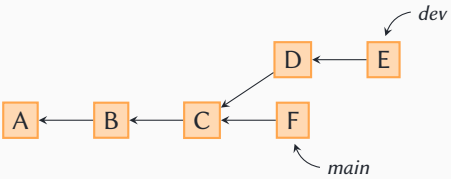
HEAD → ~~**dev**~~ **main**

```
git branch dev
git switch dev
[...trabalha...]
git commit
[...trabalha...]
git commit
git switch main
```



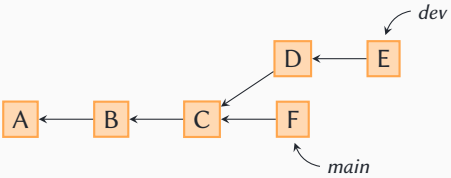
HEAD → **main**

```
git branch dev
git switch dev
[...trabalha...]
git commit
[...trabalha...]
git commit
git switch main
[...trabalha...]
```



HEAD → **main**

```
git branch dev
git switch dev
[...trabalha...]
git commit
[...trabalha...]
git commit
git switch main
[...trabalha...]
git commit
```



HEAD → **main**

git branch dev

git switch dev

[...trabalha...]

git commit

[...trabalha...]

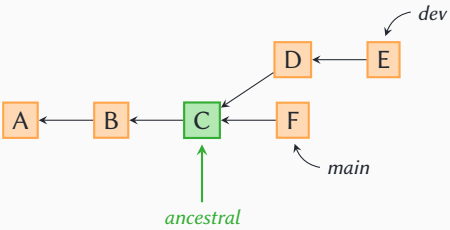
git commit

git switch main

[...trabalha...]

git commit

git merge dev ?



HEAD → **main**

git branch dev

git switch dev

[...trabalha...]

git commit

[...trabalha...]

git commit

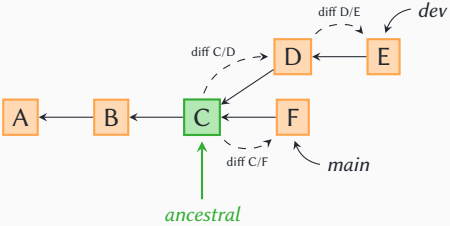
git switch main

[...trabalha...]

git commit

git merge dev ?

Ancestral comum...



HEAD → **main**

git branch dev

git switch dev

[...trabalha...]

git commit

[...trabalha...]

git commit

git switch main

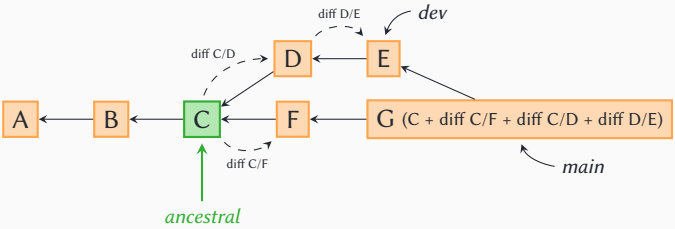
[...trabalha...]

git commit

git merge dev ?

Ancestral comum...

Calcula as diferenças...



HEAD → **main**

git branch dev

git switch dev

[...trabalha...]

git commit

[...trabalha...]

git commit

git switch main

[...trabalha...]

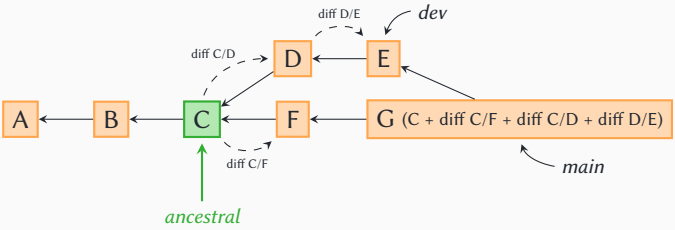
git commit

git merge dev ?

Ancestral comum...

Calcula as diferenças...

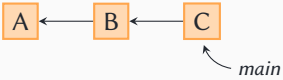
Sintetiza nova *revision*



`HEAD` → `main`

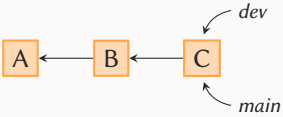
- **G tem dois pais (o histórico inclui uma bifurcação)**
  - **G** "herda" todas as modificações de ambos os pais
    - » *não se esqueça:* ***G*** *é uma revision sintética!*
- **O resultado seria o mesmo se fizéssemos `git merge main` no *branch* `dev`**
  - Mas **`dev`** apontaria para **G** ao invés de **`main`**

`git branch dev`
`git switch dev`
[...trabalha...]
`git commit`
[...trabalha...]
`git commit`
`git switch main`
[...trabalha...]
`git commit`
`git merge dev` ?
Ancestral comum...
Calcula as diferenças...
Sintetiza nova *revision*



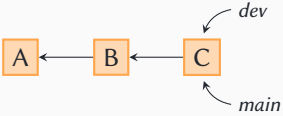
HEAD → **main**

`git branch dev`



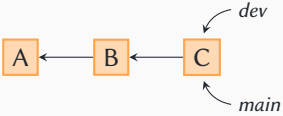
`HEAD` → `main`

```
git branch dev
git switch dev
```



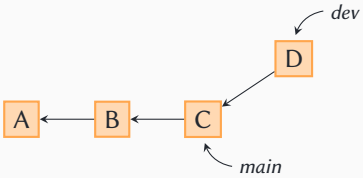
HEAD → ~~**main**~~ **dev**

```
git branch dev
git switch dev
```
[...trabalha...]



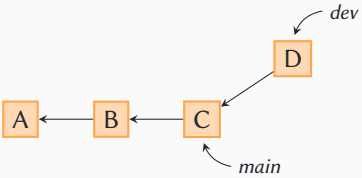
HEAD $\longrightarrow$ **dev**

```
git branch dev
git switch dev
[...trabalha...]
git commit
```

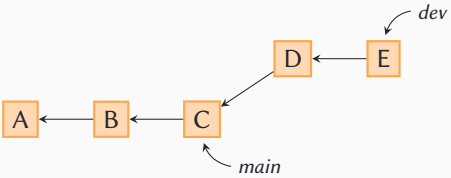


HEAD → **dev**

```
git branch dev
git switch dev
```
[...trabalha...]
```
git commit
```
[...trabalha...]

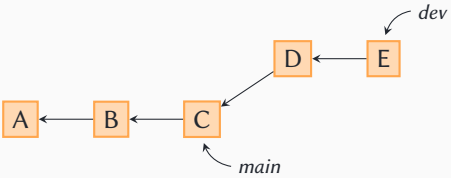


HEAD $\rightarrow$ **dev**



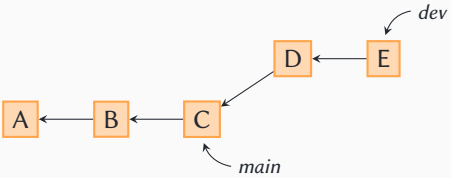
HEAD → **dev**

```
git branch dev
git switch dev
[...trabalha...]
git commit
[...trabalha...]
git commit
```



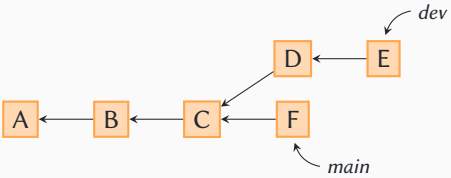
HEAD → ~~**dev**~~ **main**

```
git branch dev
git switch dev
[...trabalha...]
git commit
[...trabalha...]
git commit
git switch main
```



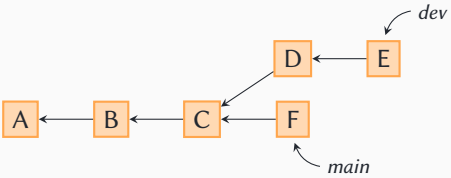
HEAD → **main**

```
git branch dev
git switch dev
[...trabalha...]
git commit
[...trabalha...]
git commit
git switch main
[...trabalha...]
```



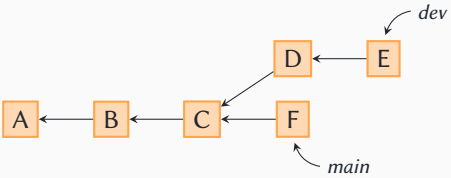
HEAD → **main**

```
git branch dev
git switch dev
[...trabalha...]
git commit
[...trabalha...]
git commit
git switch main
[...trabalha...]
git commit
```



HEAD → ~~**main**~~ **dev**

```
git branch dev
git switch dev
[...trabalha...]
git commit
[...trabalha...]
git commit
git switch main
[...trabalha...]
git commit
git switch dev
```



HEAD $\rightarrow$ **dev**

git branch dev

git switch dev

[...trabalha...]

git commit

[...trabalha...]

git commit

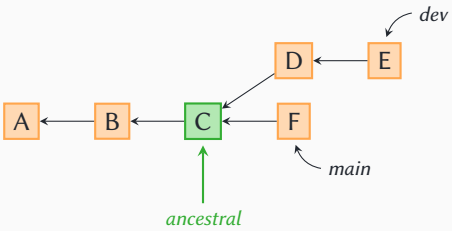
git switch main

[...trabalha...]

git commit

git switch dev

git rebase main ?



HEAD → **dev**

git branch dev

git switch dev

[...trabalha...]

git commit

[...trabalha...]

git commit

git switch main

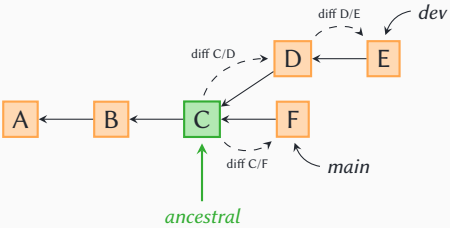
[...trabalha...]

git commit

git switch dev

git rebase main ?

Ancestral comum...



HEAD → **dev**

git branch dev

git switch dev

[...trabalha...]

git commit

[...trabalha...]

git commit

git switch main

[...trabalha...]

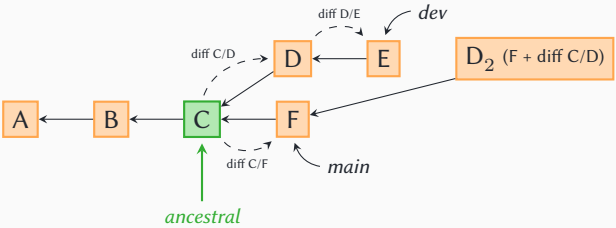
git commit

git switch dev

git rebase main ?

Ancestral comum...

Calcula as diferenças...



HEAD → **dev**

git branch dev

git switch dev

[...trabalha...]

git commit

[...trabalha...]

git commit

git switch main

[...trabalha...]

git commit

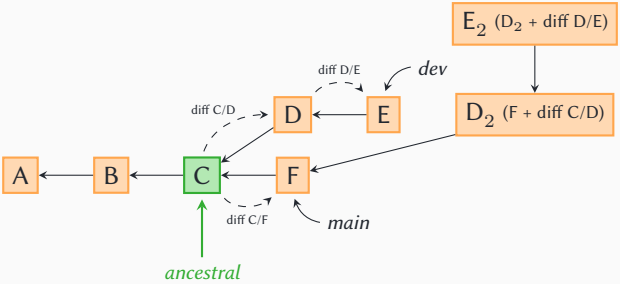
git switch dev

git rebase main ?

Ancestral comum...

Calcula as diferenças...

Sintetiza novas *revisions*



HEAD → **dev**

git branch dev

git switch dev

[...trabalha...]

git commit

[...trabalha...]

git commit

git switch main

[...trabalha...]

git commit

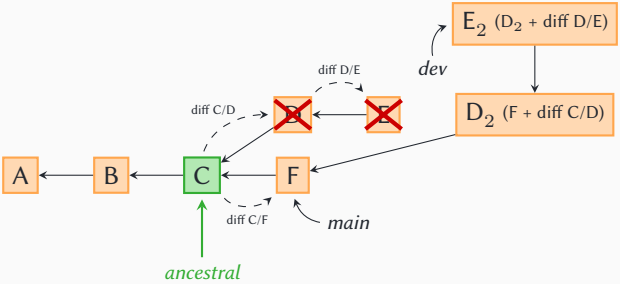
git switch dev

git rebase main ?

Ancestral comum...

Calcula as diferenças...

Sintetiza novas *revisions*



HEAD → **dev**

git branch dev

git switch dev

[...trabalha...]

git commit

[...trabalha...]

git commit

git switch main

[...trabalha...]

git commit

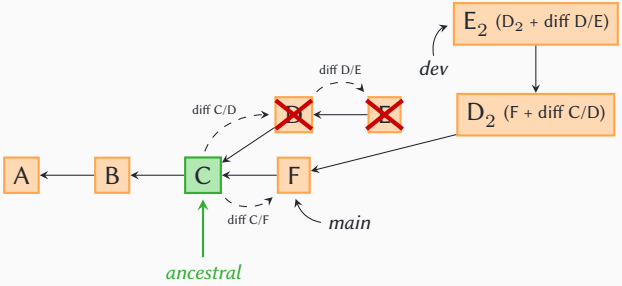
git switch dev

git rebase main ?

Ancestral comum...

Calcula as diferenças...

Sintetiza novas *revisions*



`HEAD` → `dev`

- **E_2 = C + diff C/F + diff C/D + diff D/E (como no merge)**
- **Simula histórico linear (como se fosse só um *branch*)**
  - "Pega **D** e **E** e coloca no final (depois de **F**) ao invés de **C**"
    - » *Nenhum passo (diff) é perdido, mas* D_2 *e* $E_2 \neq$ ***D*** *e* ***E***
- **`git rebase dev` no *branch* `main` resultaria numa *revision* igual, mas com um histórico diferente**
  - ao invés de **F**, D_2, E_2, seria **D**, **E**, F_2

```
git branch dev
git switch dev
[...trabalha...]
git commit
[...trabalha...]
git commit
git switch main
[...trabalha...]
git commit
git switch dev
git rebase main ?
```

Ancestral comum...
Calcula as diferenças...
Sintetiza novas *revisions*

Versões sintéticas

- **Lição de casa**
  1. O que é *fast-forward*?
  2. Quando `git rebase` é uma má ideia?
  3. Execute o comando `git rebase --interactive ID`
     (`ID` → alguma versão do passado)
     Pense em como funciona cada opção que ele oferece

O que é o git

OK, eu menti antes :)

Tudo o que git faz é

- **Criar novas *revisions* (com a informação sobre os antepassados)**
  - (O que envolve copiar arquivos de/para o repositório, o *workdir* e a *staging area*)
- **Manipular a tabela de *heads***

Todos os demais recursos são baseados nesses

O que é o git

OK, eu menti antes :)

Tudo o que git faz é

- **Criar novas *revisions* (com a informação sobre os antepassados)**
  - (O que envolve copiar arquivos de/para o repositório, o *workdir* e a *staging area*)
- **Manipular a tabela de *heads***
- **Fabricar *revisions* sintéticas**

Todos os demais recursos são baseados nesses

O que é o git

OK, eu menti antes :)

Tudo o que git faz é

- **Criar novas *revisions* (com a informação sobre os antepassados)**
  - (O que envolve copiar arquivos de/para o repositório, o *workdir* e a *staging area*)
- **Manipular a tabela de *heads***
- **Fabricar *revisions* sintéticas**
  - O que é essencial!

Todos os demais recursos são baseados nesses

Entrelaçamento quântico

Entrelaçamento quântico: trabalhando através da rede

- **git pode trabalhar via rede (óbvio)**

Entrelaçamento quântico: trabalhando através da rede

- **git pode trabalhar via rede (óbvio)**
  - Recebe e envia *revisions* de/para outros repositórios

Entrelaçamento quântico: trabalhando através da rede

- **git pode trabalhar via rede (óbvio)**
  - Recebe e envia *revisions* de/para outros repositórios
  - Atualiza a tabela de *branches*/*heads* de outros repositórios

Entrelaçamento quântico: trabalhando através da rede

- **git pode trabalhar via rede (óbvio)**
  - Recebe e envia *revisions* de/para outros repositórios
  - Atualiza a tabela de *branches*/*heads* de outros repositórios
  - Mantém cópias das listas de *branches*/*heads* de outros repositórios
    - » *"Remote tracking branches" (branches virtuais)*
    - » *Permitem saber o estado remoto sem usar a rede a cada comando*

Entrelaçamento quântico: trabalhando através da rede

- **git pode trabalhar via rede (óbvio)**
  - Recebe e envia *revisions* de/para outros repositórios
  - Atualiza a tabela de *branches*/*heads* de outros repositórios
  - Mantém cópias das listas de *branches*/*heads* de outros repositórios
    - » *"Remote tracking branches" (branches virtuais)*
    - » *Permitem saber o estado remoto sem usar a rede a cada comando*
- **Só isso!**

Entrelaçamento quântico: quem é quem

- **É preciso "cadastrar" cada repositório remoto**
- **Um repositório remoto cadastrado é um *remote***
- **Em geral só há um *remote*, normalmente chamado *origin***
- **`git remote add <NOME-REPO> <URL>`**
  - **`NOME-REPO`** é o nome que usaremos para acessar o repositório remoto

Repositórios remotos (remotes)

remote	URL	login
adahome	file:///home/ada/trab1	...
alancom	https://alan.com/data	...
charlescom	ssh://charles.com	...
gitlabgrace	ssh://gitlab.com/grace	...
origin	http://gitlab.com/grupo	...

Entrelaçamento quântico: online e offline

- **O mais óbvio seria que os comandos de git (`git log`, `git diff`, `git merge` etc.) fossem capazes de operar com repositórios remotos**
  - "`git log repo-remoto branch-remoto`" etc.
- **Isso teria duas desvantagens**
  - A implementação seria mais complexa
    - » *todos os comandos teriam que ser capazes de operar via rede*
  - Seria necessário estar *online* até para usar `git log` com um repositório remoto
- **Ao invés disso, git conhece o estado do repositório remoto**
  1. Cópia local dos *branches* remotos (*remote tracking branches*)
     - » *Lembrando que um branch é apenas um item em uma tabela*
  2. Cópia da *revision* que está na *head* de cada *branch* remoto
     - » *E de seus antepassados*
- **Apenas 2 comandos precisam operar via rede: `push` e `fetch` (`pull` utiliza `fetch`)**
  - Os comandos em geral (`git log` etc.) operam sem usar a rede

Repositórios remotos (remotes)

remote	URL	login
adahome	file:///home/ada/trab1	...
alancom	https://alan.com/data	...
charlescom	ssh://charles.com	...
gitlabgrace	ssh://gitlab.com/grace	...
origin	http://gitlab.com/grupo	...

Repositórios remotos (remotes)

remote	URL	login
adahome	file:///home/ada/trab1	...
alancom	https://alan.com/data	...
charlescom	ssh://charles.com	...
gitlabgrace	ssh://gitlab.com/grace	...
origin	http://gitlab.com/grupo	...

Ramos/branches

(referências são atualizadas automaticamente)

ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	ba0ddf7f
HEAD	main

Repositórios remotos (remotes)

remote	URL	login
adahome	file:///home/ada/trab1	...
alancom	https://alan.com/data	...
charlescom	ssh://charles.com	...
gitlabgrace	ssh://gitlab.com/grace	...
origin	http://gitlab.com/grupo	...

Ramos/branches

(referências são atualizadas automaticamente)

ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	ba0ddf7f
HEAD	main
adahome/main	953fdf66
alancom/alan	9f9d6c73
charlescom/mypart	cc5ea79e
gitlabgrace/master	6dd6a517
origin/master	ba0ddf7f

Entrelaçamento quântico: online e offline

- **`git fetch NOME-REPO NOME-BRANCH`**
  - Copia o valor (remoto) de **`NOME-BRANCH`** para a tabela (local) de *branches/heads*
    - » *Ou seja, atualiza o remote tracking branch ("branch virtual") correspondente*
    - » *O remote tracking branch de **`REPO-REMOTO NOME-BRANCH`** se chama **`NOME-REPO/NOME-BRANCH`*** 👏

Entrelaçamento quântico: online e offline

- **`git fetch NOME-REPO NOME-BRANCH`**
  - Copia o valor (remoto) de **`NOME-BRANCH`** para a tabela (local) de *branches*/*heads*
    - » *Ou seja, atualiza o remote tracking branch ("branch virtual") correspondente*
    - » *O remote tracking branch de* ***`REPO-REMOTO NOME-BRANCH`*** *se chama* ***`NOME-REPO/NOME-BRANCH`*** 👏
  - Além disso, baixa a *revision* correspondente à *head* e todos seus antepassados

Entrelaçamento quântico: online e offline

- **`git fetch NOME-REPO NOME-BRANCH`**
  - Copia o valor (remoto) de **`NOME-BRANCH`** para a tabela (local) de *branches*/*heads*
    - » *Ou seja, atualiza o remote tracking branch ("branch virtual") correspondente*
    - » *O remote tracking branch de **`REPO-REMOTO NOME-BRANCH`** se chama **`NOME-REPO/NOME-BRANCH`*** 👏
  - Além disso, baixa a *revision* correspondente à *head* e todos seus antepassados
- **`git fetch NOME-REPO`**
  - Faz o mesmo para todos os *branches* de uma vez

Observe que, se você fizer apenas **`git fetch nome-repo nome-branch`**, o repositório local só terá conhecimento sobre o *branch* remoto "**`nome-branch`**"; outros *branches* remotos são desconhecidos até que você faça **`git fetch nome-repo outro-branch`** ou **`git fetch nome-repo`**.

Entrelaçamento quântico: online e offline

- **`git log NOME-REPO/NOME-BRANCH`**
  - Com **`git log`** tem **`/`**, com **`git fetch`** não!
    - » *Porque **`git log`** se refere ao remote tracking branch, não ao repositório remoto*
- **Ou seja**
  - Se o comando faz uma operação de rede (apenas **`fetch`**, **`pull`** e **`push`**), é preciso indicar o repositório e o *branch*: **`NOME-REPO NOME-BRANCH`**
  - Se o comando se refere a um *branch* local ou a um *remote tracking branch* (**`log`**, **`diff`** etc.), é preciso dizer o nome do *branch*: **`NOME-BRANCH`** (ramo local) ou **`NOME-REPO/NOME-BRANCH`** (*remote tracking branch*)
- **De novo: *remote tracking branches* simplificam a implementação**
  - A maioria dos comandos (**`log`**, **`diff`**, **`merge`** etc.) só precisa operar localmente
    - » *remote tracking branches são simplesmente itens na tabela de branches/heads*
  - Apenas **`fetch`** e **`push`** precisam saber como operar via rede
- **Além disso, permitem fazer as operações comuns (`log`, `diff` etc.) offline**

Entrelaçamento quântico: fluxo de trabalho

- **Não é possível trabalhar diretamente em um *branch* remoto**
  - Não existe "`git commit repo-remoto`" etc.
- ***Remote tracking branches* → apenas operações de leitura (`log` , `diff` etc.)**
  - São *branches* virtuais que apenas representam o estado remoto
- **Qual é o fluxo de trabalho?**
  - Criamos um novo *branch* local
  - Copiamos as alterações *branch* local ↔ *branch* remoto
    - » *Com `push` e `fetch`, os únicos comandos que operam via rede*

Entrelaçamento quântico: fluxo de trabalho

- **`git branch <NOME> repo-remoto/branch`**
  - Cria um novo *branch* a partir do *remote tracking branch*

- **remoto → local: `git pull repo-remoto branch`**
  - significa **`fetch repo-remoto branch`** + **`merge repo-remoto/branch`** (incluindo **`commit`**)
  - Não é preciso especificar o *branch* local
    - » *Porque* **`merge`** *usa o workdir → a mescla sempre acontece no branch atual*

- **local → remoto: `git push repo-remoto branch-local:branch-remoto`**
  - Atualiza a *head* **`branch-remoto`** para ser igual a **`branch-local`**
  - Envia a *revision* que está na *head* de **`branch-local`** e seus antepassados
  - Não é preciso que **`branch-local`** seja o *branch* atual
    - » *Porque especificamos no comando os branches de origem e destino*

A “ida” e a “volta” são diferentes
e as sintaxes são diferentes!

Entrelaçamento quântico: hello, goodbye

- **Só uma "ida" (`push`) mas duas "voltas" (`fetch` e `pull`)**
- **Com `push`, a `head` remota é modificada diretamente**
  - Sempre é algo similar a um *fast-forward* remoto
  - pode ou não usar "`:`" para indicar a relação local → remoto
- **Com `pull`, há um `merge` e um `commit` implícitos no *branch* atual**
  - (pode ser um `merge` comum, *fast-forward* ou `rebase` com `git pull -r`)
  - A `head` local é modificada porque há um `commit`
  - nunca usa "`:`", pois o *branch* local sempre é o atual
- **`git fetch repo-remoto branch-remoto` não usa "`:`"**
  - Não modifica nenhum *branch* local, apenas memoriza o *status* remoto (*remote tracking branch*)
- **Não existe `git fetch repo-remoto branch-remoto:branch-local` ?**
  - Existe, mas você **nunca** deve usar esse comando
    - » *Use `merge`, `reset` etc. com o remote tracking branch correspondente (`repo-remoto/branch`) ou `pull`*

Trabalhando através da rede: atalhos

- **Lição de casa**
  - O que é um repositório *bare*?
  - Qual o problema com **`push`** em um repositório que não é *bare*? Por quê?

Trabalhando através da rede: atalhos

- **É possível copiar de/para quaisquer *branches* locais/remotos**
  - Mas, em geral, um *branch* local tem exatamente um *branch* remoto correspondente
  - Esse *branch* remoto é o **fornecedor** (*upstream*)
    - » *Não é obrigatório que haja um upstream*
    - » *Serve principalmente para economizar na hora de digitar :)*
- **Se o *upstream* é remoto, usamos o *remote tracking branch* correspondente**
  - Mas git identifica o repositório remoto real quando necessário
    - » *OK, na verdade é mais complexo, mas faz de conta que é assim*
- **`git branch --track <NOME> repo-remoto/branch`**
  - A opção **`--track`** registra repo-remoto/branch (um *remote tracking branch*) como *upstream* do novo *branch* NOME
    - » *A palavra **`--track`** não tem nada a ver com remote tracking branches!* 
    - » *É possível (e comum) usar **`--track`** para registrar um branch comum como upstream de outro*
  - Para mudar o *upstream* de um *branch* já existente, use **`--set-upstream-to`**

Repositórios remotos (remotes)

remote	URL	login
adahome	file:///home/ada/trab1	...
alancom	https://alan.com/data	...
charlescom	ssh://charles.com	...
gitlabgrace	ssh://gitlab.com/grace	...
origin	http://gitlab.com/grupo	...

Ramos/branches

(referências são atualizadas automaticamente)

ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	ba0ddf7f
HEAD	main
adahome/main	953fdf66
alancom/alan	9f9d6c73
charlescom/mypart	cc5ea79e
gitlabgrace/master	6dd6a517
origin/master	ba0ddf7f

Repositórios remotos (remotes)

remote	URL	login
adahome	file:///home/ada/trab1	...
alancom	https://alan.com/data	...
charlescom	ssh://charles.com	...
gitlabgrace	ssh://gitlab.com/grace	...
origin	http://gitlab.com/grupo	...

Ramos/branches

(referências são atualizadas automaticamente)

ramo	head	fornecedor (upstream)
ada	953fdf66	adahome/main
alan	9f9d6c73	alancom/alan
charles	cc5ea79e	charlescom/mypart
grace	6dd6a517	gitlabgrace/master
main	ba0ddf7f	origin/master
HEAD	main	—
adahome/main	953fdf66	—
alancom/alan	9f9d6c73	—
charlescom/mypart	cc5ea79e	—
gitlabgrace/master	6dd6a517	—
origin/master	ba0ddf7f	—

Trabalhando através da rede: atalhos

- **`git status`**
  - "Your branch is ahead of 'origin/main' by 4 commits."
- **Recebendo explicitamente**
  - **`git pull repo-remoto branch`**
    - » *Sempre no branch atual!*
- **Recebendo implicitamente do *upstream***
  - **`git pull`**
    - » *Sempre no branch atual!*
- **Enviando explicitamente**
  - **`git push nome-repo branch-local:branch-remoto`**
- **Enviando implicitamente para o *upstream***
  - **`git push`**
    - » *Envia o branch atual para o upstream*
    - » *Por segurança, só funciona se o nome do branch remoto for igual ao local*

Colaborando

- **Em geral, você não começa um projeto do zero**
- **Ao invés disso, você vai interagir com um repositório existente**
  Isso vale também para repositórios que você criar em sítios como github ou gitlab
  - `git init .`
  - `git add remote origin <URL>`
  - `git fetch origin`
  - `git branch --track main origin/main`
  - `git switch main`
    - `main` *(antigamente,* `master`*) é o branch default*
- **Vamos facilitar?**
  - `git clone <URL> .`
    - *Corresponde exatamente à sequência de comandos acima*

Colaborando

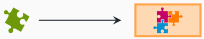
Depois de fazer `clone` etc., como enviar suas mudanças?

- **Você pode fazer `merge` e `push` das mudanças diretamente...**
- **Ou pode pedir para alguém revisar e fazer `merge`**
  - "pull request"
    - » *Publica o branch com a alteração e solicita que alguém faça `pull`*
- **Útil para facilitar a revisão, discussão, melhorias etc. antes do merge**
- **Inescapável se você não tem poderes de escrita no repositório**
  - Pode ser um simples email: "Dê uma olhada no repositório X, branch Y"
  - Pode ser um email automático gerado com `git request-pull`
  - Pode ser um email usando `git format-patch` (avançado)
  - Em projetos que usam github ou gitlab, há uma interface web para isso

O que é o git

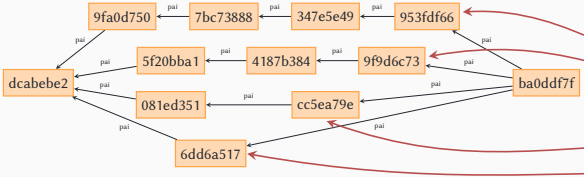
Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)



Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

(referências são atualizadas automaticamente)

ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
main	ba0ddf7f
HEAD	main

O que é o git

Tudo o que git faz é

- **Criar novas *revisions* (com a informação sobre os antepassados)**
  - (O que envolve copiar arquivos de/para o repositório, o *workdir* e a *staging area*)
- **Manipular a tabela de *heads***
- **Fabricar *revisions* sintéticas**
  - O que é essencial!

Todos os demais recursos são baseados nesses

Exemplo: *tags*

***tags* não são novidade:**

Etiquetas/tags
(referências estáticas)

etiqueta	ID
v1beta	ccfea1ee
v1	752bdfc6
v2	4f5dac23

É a mesma ideia que a tabela de *branches*/*heads*!

Qual a diferença?

- *tags* são fixas
- *heads* são atualizadas automaticamente a cada `commit`

O que é o git

Tudo o que git faz é

- **Criar novas *revisions* (com a informação sobre os antepassados)**
  - (O que envolve copiar arquivos de/para o repositório, o *workdir* e a *staging area*)
- **Manipular a tabela de *heads***
- **Fabricar *revisions* sintéticas**
  - O que é essencial!

Todos os demais recursos são baseados nesses