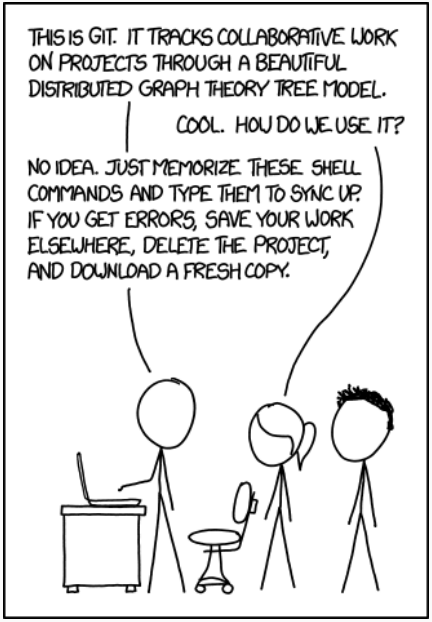
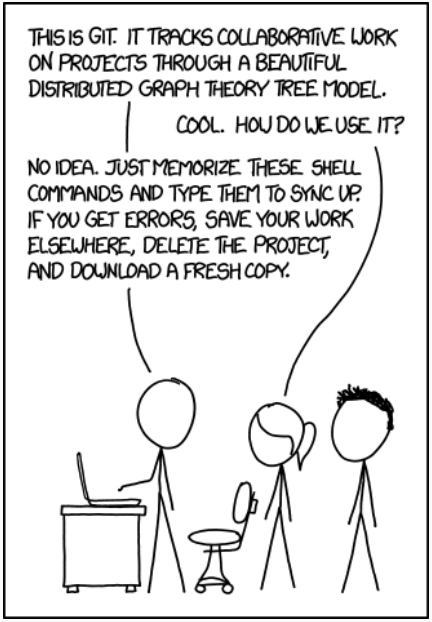
Parte II

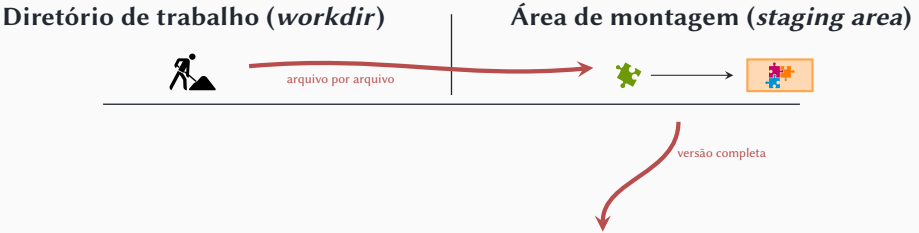
Entendendo o git

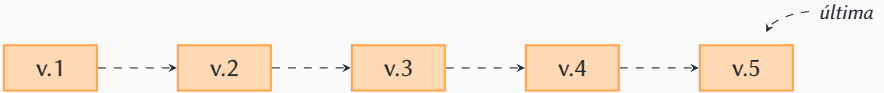

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.



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...

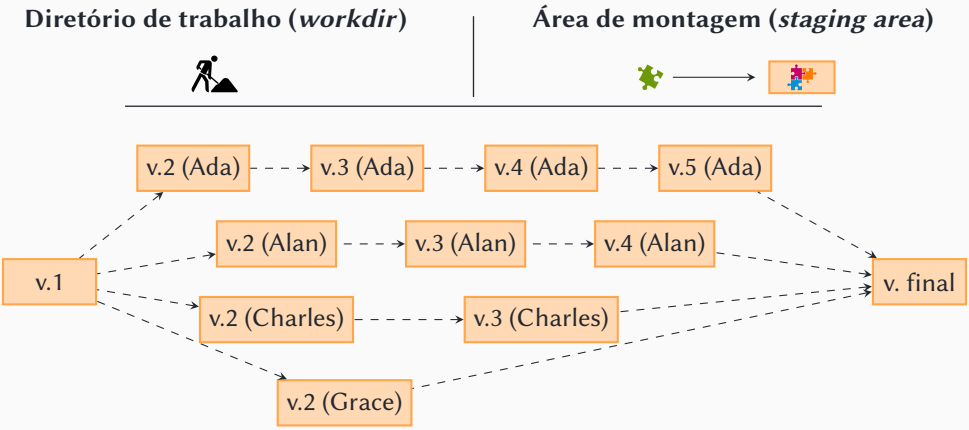
- **A área de montagem nunca está vazia!**
  - Ela sempre tem uma cópia da "última foto", ou seja, da última versão
  - Não é preciso fazer **`git add`** com arquivos que não mudaram, só o que efetivamente mudou
- **E como remover um arquivo da área de montagem?**
  - **`git rm arquivo`**
    - *Apaga da área de trabalho também!*
    - ***Não é** o contrário de **`git add`**!*

De onde vêm esses IDs de cada *revision*?

Por que não é v.1 → v.2 → v.3 etc.?

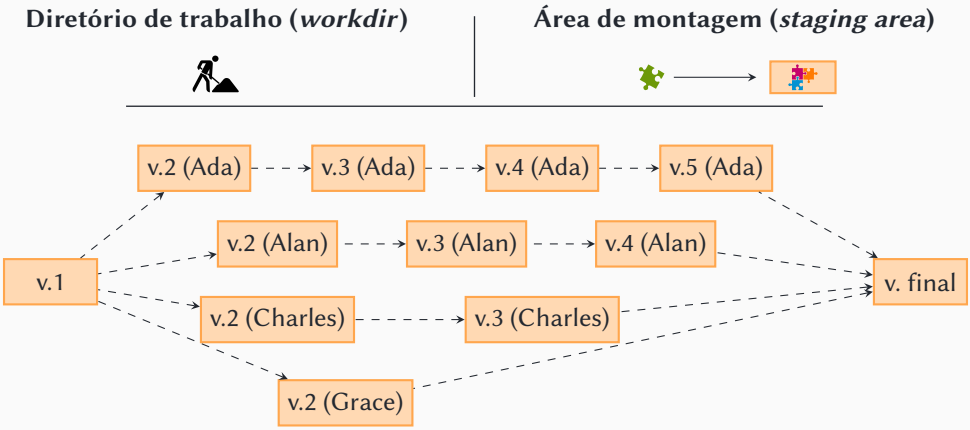
Multiverso: o mundo não é linear

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



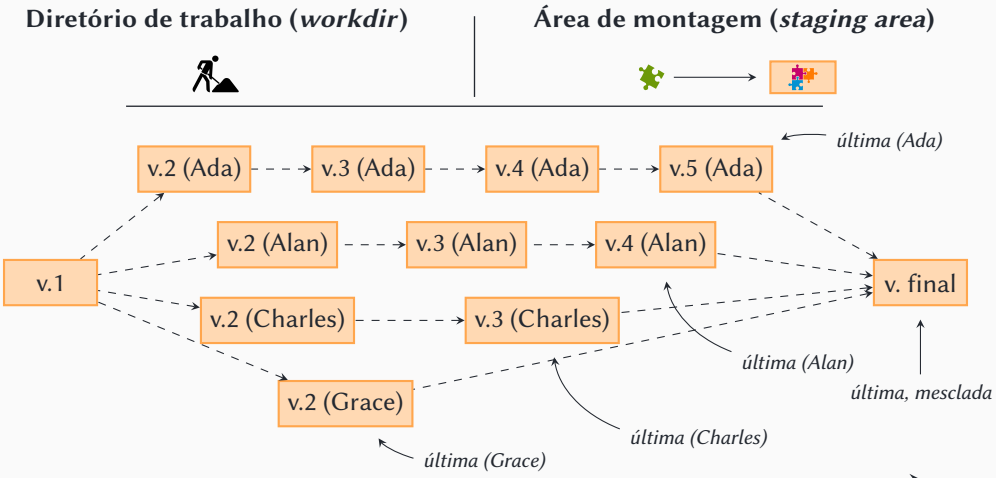
“Cada um faz sua parte e depois a gente junta tudo”

- Cada um vê uma linha do tempo diferente!



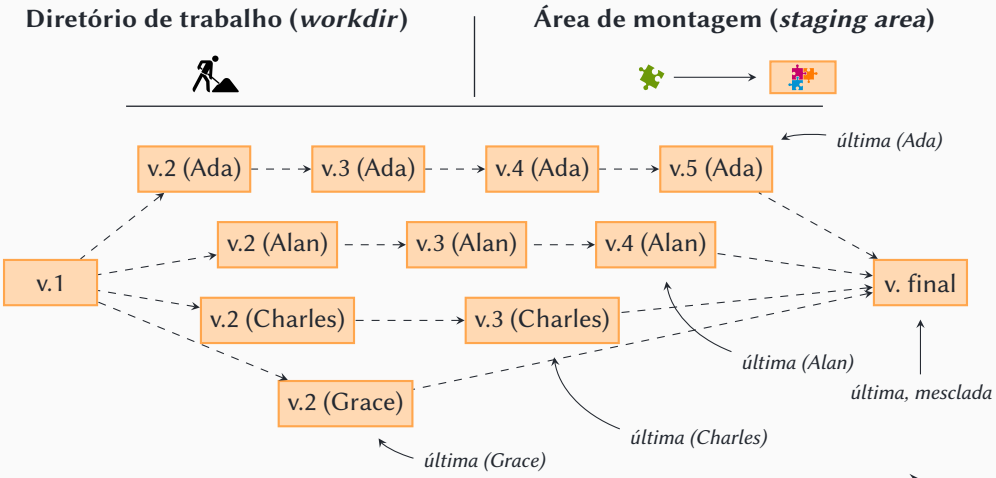
“Cada um faz sua parte e depois a gente junta tudo”

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



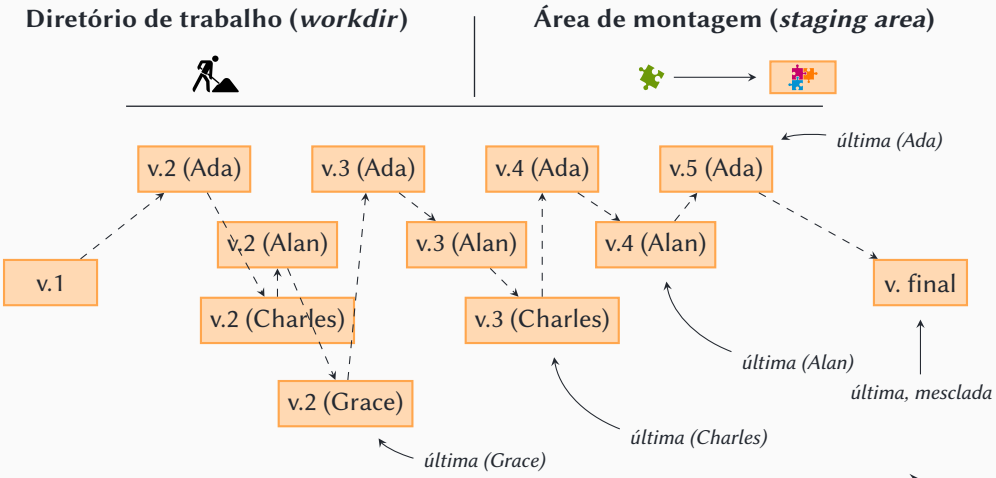
“Cada um faz sua parte e depois a gente junta tudo”

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



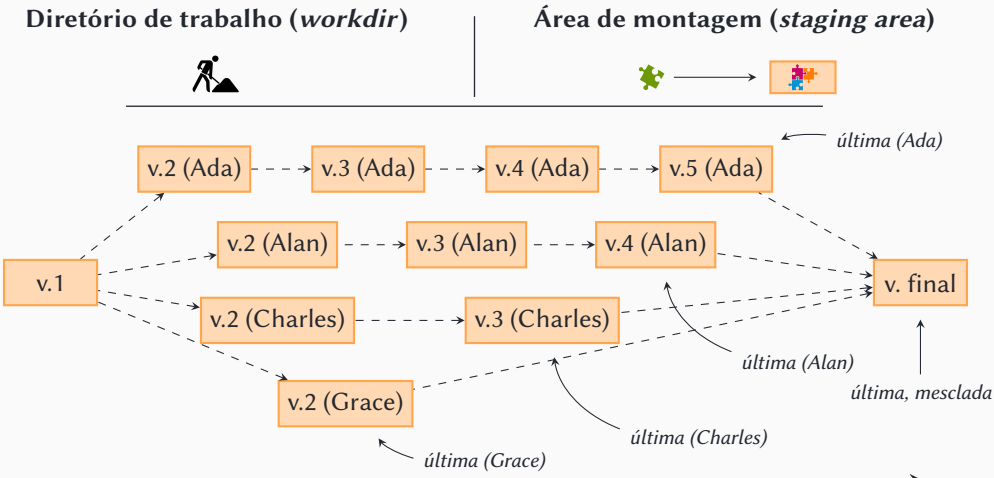
“Cada um faz sua parte e depois a gente junta tudo”

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

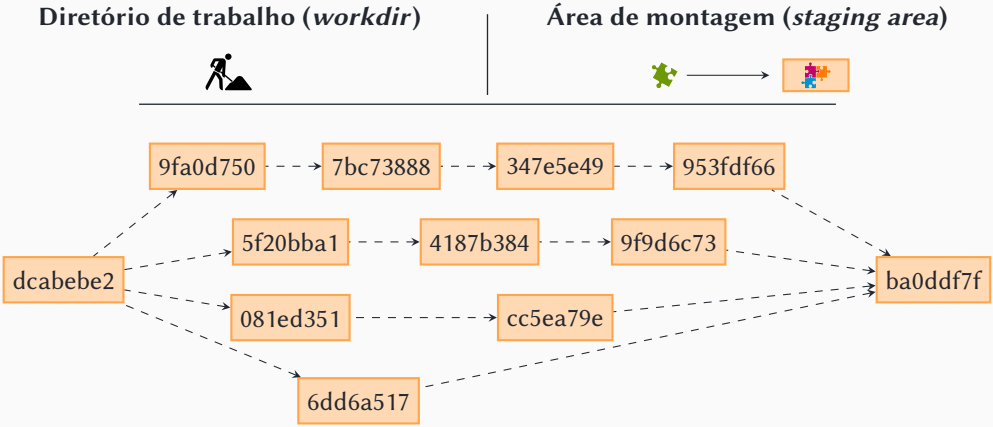


Não é possível inferir uma ordem entre as versões!

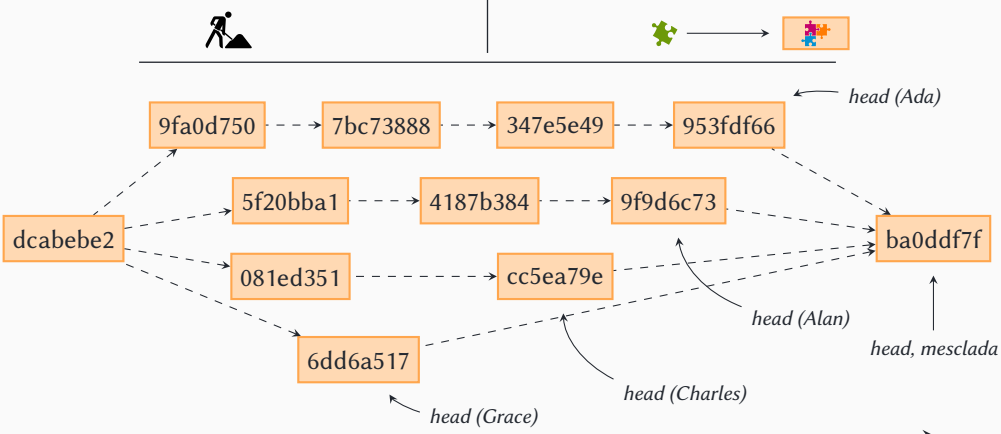
- É preciso explicitar a sequência
- IDs não são confiáveis (podem ser repetidos, significado indefinido)



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

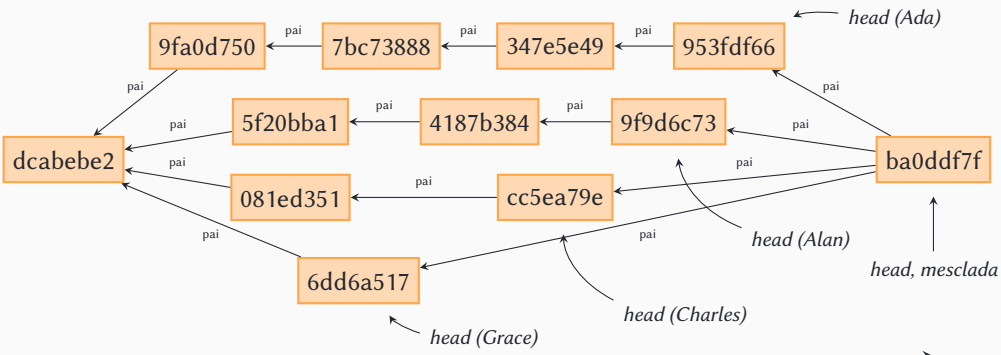


- **IDs aleatórios (não exatamente)**
  - Unicidade
- **Cada "última" versão → *head***



- **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*

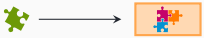
Diretório de trabalho (*workdir*) | **Área de montagem (*staging area*)**



O que é o git

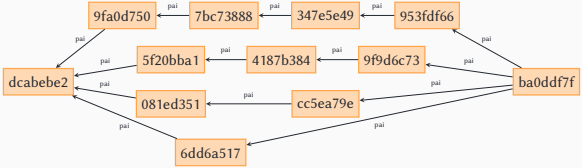
Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)



Depósito de versões encadeadas

(grafo de versões)



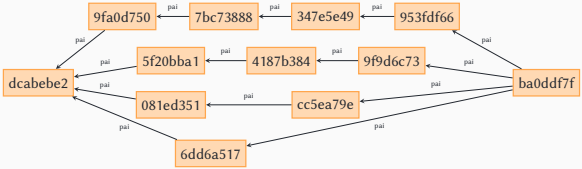
O que é o git

Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)

Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

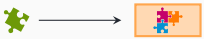
(referências são atualizadas automaticamente)

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

O que é o git

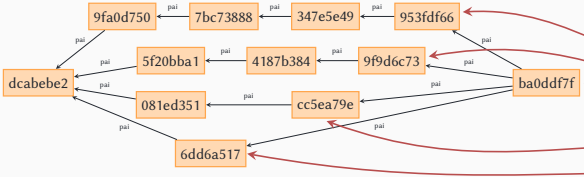
Diretório de trabalho (*workdir*)

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
master	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*

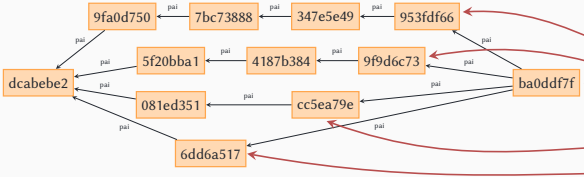
- `git commit --amend`

Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)

Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

(referências são atualizadas automaticamente)

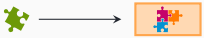
ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
master	ba0ddf7f

Manipulando a tabela de *heads*

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

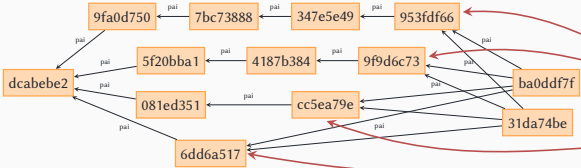
Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)



Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

(referências são atualizadas automaticamente)

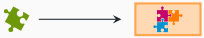
ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
master	ba0ddf7f

Manipulando a tabela de *heads*

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

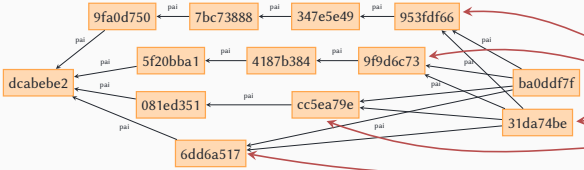
Diretório de trabalho (*workdir*)

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
master	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: **`master`** ou **`main`**
- **`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?

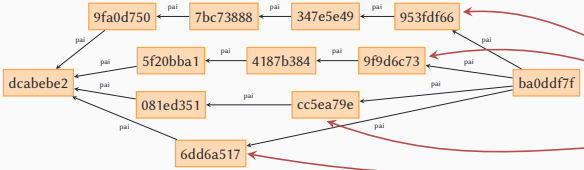
- **Se temos vários ramos...**
  - Qual versão temos no *workdir* e na *staging area*?
  - Quando digo **`commit`**, qual *branch*/*head* vai ser atualizado?
  - Quando digo **`log`**, vou ver o histórico de qual *branch*?
  - Quando digo **`diff`**, vamos comparar quem com quem?
- **Oncotô? (O que é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

Diretório de trabalho (*workdir*)

Área de montagem (*staging area*)

Depósito de versões encadeadas

(grafo de versões)



Ramos/branches

(referências são atualizadas automaticamente)

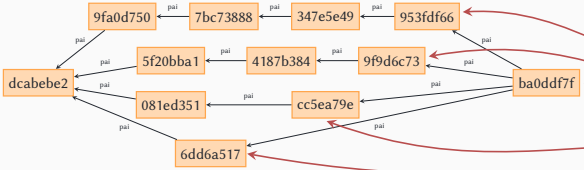
ramo	head
ada	953fdf66
alan	9f9d6c73
charles	cc5ea79e
grace	6dd6a517
master	ba0ddf7f

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
master	ba0ddf7f
HEAD	master

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`**)
    - » *Atualiza o workdir e a staging area*
    - » *Faz HEAD* → ***`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`**)
    - » *Atualiza o workdir e a staging area*
    - » *Faz HEAD* → ***`grace`***
- **Lição de casa**
  - **`git switch --detached ID`** quando ID não é uma *head*
    - » *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

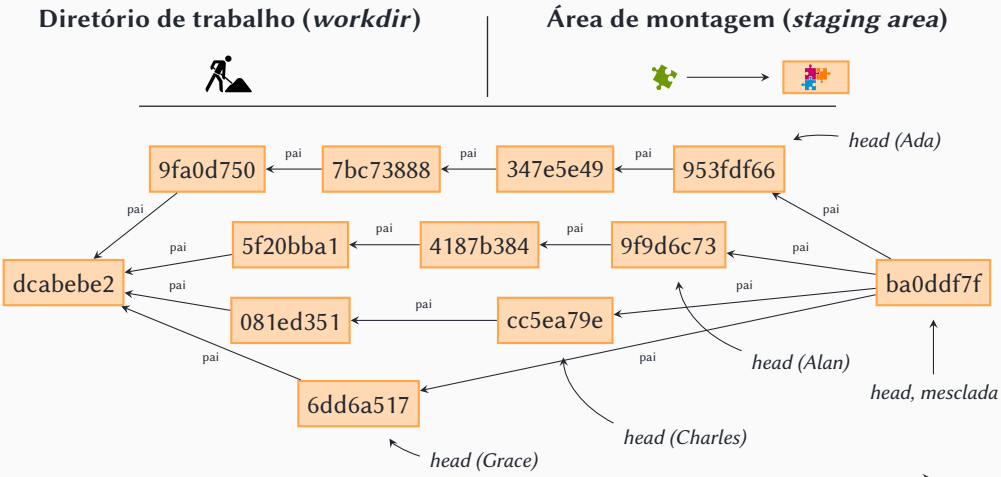
- **Você acrescentou indevidamente uma funcionalidade que só funciona no Linux**
  - Opção 1: Cria um novo *branch* diferente (“multiplataforma”) sem essa alteração
  - Opção 2: Continua “em frente”: cria uma nova *revision* que “des-modifica” o que foi feito
    - » *Não é uma “viagem no multiverso”,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

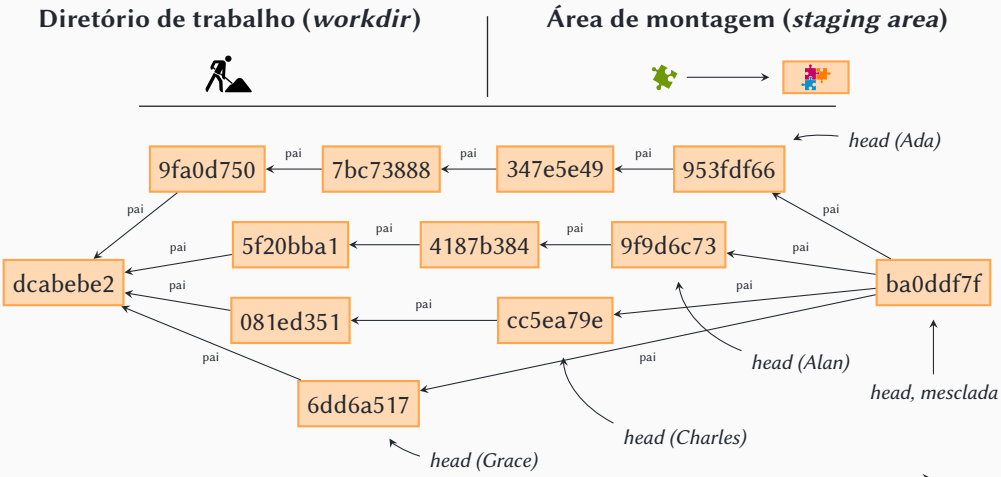
- **Você acrescentou indevidamente uma funcionalidade que só funciona no Linux**
  - Opção 1: Cria um novo *branch* diferente (“multiplataforma”) sem essa alteração
  - Opção 2: Continua “em frente”: cria uma nova *revision* que “des-modifica” o que foi feito
    - » *Não é uma “viagem no multiverso”, 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)**
  - O próximo **`git commit`** continua a história normalmente
  - É possível inclusive fazer outras modificações no arquivo antes de **`git commit`**
  - **`git restore --source ID arquivo`**: copia **`arquivo`** da *revision* **`ID`** para a área de trabalho

Versões sintéticas e mesclas

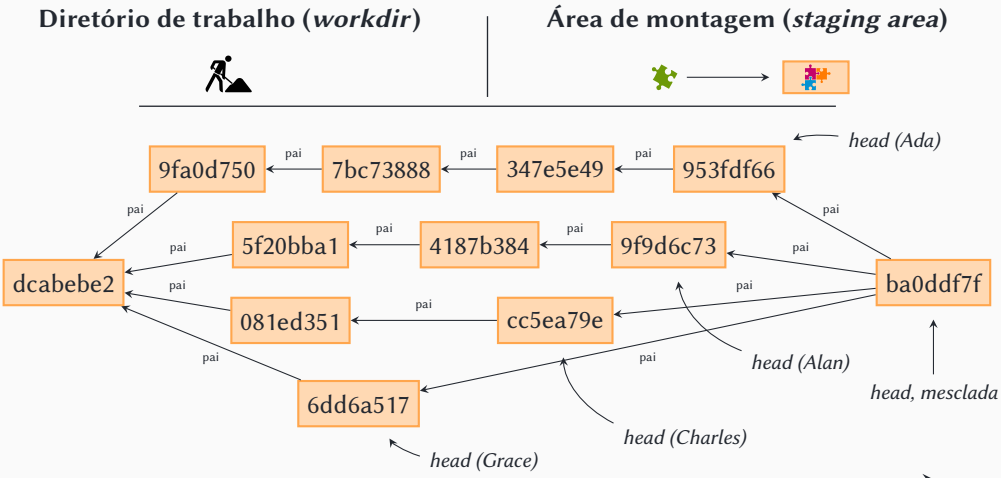
- Quem criou a *revision* `ba0ddf7f`?



- **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**
- **git gera *revisions* sintéticas de várias formas**
  - **`git rebase`**
  - **`git cherry-pick`**
  - **`git format-patch`** / **`git am`**
  - **`git revert`**
  - ...
- **A mais comum (e importante) são *mesclas***
  - **`git merge`** (óbvio)
  - **`git stash pop`**
  - **`git pull`**
  - **`git switch`** — surpresa!
- **Lição de casa**
  - O que é *fast-forward*?

ATENÇÃO

git usa a área de trabalho para gerar *revisions* sintéticas!

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

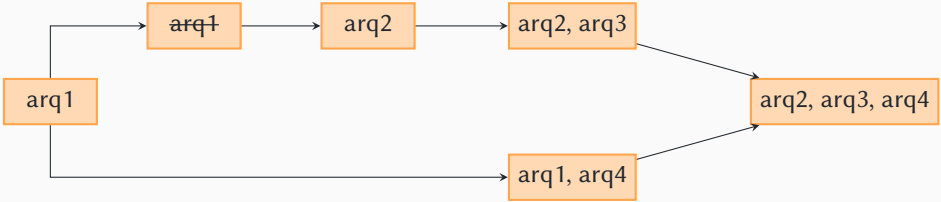
OK, eu menti antes :)

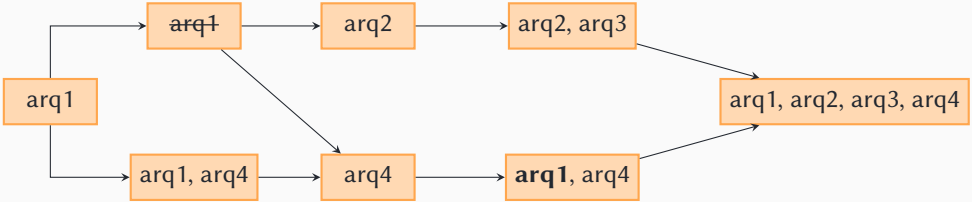
Tudo o que git faz é

- **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!

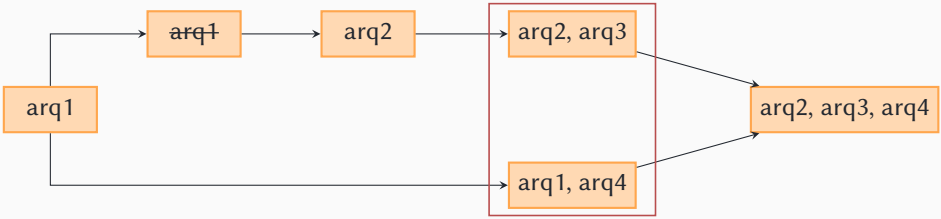
Todos os demais recursos são baseados nesses

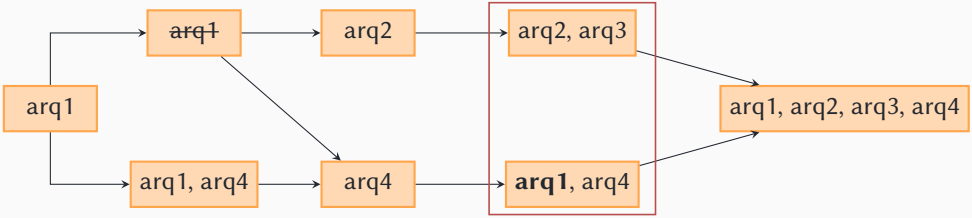
- **git utiliza o histórico ao mesclar**





- **git utiliza o histórico ao mesclar**





Fluxo de trabalho

- **Lição de casa**
  - Aprenda como funciona **`git rebase`**
    - » *Uma boa explicação: git-scm.com/book/en/v2/Git-Branching-Rebasing*

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"*

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"*
- **Só isso!**

Entrelaçamento quântico: trabalhando através da rede

- **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*
  - `git fetch NOME-REPO`
    - » *Atualiza a cópia (local) da tabela de branches/heads de* ***NOME-REPO***
    - » *Baixa as revisions de cada histórico correspondente*
  - `git log NOME-REPO/master`

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
master	ba0ddf7f
HEAD	master

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
master	ba0ddf7f
HEAD	master
adahome/main	953fdf66
alancom/alan	9f9d6c73
charlescom/mypart	cc5ea79e
gitlabgrace/master	6dd6a517
origin/main	ba0ddf7f

Entrelaçamento quântico: trabalhando através da rede

- **Não é possível trabalhar diretamente em um *branch* remoto**
  - (Mas é possível fazer **`git log`**, **`git diff`** etc.)
- **Ao invés disso**
  - Criamos um novo *branch* local
  - Copiamos as alterações *branch* local ↔ *branch* remoto
- **`git branch <NOME> repo-remoto/branch`** / **`git switch <NOME>`**
- **`git pull repo-remoto branch`**
  - **`git pull`** faz **`fetch`** automaticamente
  - Não é preciso especificar o *branch* local
    - » *Porque* ***`git pull`*** *realiza um* ***`merge`*** *e usa o workdir → a mescla acontece no branch atual*
- **`git push repo-remoto branch-local:branch-remoto`**
  - Envia as novas *revisions* de **`branch-local`**
  - Atualiza a *head* **`branch-remoto`** para ser igual a **`branch-local`**
  - Não é preciso que **`branch-local`** seja o *branch* atual
    - » *Porque especificamos no comando os branches de origem e destino*

Entrelaçamento quântico: trabalhando através da rede

- **A "ida" e a "volta" são diferentes e as sintaxes são diferentes!**
  - Com **`pull`**, há um **`merge`** e um **`commit`** implícitos no *branch* atual
    - » *(mas pode ser um fast-forward)*
    - » *A* **`head`** *local é modificada porque há um* **`commit`**
    - » **`git pull -r`** *é similar, mas faz* **`rebase`** *ao invés de* **`merge`**
  - Com **`push`**, a **`head`** remota é modificada diretamente
    - » *Sempre é algo similar a um fast-forward remoto*
- **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 :)*
  - `git branch --track <NOME> repo-remoto/branch` / `git switch <NOME>`
    - » `--track` *registra repo-remoto/branch como upstream do novo branch NOME*
    - » *Não tem nada a ver com remote tracking branches!*

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
master	ba0ddf7f
HEAD	master
adahome/main	953fdf66
alancom/alan	9f9d6c73
charlescom/mypart	cc5ea79e
gitlabgrace/master	6dd6a517
origin/main	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
master	ba0ddf7f	origin/main
HEAD	master	—
adahome/main	953fdf66	—
alancom/alan	9f9d6c73	—
charlescom/mypart	cc5ea79e	—
gitlabgrace/master	6dd6a517	—
origin/main	ba0ddf7f	—

Trabalhando através da rede: atalhos

- `git status`

- **Enviando sem *upstream* definido**
  - `git push nome-repo branch-local:branch-remoto`
- **Enviando com *upstream* definido**
  - `git push nome-repo branch-local`
    - » *Se o branch atual for o branch a ser enviado, basta* `git push`
- **Recebendo sem *upstream* definido**
  - `git pull repo-remoto branch`
    - » *Sempre no branch atual!*
- **Recebendo com *upstream* definido**
  - `git pull`
    - » *Sempre no branch atual!*

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 master origin/master`
  - `git switch master`
    - » *master é o branch default, então este passo não é realmente necessário*
- **Vamos facilitar?**
  - `git clone <URL> .`

Colaborando

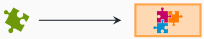
Depois de fazer `clone` etc., como enviar suas mudanças?

- **Você pode fazer `merge` e `push` das mudanças diretamente...**
- **Ou pode pedir para alguém revisar e fazer `merge`**
  - "pull request"
  - 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

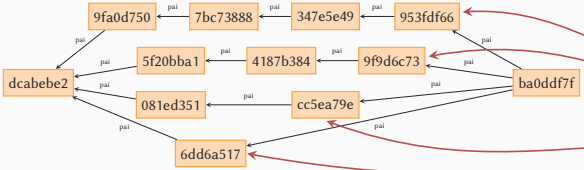
Diretório de trabalho (*workdir*)

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
master	ba0ddf7f
HEAD	master

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

Por exemplo, *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