Git Over Heeeeeeere!

Lucas Fialho Zawacki

Sobre mim

- Cientista da Computação 9º semestre (quase lá)
- Ex-bolsista do Grupo PET Computação
- Participação como estudando no Google Summer of Code 2011 e 2012
  - Wine: http://www.winehq.org/
- Coding Dojo UFRGS
- Hackerspace Matehackers

Esta palestra

- Introdução ao versionamento de código
- Motivação para um sistema como o Git
- Decisões chave de projeto do Git
- Lista de comandos úteis
- Demonstrações práticas
- Dicas úteis

Git

- Criado por Linus Torvalds
- Substituto para o BitKeeper
- É usado largamente no desenvolvimento do Linux
- Muito popular atualmente

Motivacional

- Inspirado por The Git Parable [1]
- Um programador, um computador, um destino
- Escrever e gerenciar o código de um software complicado

[1] : http://tom.preston-werner.com/2009/05/19/the-git-parable.html

Tentativa 1 - Snapshots

Tentativa 1 - Snapshots

copia-1

Código e arquivos iniciais.

copia-2

Updates automáticos no twitter.

copia-3

Banco de dados com yaml.

...

copia-42

Versão 1.0 rodando no servidor.

Tentativa 1 - Análise

- Vantagens
  - Simples
  - Ordenação fácil (nome ou data dos arquivos)
- Desvantagens
  - Exige significativo esforço do usuário
  - Desenvolvimento é sempre linear?

Tentativa 1 - Problema

copia-42

copia-43

copia-44

Versão 1.0 rodando no servidor.

gettimeofday função maldita

Nunca programe e dirija.

Tentativa 1 – Bugs!

copia-42

copia-43

copia-44

Versão 1.0 rodando no servidor.

gettimeofday função maldita

Nunca programe e dirija.

Tentativa 1 – Bugs!

copia-42

Versão 1.0 rodando no servidor.

copia-43

gettimeofday função maldita

copia-44

Nunca programe e dirija

copia-??

Corrigido exploit que permitia a lobotomia não autorizada do usuário

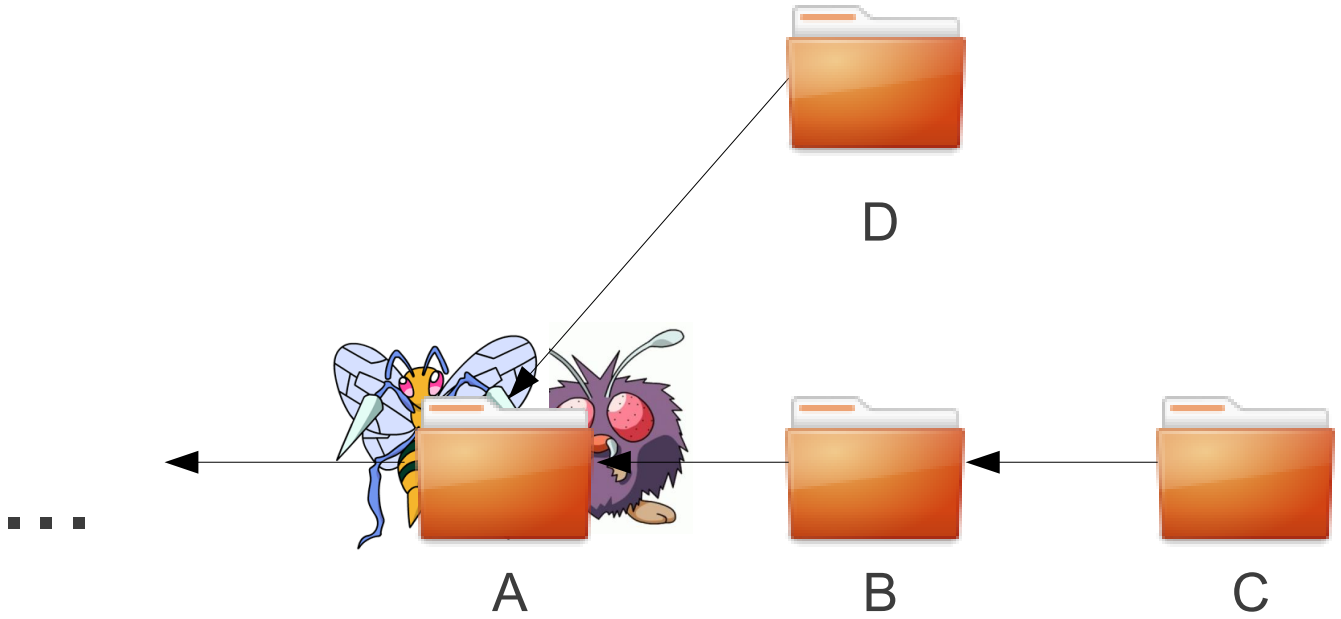
Tentativa 1 - Problema

- Qual o nome da próxima versão?
  - copia-43
  - copia-45
  - copia-43-2
  - Nenhuma das acima
  - Todas acima

Tentativa 2 – Branches

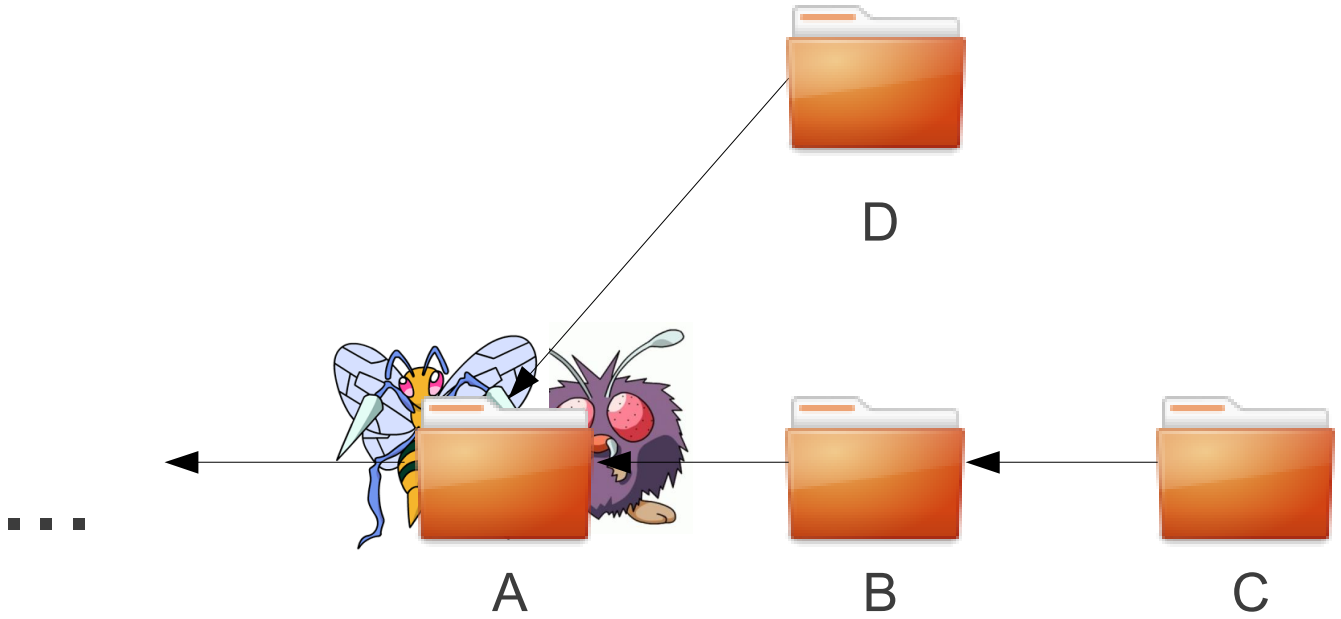
- Nome da cópia não importa
- Usar representação em forma de árvore
  - Cada raiz é a 'última versão do código'
  - Cada versão aponta para sua versão anterior

Tentativa 2 - Branches



Tentativa 2 – Nomes são importantes

Branches
→ C
D

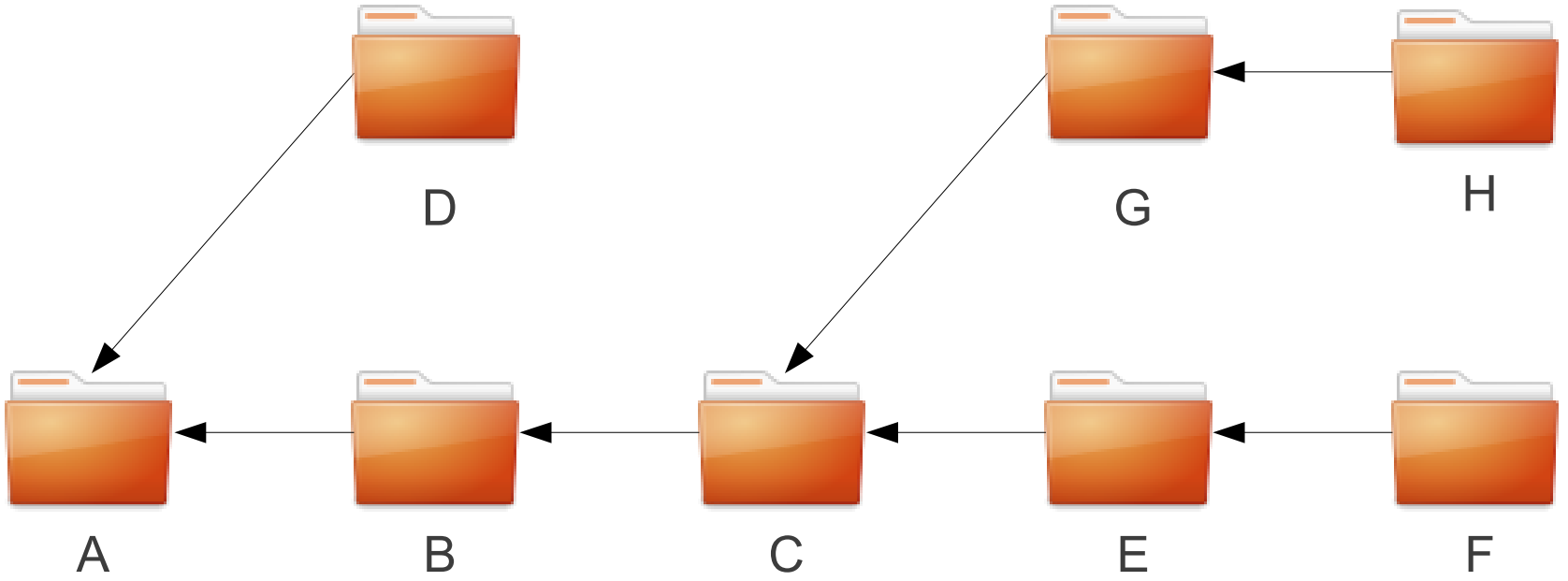


Tentativa 2 - Análise

- Vantagens
  - Mais organizado
  - Versões alternativas do código
  - Não é realmente complicado. O “ponteiro” para o pai pode ser guardado junto com a mensagem.
- Desvantagens
  - Precisamos de uma maneira para juntar o trabalho feito em diferentes branches

Tentativa 2 – Efeito Colateral

Branches
→ F (master)
D (1.1)
H (novo-bd)



Recapitulando

- Versões ordenadas temporalmente de todos estágios de desenvolvimento
- Branches de desenvolvimento alternativos
- Dados em uma versão
  - Código
  - Nome
  - Nome da versão anterior
  - Mensagem

Mais usuários

- Chegamos a um ponto em que queremos trabalhar em conjunto com outros desenvolvedores
- Como acessar o “repositório” de diferentes computadores?
  - Centralizado
  - Distribuído

Centralizado

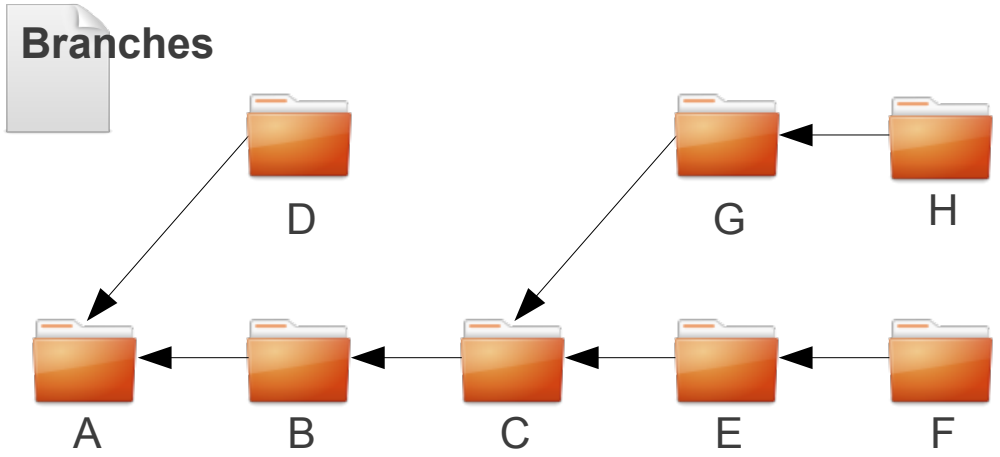
- O código inteiro se encontra em um lugar apenas (um “servidor”)
- Usuário adquire uma cópia da revisão que deseja trabalhar
- É preciso acessar o servidor para realizar qualquer operação sobre o código

Distribuído

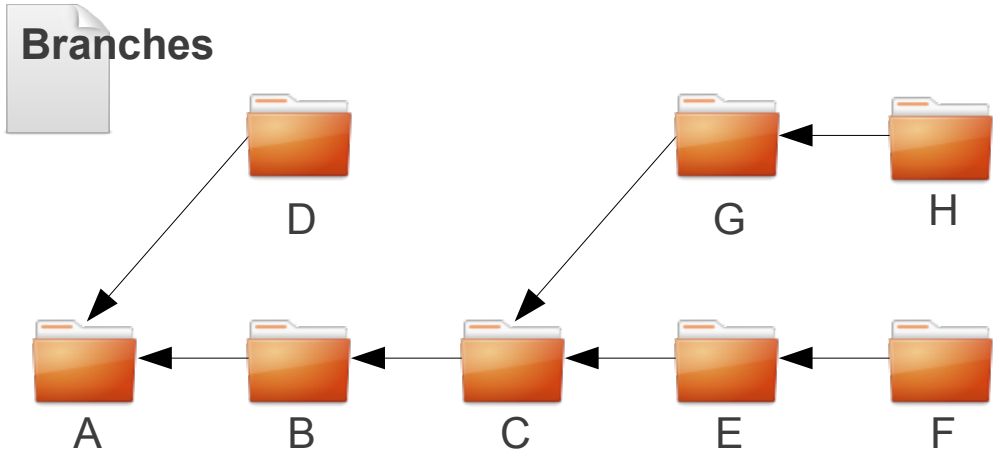
- Cada usuário recebe uma cópia de todo código e de todas versões anteriores
- Pode realizar operações de versionamento sem requisitar um servidor remoto
- Como trabalhar em equipe?

Sistema distribuído

Computador 1

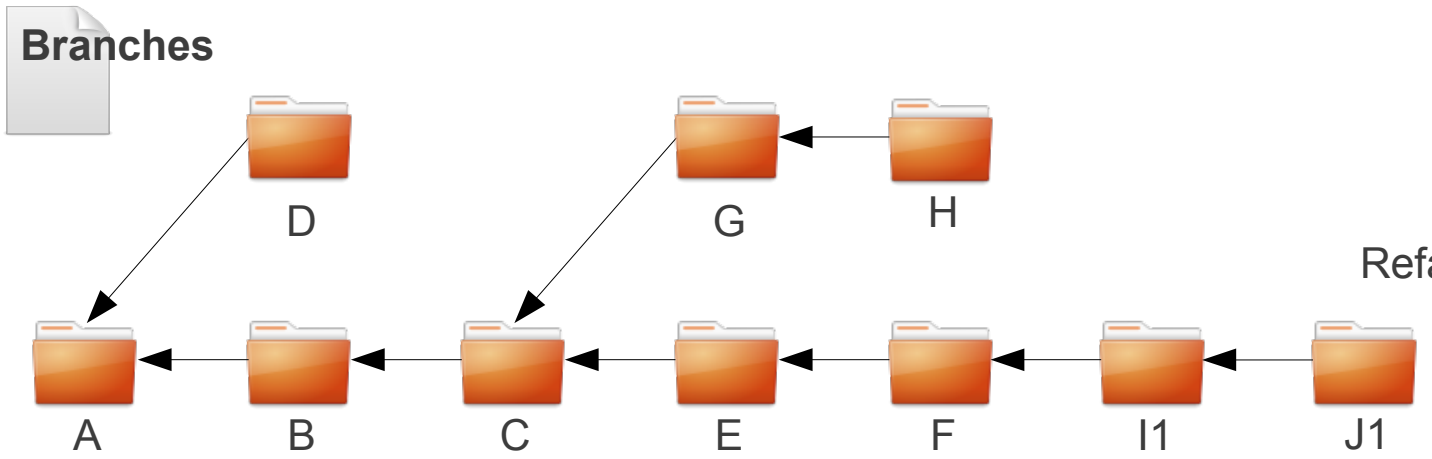


Computador 2



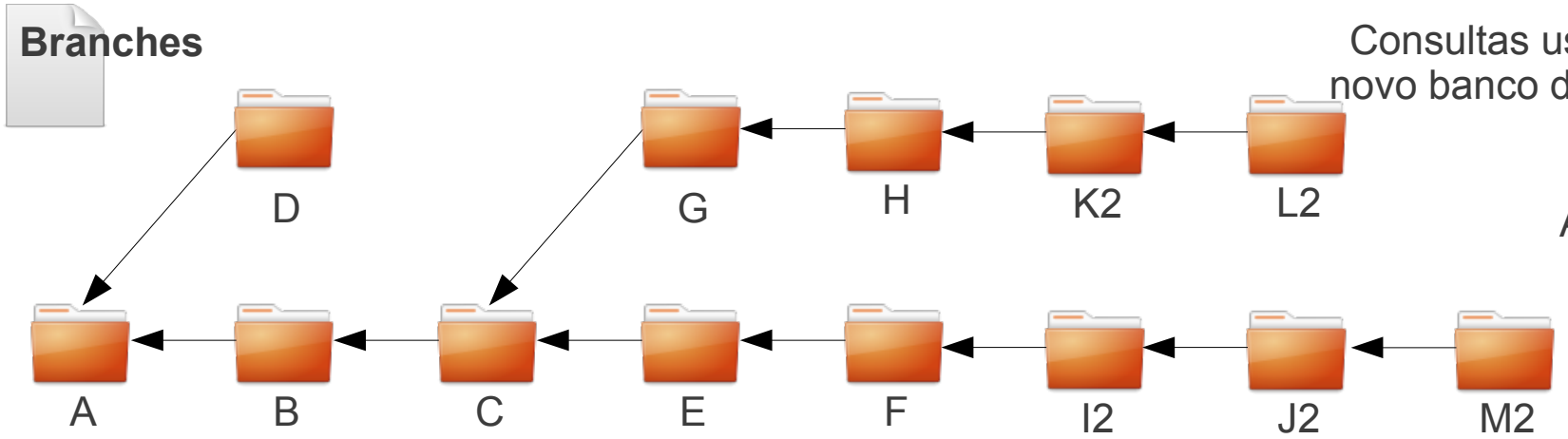
Sistema distribuído

Computador 1



Refatorando o código do raio laser mortal.

Computador 2



Consultas usando o novo banco de dados.

Aumentando o poder de destruição.

Múltiplas cópias do repositório

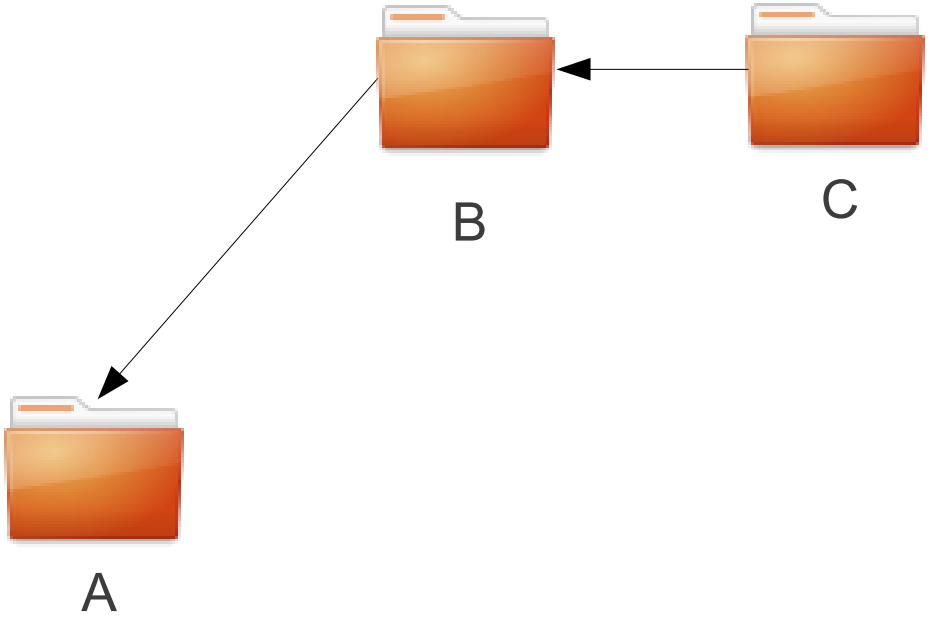
- Para que este esquema funcione precisamos de algumas coisas:
  - Informação sobre o autor de cada versão
  - Unicidade do nome de cada versão
- Podemos contar com alguns fatos
  - Existe um pai comum entre qualquer duas versões.
  - Pode-se misturar os dois repositórios em um momento futuro e criar uma “única” cópia novamente.
    - Desenvolvimento “offline”

Nome e autor da versão

- Podemos guardar o nome do autor da versão juntamente com a mensagem.
- Podemos usar o conteúdo da mensagem para criar um hash único e usar este hash como o nome.
  - Git usa SHA1

Merges

Branches
→ A (master)
C (alternativo)

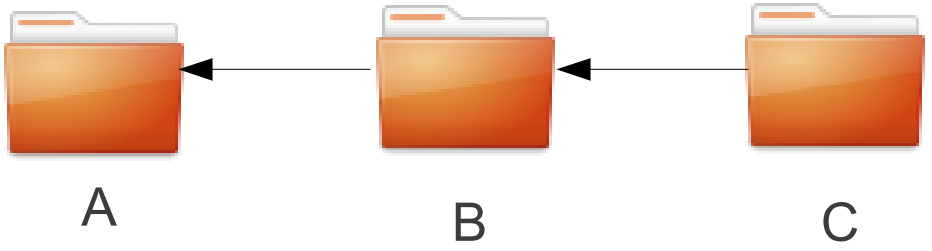


Merges – Fast Forward

Branches
→ C (master)
C (alternativo)

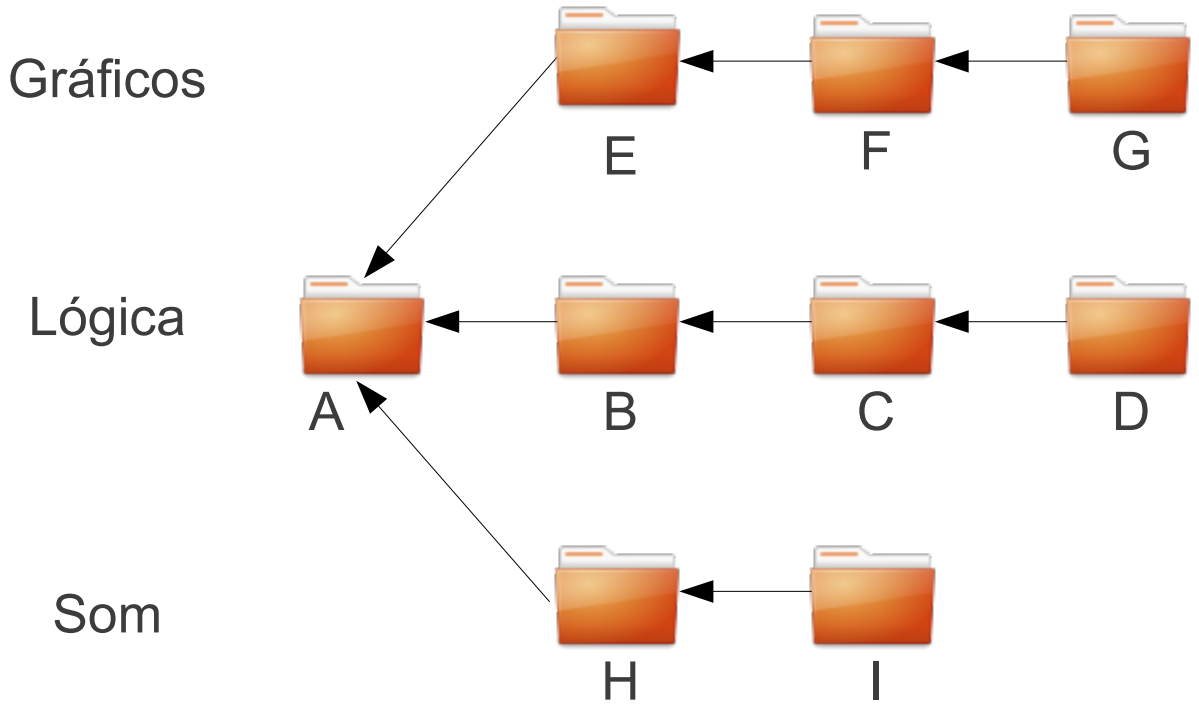
Branches

master +
alternativo (C)



Merges – 2 pais

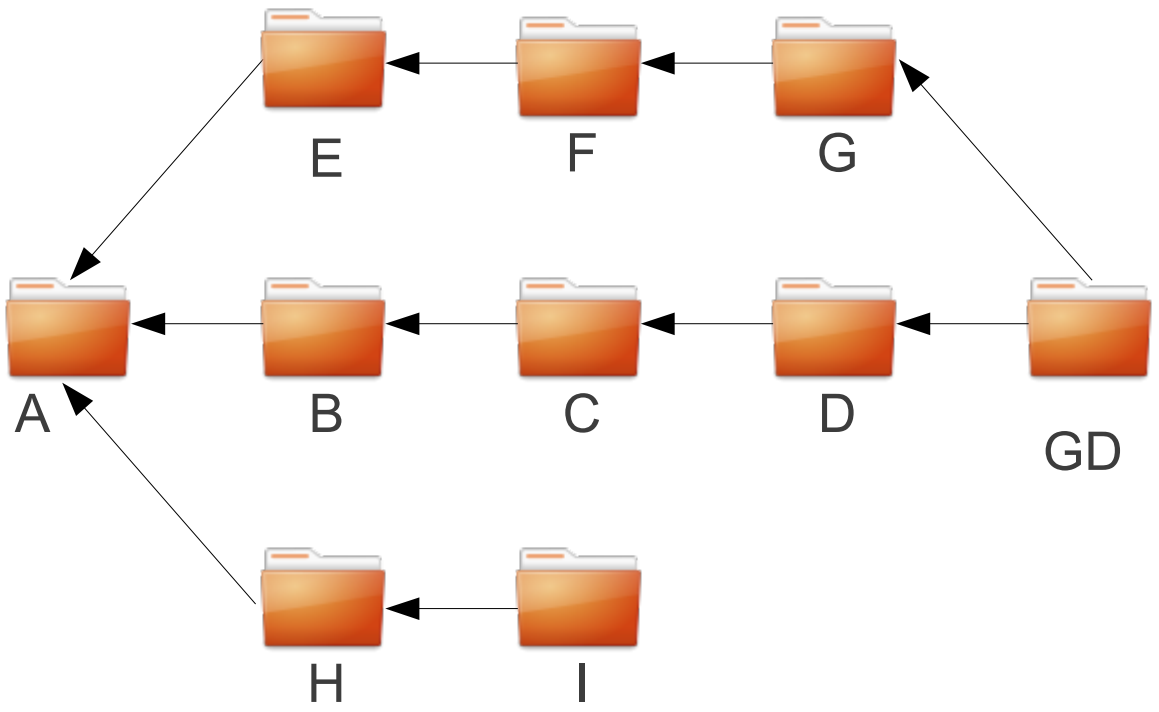
Branches



Branches
→ D (logic)
G (graphics)
I (sound)

Merges – 2 pais

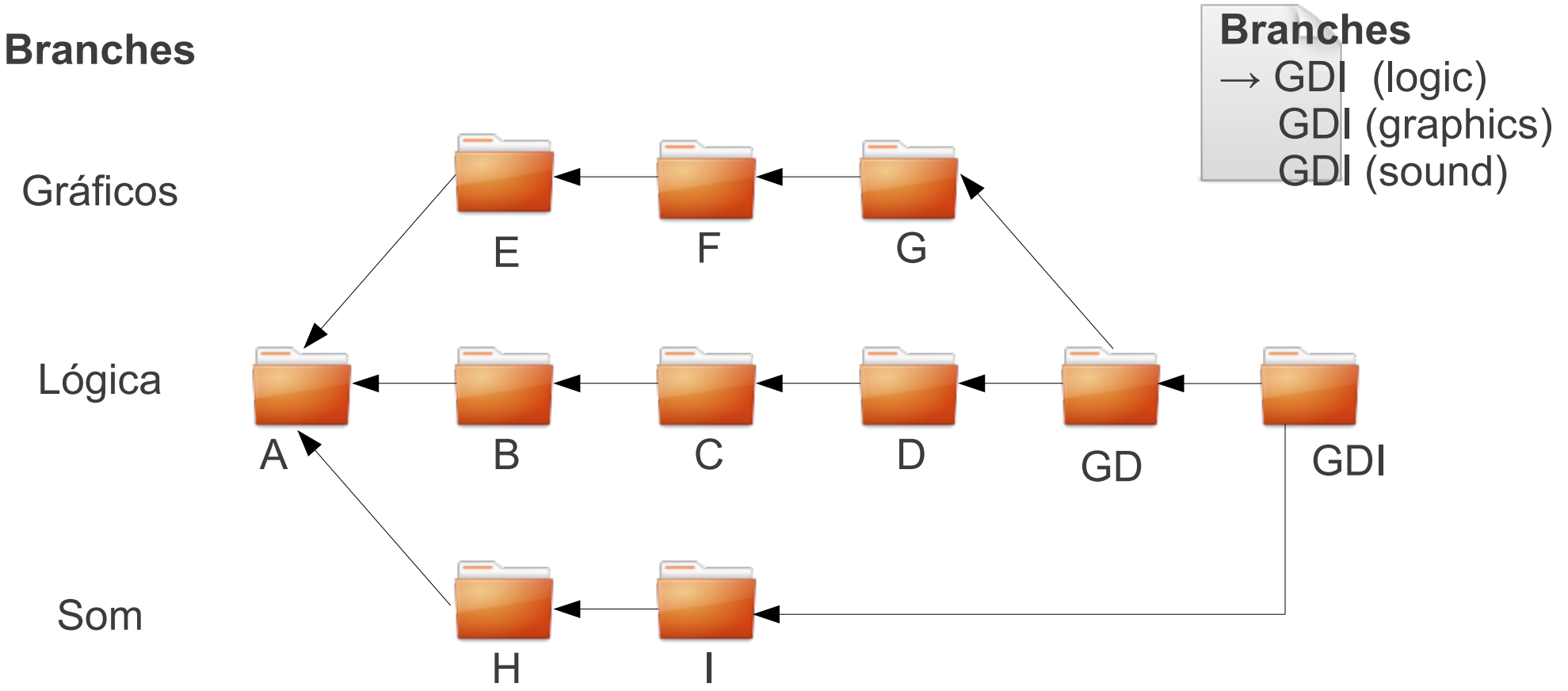
Branches



Branches
→ GD (logic)
GD (graphics)
I (sound)

Versão especial para o merge.

Merges – 2 pais



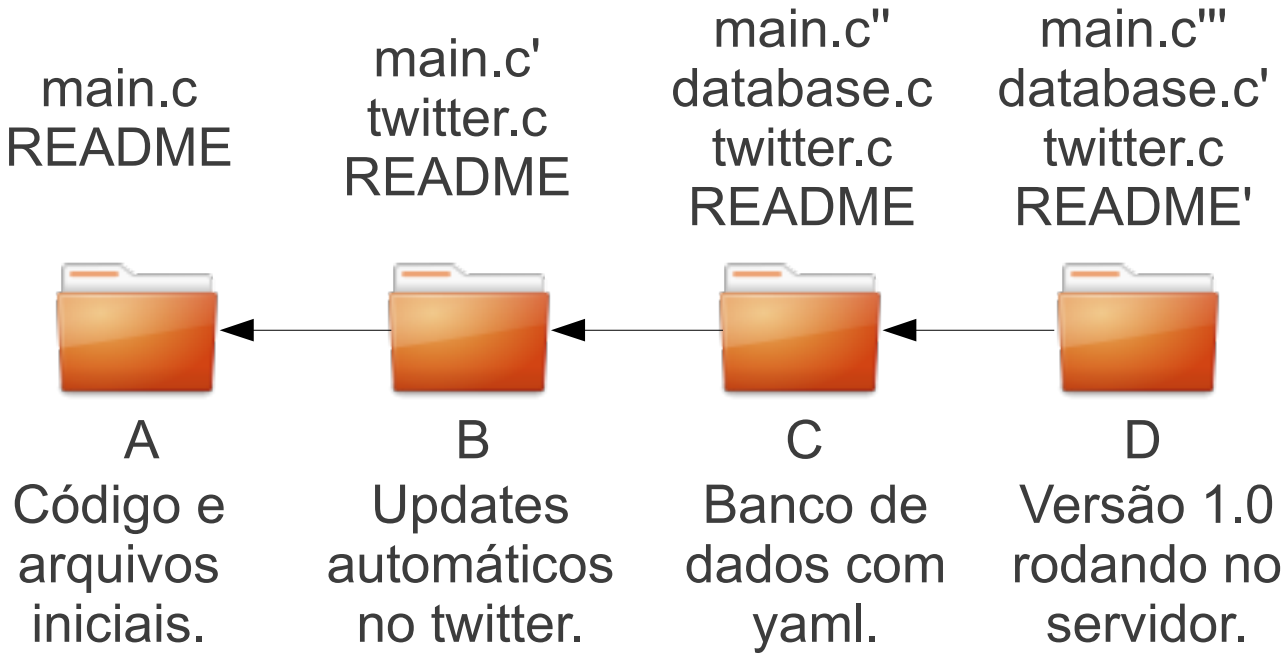
Recapitulando 2

- Versões ordenadas temporalmente de todos estágios de desenvolvimento
- Branches de desenvolvimento alternativos e uma maneira de juntá-los
- Repositório distribuído, com nomes de versão e autores únicos
- Dados em uma versão
  - Código, Nome da versão (hash),Nome da versão anterior, Mensagem, Autor da mudança

Duplicação

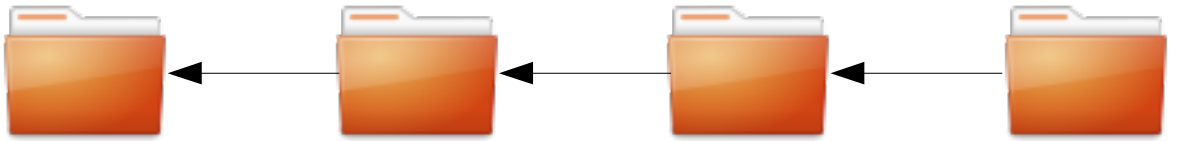
- Nosso sistema de versionamento está “pronto”, mas algumas coisas poderiam ser feitas melhor.
- Sistema de snapshots usa muita memória e guarda diversas vezes o mesmo arquivo.
- Não existe uma maneira fácil de visualizar a diferença entre dois snapshots do software.

Blobs


main.c
README
main.c'
twitter.c
README
main.c''
database.c
twitter.c
README
main.c'''
database.c'
twitter.c
README'
A
Código e arquivos iniciais.
B
Updates automáticos no twitter.
C
Banco de dados com yaml.
D
Versão 1.0 rodando no servidor.

Blobs

Repositório



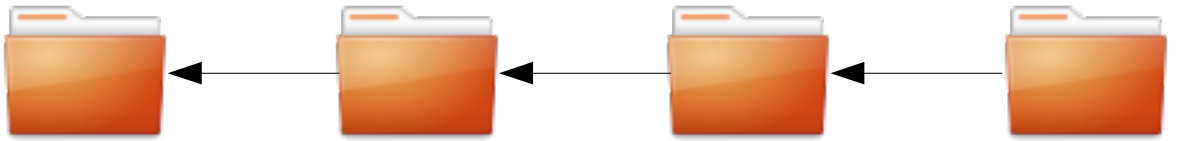
A	B	C	D
main.c README	main.c' twitter.c README	main.c'' database.c twitter.c README	main.c''' database.c' twitter.c README'
Código e arquivos iniciais.	Updates automáticos no twitter.	Banco de dados com yaml.	Versão 1.0 rodando no servidor.

Disco

main.c README twitter.c database.c database.c'
main.c', main.c'', main.c''' twitter.c README'
main.c README README twitter.c

Blobs

Repositório

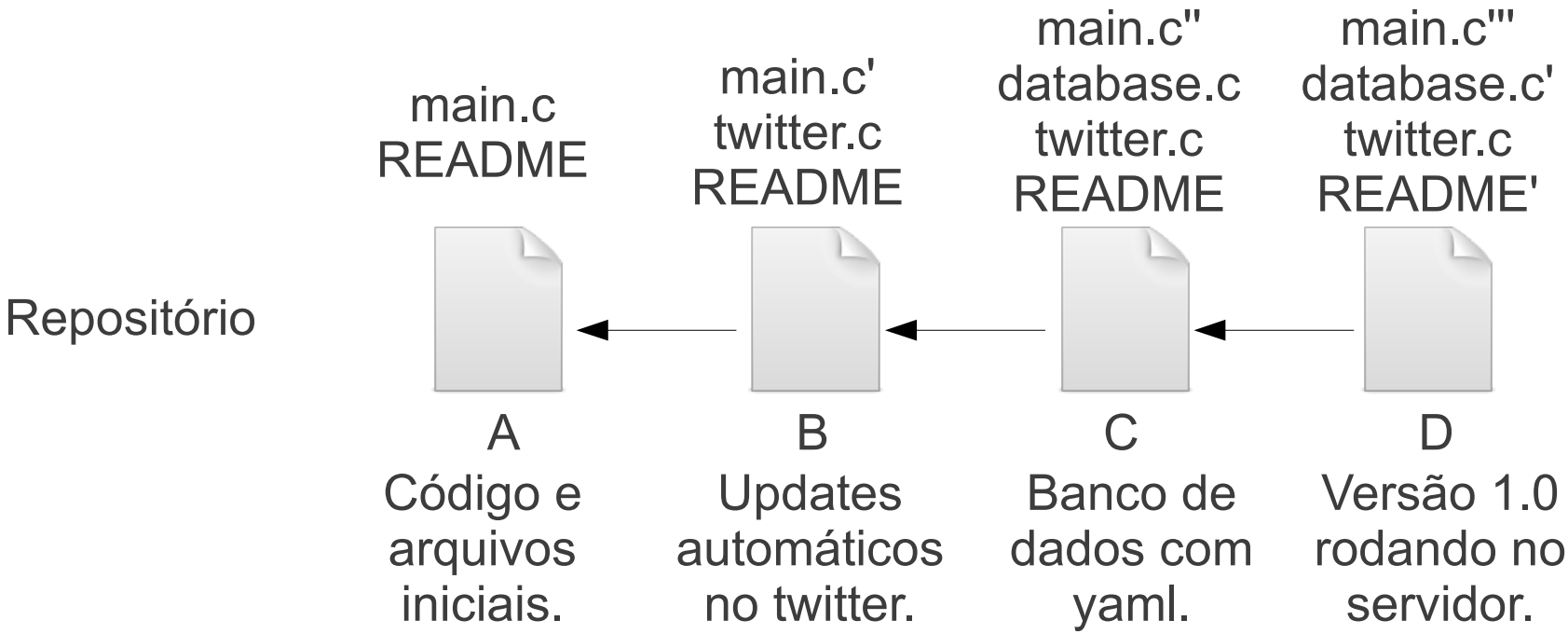


A	B	C	D
main.c README	main.c' twitter.c README	main.c'' database.c twitter.c README	main.c''' database.c' twitter.c README'
Código e arquivos iniciais.	Updates automáticos no twitter.	Banco de dados com yaml.	Versão 1.0 rodando no servidor.

Disco (sem cópias)

main.c README main.c' twitter.c main.c'' database.c main.c''' database.c' README'

Blobs



main.c README main.c' twitter.c main.c'' database.c main.c''' database.c' README'

Diff

- Como descobrir quais foram as mudanças que ocorreram entre duas versões?
  - Guardar apenas as modificações entre cada duas versões
    - Usa menos memória
    - Requer mais passos para recuperar um arquivo
  - Calcular o diff quando for requisitado
    - Usa mais memória
    - Recupera arquivos instantaneamente

Compressão

- Das duas abordagens, Git usa a segunda. Guarda cópias dos arquivos e calcula diff quando necessário. Questão de velocidade.
- Horrível quando temos arquivos enormes com pequenas modificações
- As blobs podem permanecer comprimidas e serem descomprimidas quando necessário.

Intermission

Matehackers

Git

- Acabamos de construir “bottom up” o Git com a maioria de suas features.
- Versão ou Snapshot = Commit
- Então vamos meter a mão na massa!

Começando...

- Um novo repositório

```
$ git init .
```

- De um repositório já existente

```
$ git clone git@github.com:lfzawacki/cm.git
```

Identidade

- É importante ter um nome de usuário e um email para usar como indentificação dos seus commits
- Dependendo do servidor remoto será preciso configurar algum tipo de autenticação (github usa chave pública/privada para o ssh)
- Pode ser configurado em ~/.gitconfig

Staging Area

```
$ git status
```

```
$ git add [arquivo]
```

Commits

```
$ git commit

$ git commit -m “Sua mensagem”

$ git commit -am “Sua mensagem”
```

Checkout

```
$ git checkout numero_revisao

$ git checkout HEAD

$ git checkout HEAD^

$ git checkout HEAD~3
```

Reset

```
$ git reset arquivo

$ git reset HEAD^ arquivo

$ git reset HEAD^^ arquivo

$ git reset --hard
```

Push/Pull

```
$ git push [remote] [branch]

$ git pull [remote] [branch]

$ git push origin master

$ git pull origin master
```

Branches

```
$ git branch

$ git branch novo_branch

$ git checkout novo_branch

$ git branch -d novo_branch
```

Merge

```
$ git checkout branch1

… trabalha duro …

$ git checkout branch2

$ git merge branch1
```

Help

```
$ git help commit

$ git help checkout

$ git help log

$ git help status
```

Log

```
$ git log

$ git log -p -2

$ git log --stat

$ git log --shortstat
```

Amend

```
$ git commit -m “Minhas mudancas”

$ git commit --amend -m “Outra mensagem”

… modifica um arquivo ...

$ git add arquivo
$ git commit --amend -m “Explica mudança”
```

Diff

```
$ git diff

$ git diff HEAD^

$ git diff HEAD^ HEAD~20

$ git diff HEAD^ arquivo
```

Patches

```
$ git format-patch HEAD^

$ git format-patch HEAD^^

$ git format-patch -k HEAD^
```

Stash

```
$ git stash

$ git stash pop

$ git stash list

$ git stash show
```

Configurações

- .gitconfig
- .gitignore

Interfaces gráficas

```
$ gitk
```

```
$ git instaweb
```

- Outras ...

Frontends

- Linha de comando
- msysgit: http://code.google.com/p/msysgit/
- TortoiseGit: http://code.google.com/p/tortoisegit/

Hospedagem

- http://repo.or.cz Hospdagem gratuita
- http://baregit.com Outra gratuira
- http://github.com Rede social + hospedagem (gratuito para open source)
- http://bitbucket.com Rede social + hospedagem gratuita
- Self hosted

Outros sistemas

- Centralizado
  - Subversion
  - CVS
- Distribuído
  - Mercurial
  - Bazaar
  - Fossil

Referências

- Git Magic: http://www-cs-students.stanford.edu/~blynn/gitmagic/index
- Pro Git: http://progit.org/book/
- Git Cheatsheet: http://byte.kde.org/~zrusin/git/git-cheat-sheet-medium.png
- Dicas por mim:

  http://lfzawacki.heroku.com/life/published/git

Obrigado!

Perguntas?

http://blog.lfzawacki.com

@lfzawacki