Controle de versão e fluxo de trabalho em projetos de desenvolvimento de software

Antonio Terceiro

Novembro/2012

Termos de uso

Termos de uso – condições

Desde que:

- Atribuição. Você deve dar crédito ao autor original, da forma especificada pelo Autor ou licenciante.
- Uso Não-Comercial. Você não pode utilizar esta obra com finalidades comerciais.
- Compartilhamento pela mesma Licença. Se você alterar, transformar, ou criar Outra obra com base nesta, você somente poderá distribuir a obra resultante sob Uma licença idêntica a esta.

Para cada novo uso ou distribuição, você deve deixar claro para outros os termos da licença desta obra.
Qualquer uma destas condições podem ser renunciadas, desde que Você obtenha permissão do autor.

Resumo

Será apresentado um breve histórico dos sistemas de controle de versão livres, tanto do ponto de vista tecnológico quanto do modelo de fluxo de trabalho pressuposto pelos mesmos. Será dada ênfase em sistemas de controle de versão distribuído, em especial `git` (`http://git-scm.com/`), e como eles podem suportar diferentes *workflows* em desenvolvimento de software.

Objetivos deste curso

- Motivar para o uso de sistemas de controle de versão.
- Aprender conceitos básicos sobre a utilização de sistemas de controle de versão em geral.
- Introduzir o uso de `git`, um sistema de controle de versão distribuído.
- Discutir possíveis *workflows* em projetos de software, e como um sistema de controle de versão como `git` pode suportá-los.

Roteiro

Roteiro

Motivação

Motivação

- Todo software útil passará por mudanças.

Motivação

- Todo software útil passará por mudanças.
- As mudanças pelas quais o software passa precisam ser documentadas.

Motivação

- Todo software útil passará por mudanças.
- As mudanças pelas quais o software passa precisam ser documentadas.
- As várias mudanças específicas necessárias durante o ciclo de vida de um software devem poder ser:

Motivação

- Todo software útil passará por mudanças.
- As mudanças pelas quais o software passa precisam ser documentadas.
- As várias mudanças específicas necessárias durante o ciclo de vida de um software devem poder ser:
  - separadas uma da outra

Motivação

- Todo software útil passará por mudanças.
- As mudanças pelas quais o software passa precisam ser documentadas.
- As várias mudanças específicas necessárias durante o ciclo de vida de um software devem poder ser:
  - separadas uma da outra
  - delimitadas no tempo e ordenadas

Motivação

- Todo software útil passará por mudanças.
- As mudanças pelas quais o software passa precisam ser documentadas.
- As várias mudanças específicas necessárias durante o ciclo de vida de um software devem poder ser:
  - separadas uma da outra
  - delimitadas no tempo e ordenadas
  - distribuídas por uma equipe

Sistemas de controle de versão

São sistemas que permitem armazenar conteúdo de forma que é possível:

Sistemas de controle de versão

São sistemas que permitem armazenar conteúdo de forma que é possível:

- Inspecionar o histórico de alterações a esse conteúdo;

Sistemas de controle de versão

São sistemas que permitem armazenar conteúdo de forma que é possível:

- Inspecionar o histórico de alterações a esse conteúdo;
- Verificar o teor de uma alteração específica que pode ter acontecido num momento arbitrário do passado;

Sistemas de controle de versão

São sistemas que permitem armazenar conteúdo de forma que é possível:

- Inspecionar o histórico de alterações a esse conteúdo;
- Verificar o teor de uma alteração específica que pode ter acontecido num momento arbitrário do passado;
- Gerenciar versões diferentes de conteúdo, e combiná-las de forma a gerar uma outra versão.

Sistemas de controle de versão

São sistemas que permitem armazenar conteúdo de forma que é possível:

- Inspecionar o histórico de alterações a esse conteúdo;
- Verificar o teor de uma alteração específica que pode ter acontecido num momento arbitrário do passado;
- Gerenciar versões diferentes de conteúdo, e combiná-las de forma a gerar uma outra versão.

No caso de desenvolvimento de software, esse conteúdo é o código-fonte.

Por quê você deve usar controle de versão

Por quê você deve usar controle de versão

- É possível determinar qual mudança introduziu um bug.

Por quê você deve usar controle de versão

- É possível determinar qual mudança introduziu um bug.
- Você consegue ter acesso fácil a diferentes versões do software (produção, desenvolvimento etc)

Por quê você deve usar controle de versão

- É possível determinar qual mudança introduziu um bug.
- Você consegue ter acesso fácil a diferentes versões do software (produção, desenvolvimento etc)
- É impossível desenvolver com equipes distribuídas sem usar controle de versão.

Por quê você deve usar controle de versão

- É possível determinar qual mudança introduziu um bug.
- Você consegue ter acesso fácil a diferentes versões do software (produção, desenvolvimento etc)
- É impossível desenvolver com equipes distribuídas sem usar controle de versão.
- É possível identificar as mudanças exatas que foram necessárias para introduzir uma nova funcionalidade.

Roteiro

Uma visão histórica dos sistemas de controle de versão

Uma visão histórica dos sistemas de controle de versão

- Objetivo: mostrar o histórico das ferramentas de controle de versão do ponto de vista do usuário, sem ressaltar diferenças nas implementações das ferramentas.

Uma visão histórica dos sistemas de controle de versão

- Objetivo: mostrar o histórico das ferramentas de controle de versão do ponto de vista do usuário, sem ressaltar diferenças nas implementações das ferramentas.
- Reflete a minha experiência pessoal.

Uma visão histórica dos sistemas de controle de versão

- Objetivo: mostrar o histórico das ferramentas de controle de versão do ponto de vista do usuário, sem ressaltar diferenças nas implementações das ferramentas.
- Reflete a minha experiência pessoal.
- Provavelmente reflete a experiência pessoal de muitas outras pessoas que lidam com software livre há algum tempo.

cp -r

- Técnica mais primitiva – e até intuitiva – de controle de versão
- Bastante usada por quem tem costume de trocar documentos por e-mail.
  - `projeto-v1.odt`, `projeto-v2.odt`, `projeto-v3.odt` ...
- em projetos de software, vai-se guardando cópias de diretórios inteiros para "manter o histórico".

cp -r: Vantagens e desvantagens

- Vantagens
  - Não depende de nenhuma ferramenta
  - Simples de fazer
- Desvantagens
  - Não mantém o histórico de alterações individuais
  - Difícil de analisar
  - Difícil de trabalhar em equipe

rcs

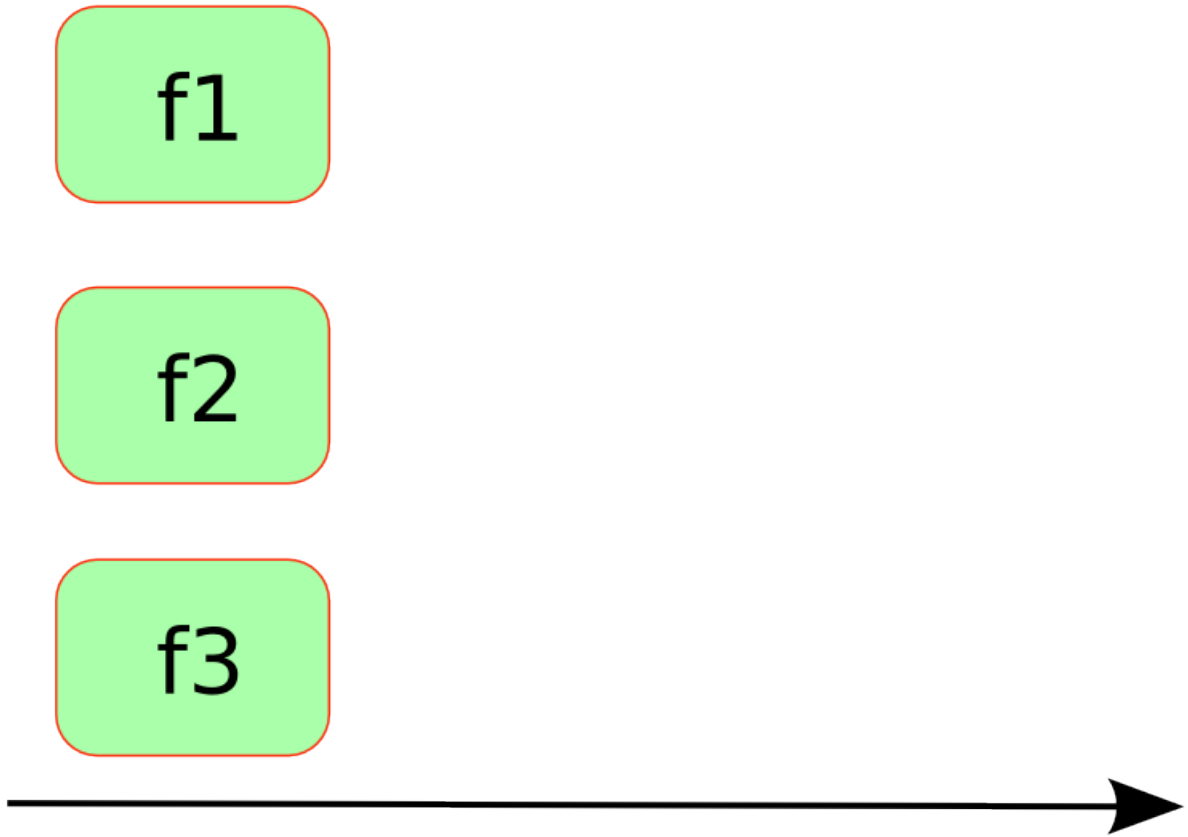
The Revision Control System (RCS) manages multiple revisions of files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, including source code, programs, documentation, graphics, papers, and form letters.

- `http://www.gnu.org/software/rcs/`

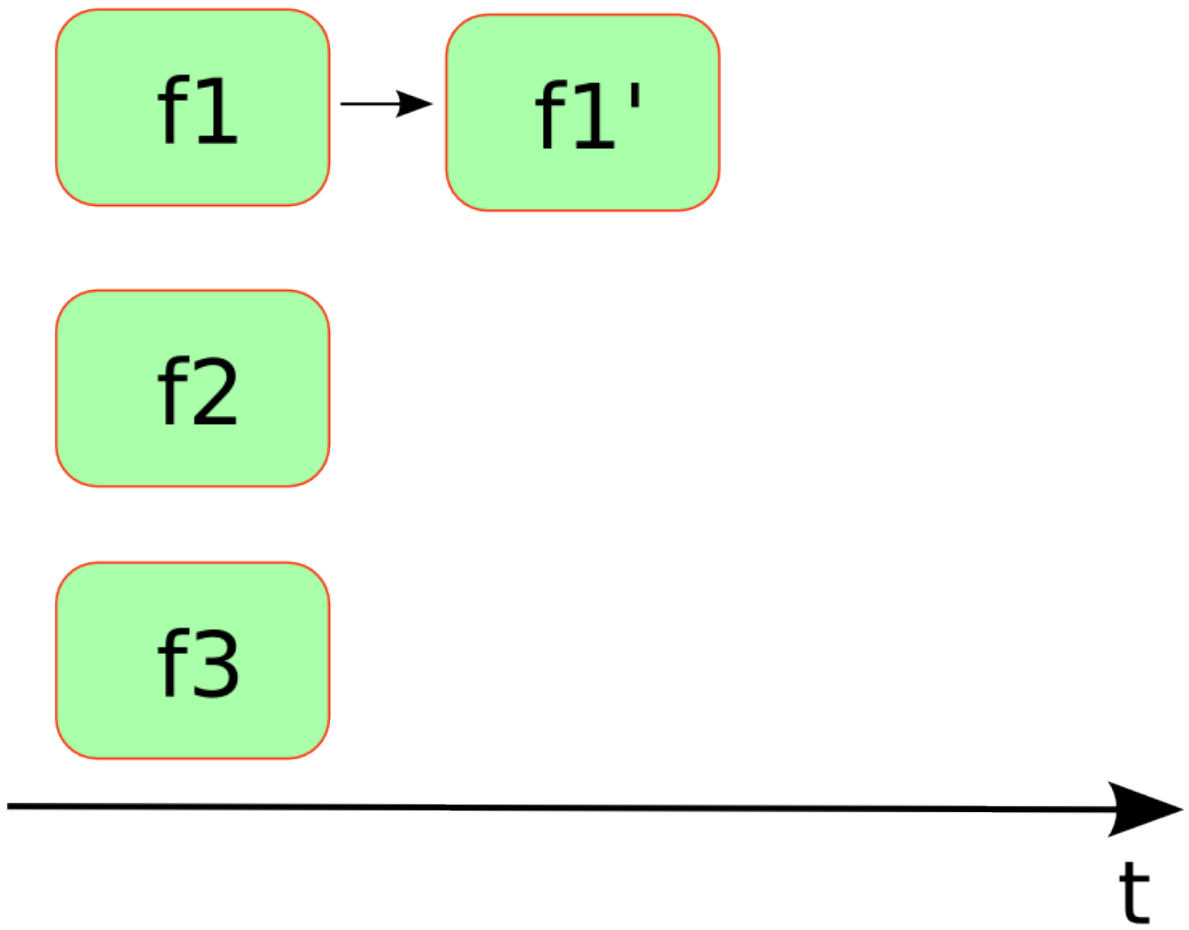
rcs, começando a usar ferramentas para controle de versão

- Melhorias
  - Já é possível trabalhar em equipe
- Dificuldades restantes
  - Desenvolvedores precisam ter acesso shell à mesma máquina.
  - Ainda não é possível trabalhar em paralelo de verdade.
  - Controle de versão é individualizado por arquivo.

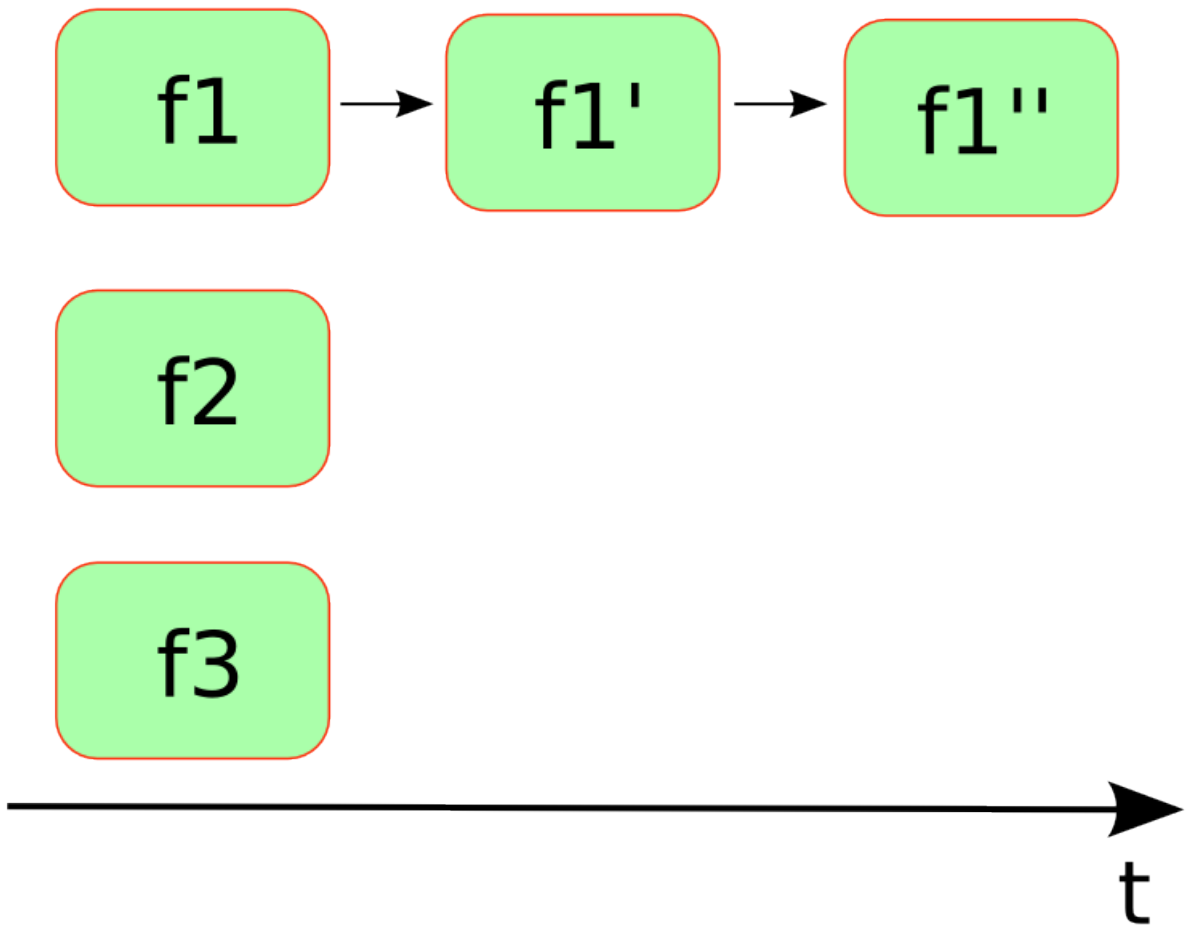
rcs: exemplo



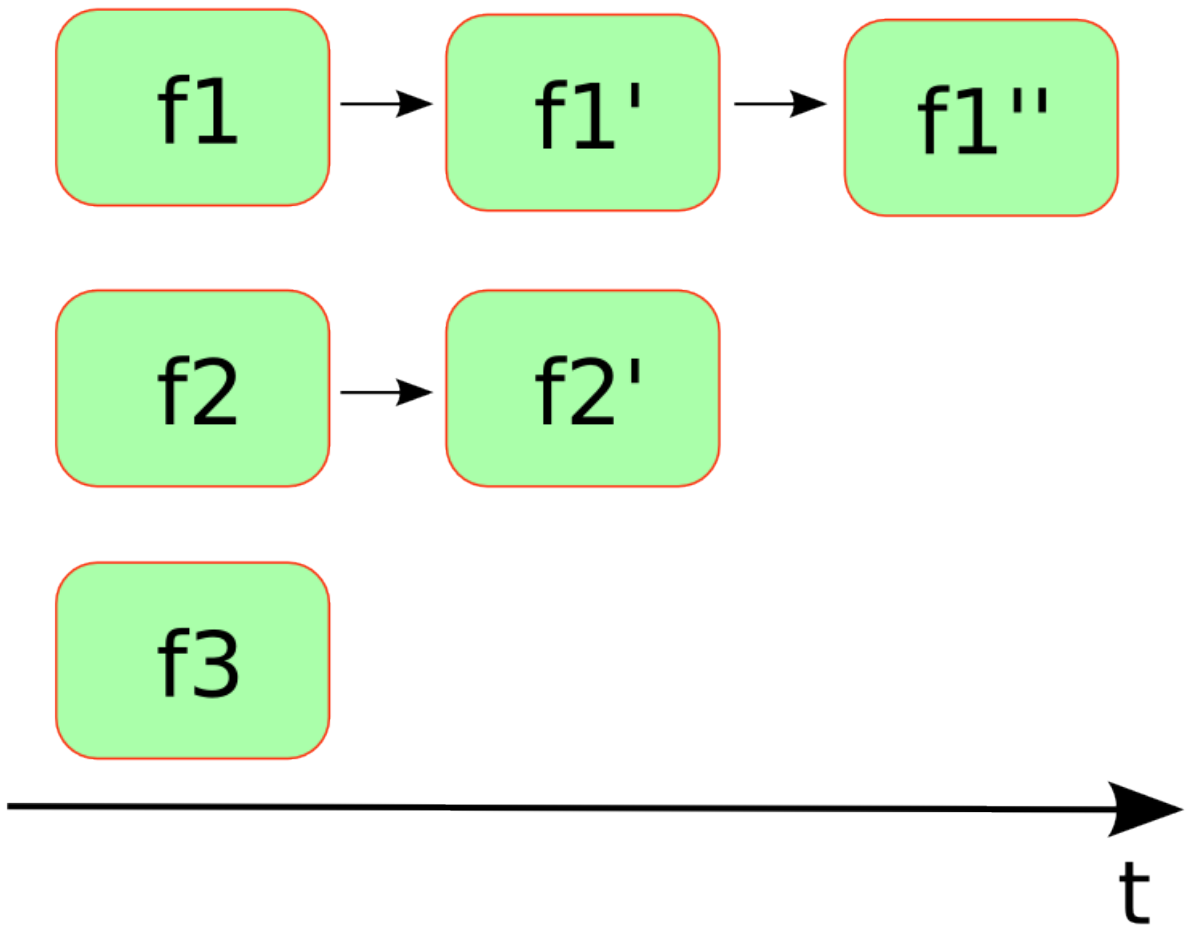
rcs: exemplo



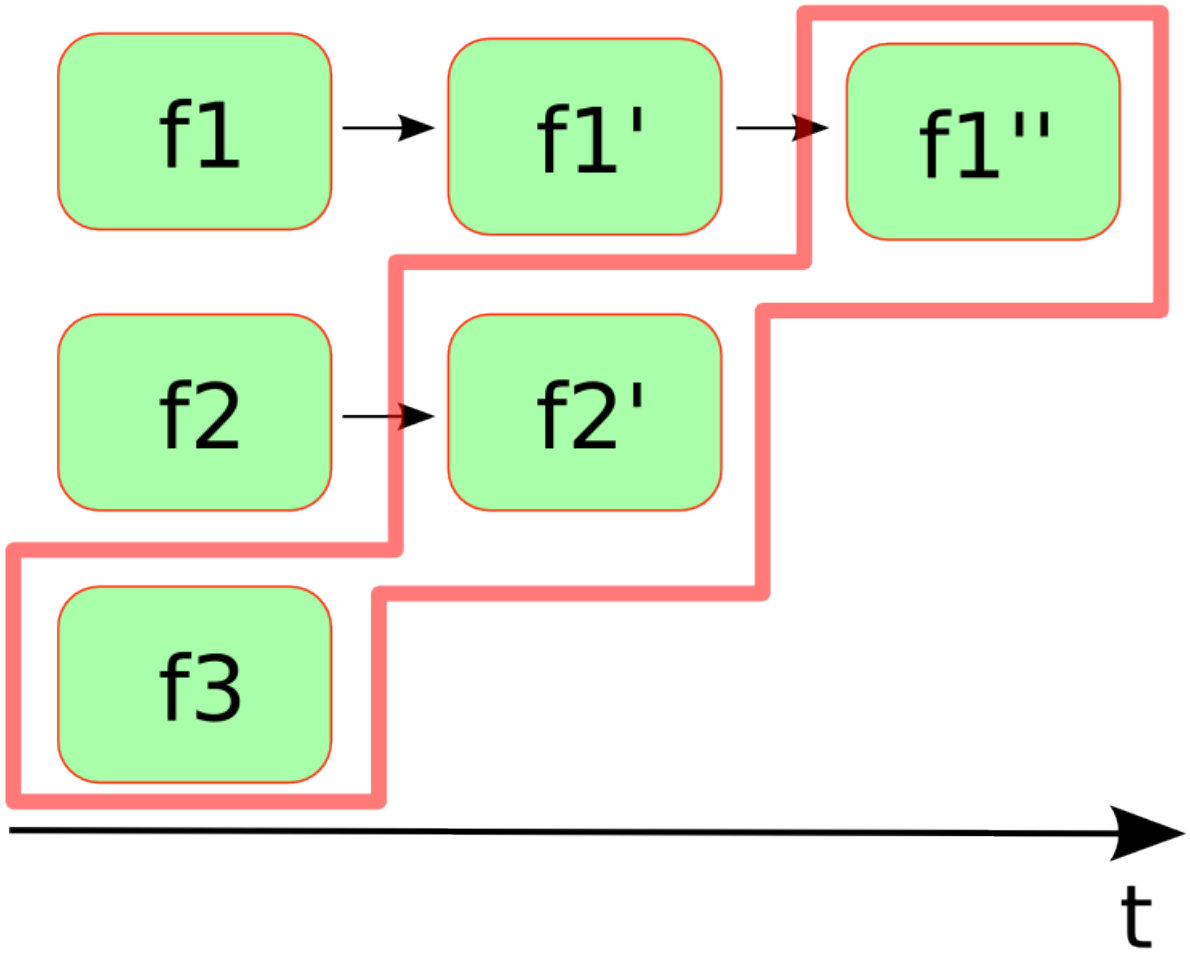
rcs: exemplo



rcs: exemplo



rcs: exemplo



CVS

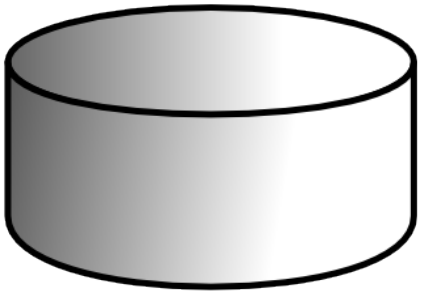
CVS is a version control system, an important component of Source Configuration Management (SCM). Using it, you can record the history of sources files, and documents. It fills a similar role to the free software RCS, PRCS, and Aegis packages.

- Concurrent Versions System
- `http://www.nongnu.org/cvs/`

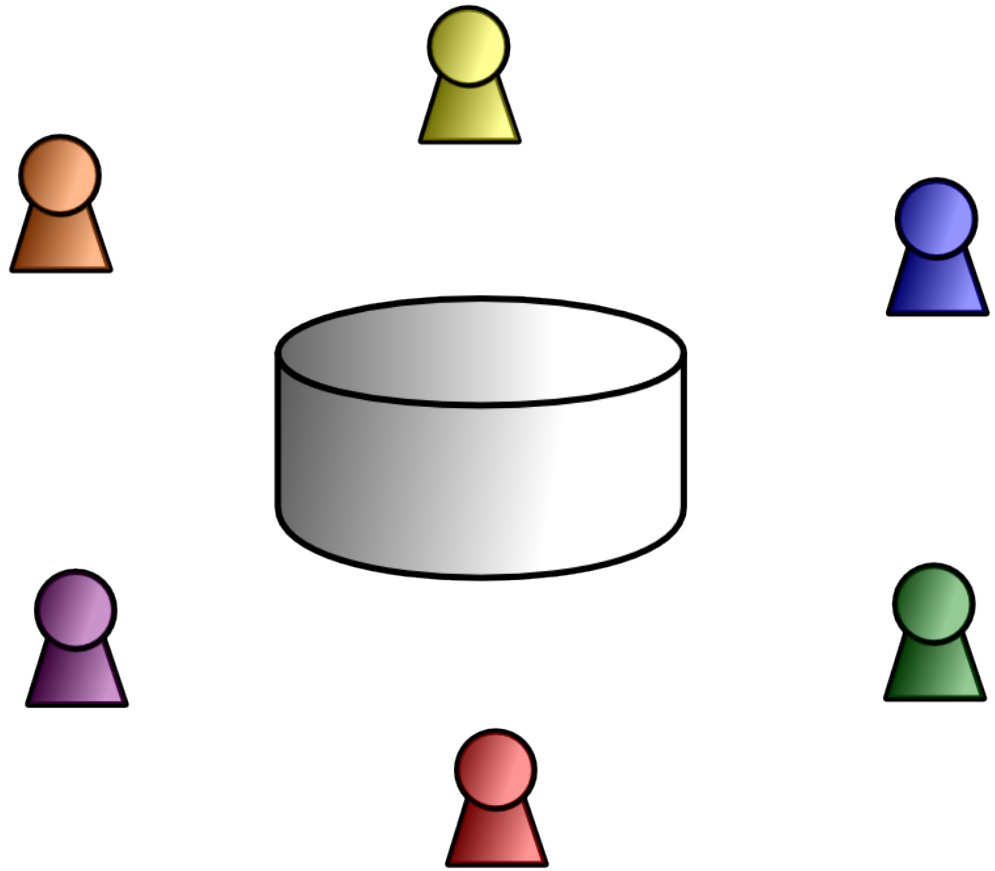
CVS, indo onde nenhum homem jamais esteve

- Avanços
  - Possível trabalhar em equipe em computadores diferentes.
  - Possível para duas pessoas trabalharem no mesmo arquivo.
  - Pode-se fazer referência ao estado de determinados arquivos num momento do tempo (tags)
- Questões remanescentes
  - Toda operação que envolve o histórico necessita que se esteja *on-line*.
  - Depende um repositório central.
  - Controle de versão ainda é individualizado por arquivo.

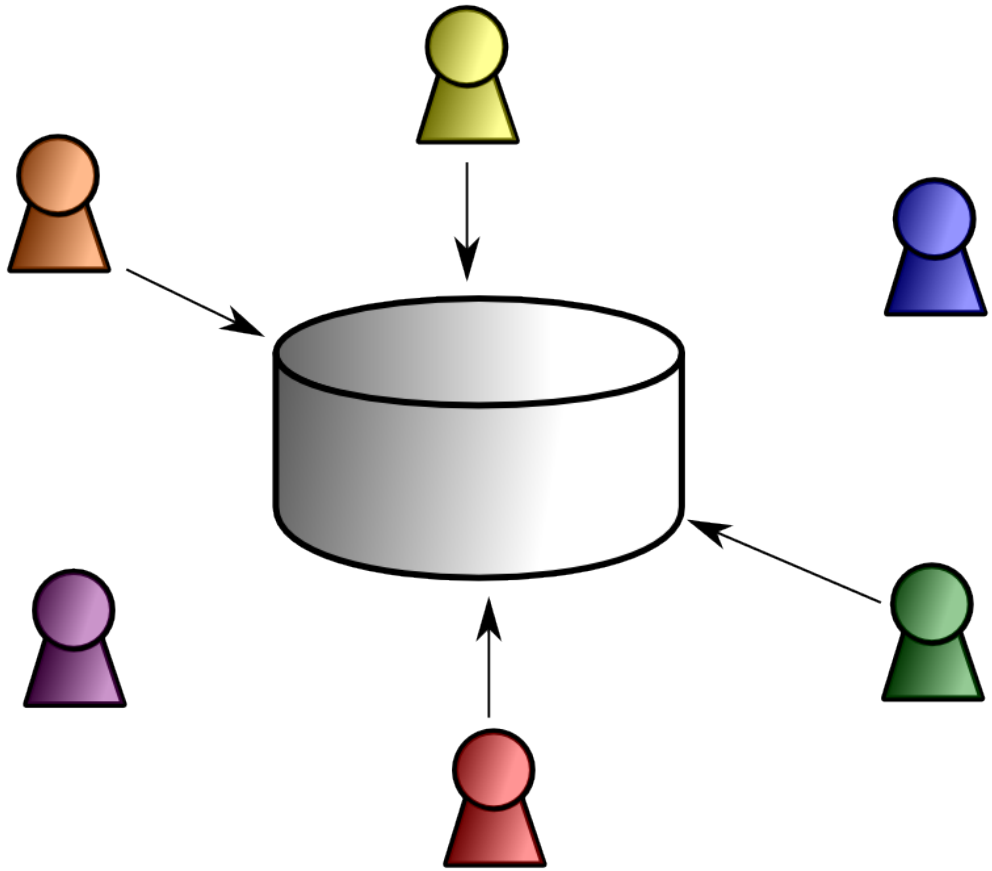
CVS: desenvolvimento distribuído



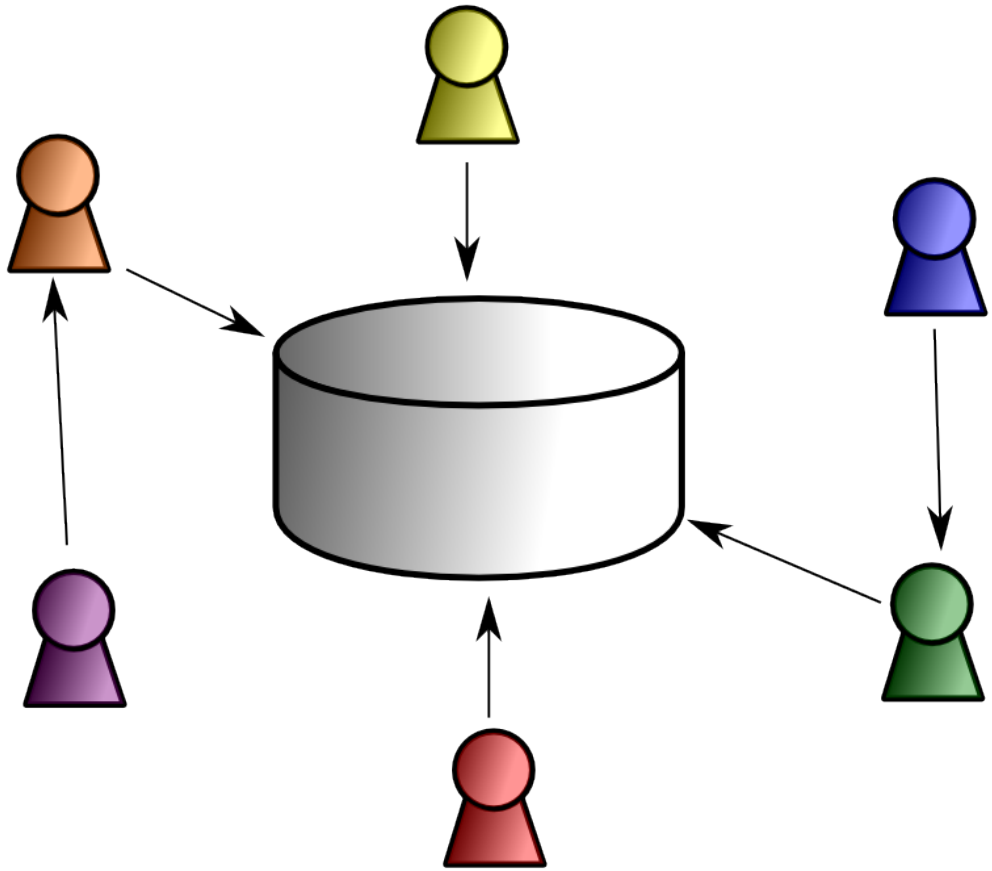
CVS: desenvolvimento distribuído



CVS: desenvolvimento distribuído



CVS: desenvolvimento distribuído

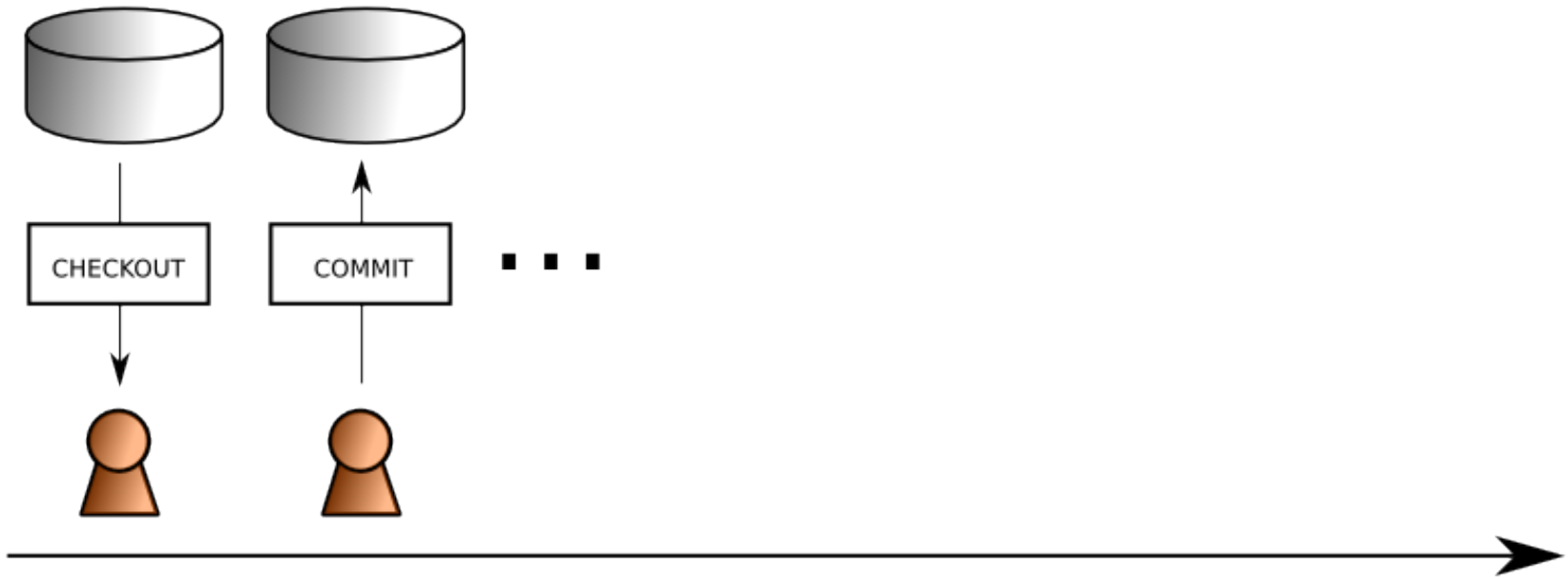


Interação de um desenvolvedor com repositório centralizado

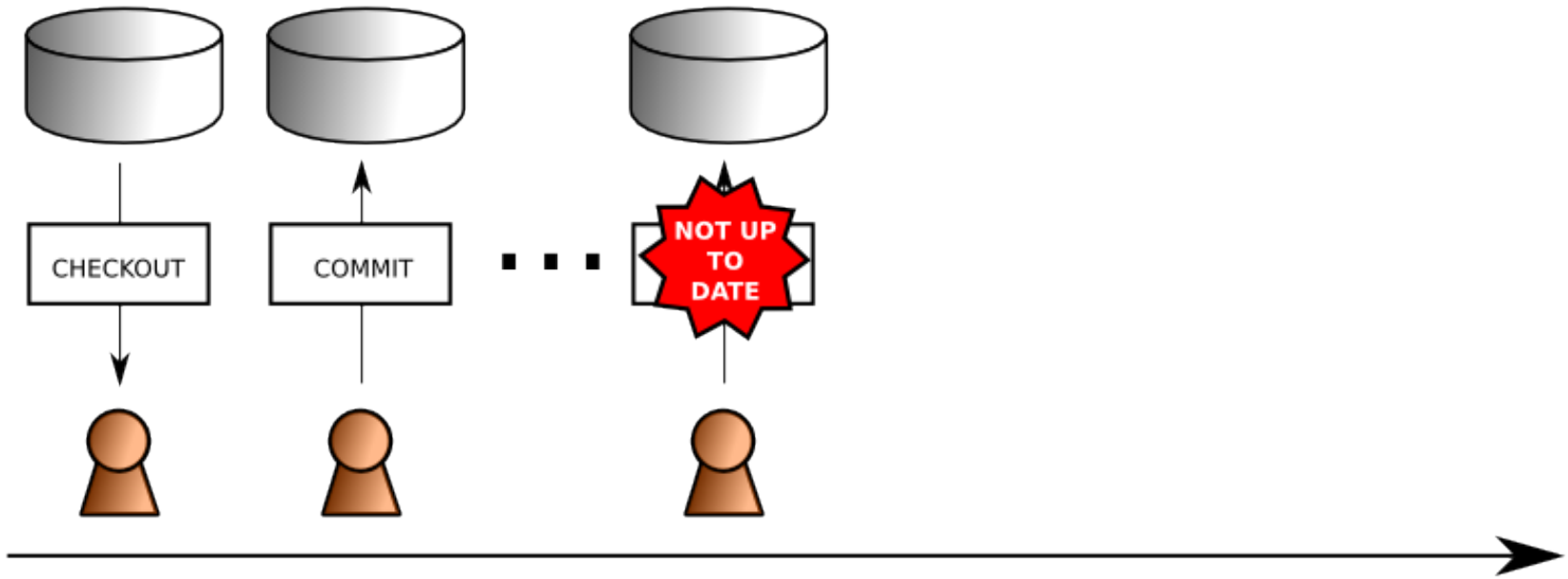
Interação de um desenvolvedor com repositório centralizado

Interação de um desenvolvedor com repositório centralizado

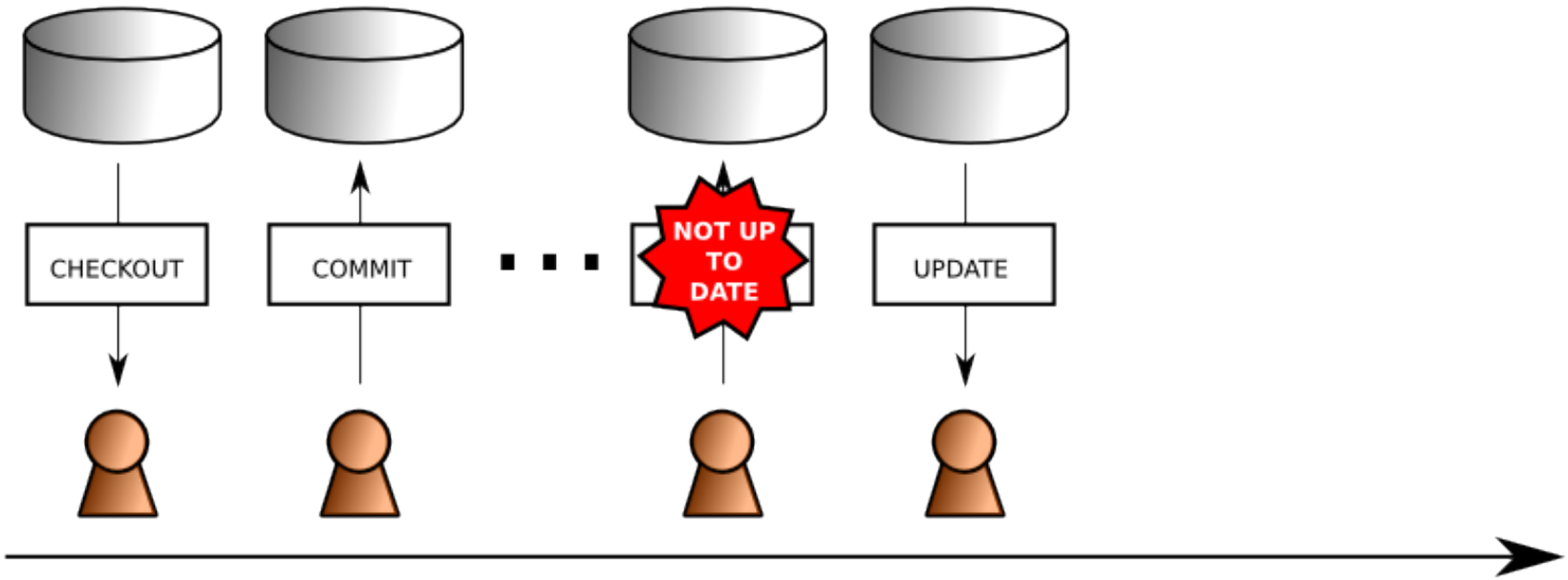
Interação de um desenvolvedor com repositório centralizado



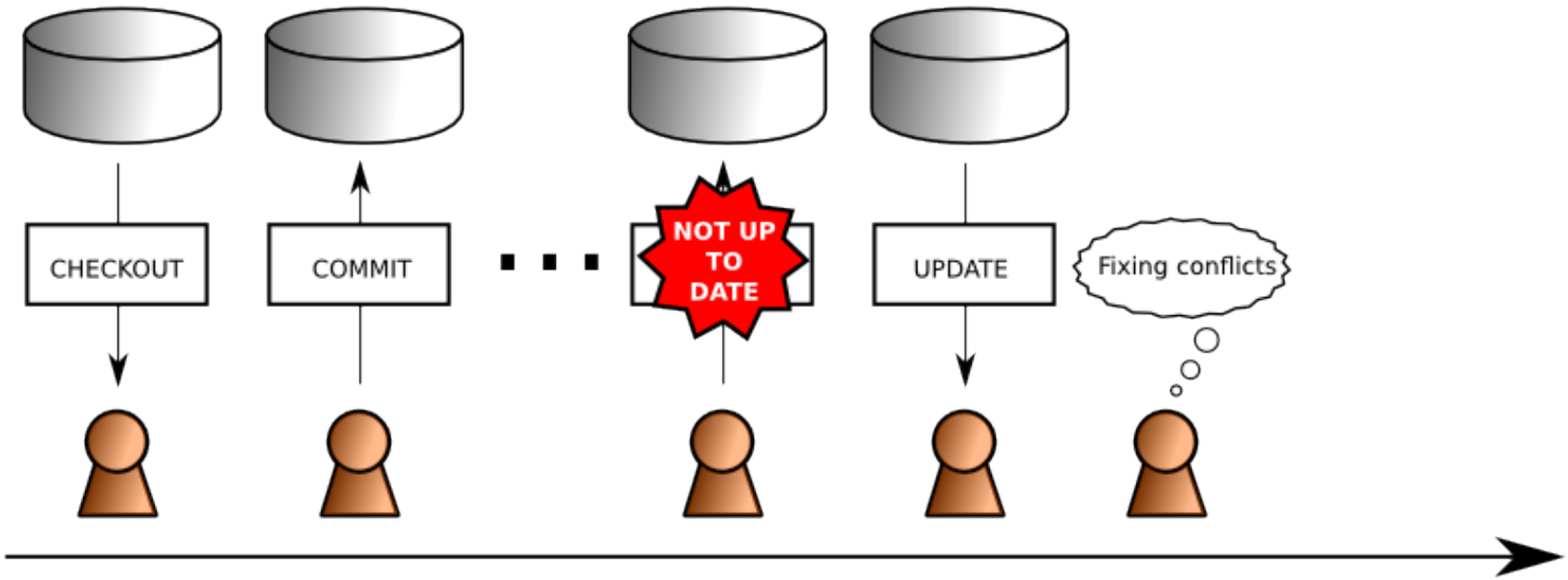
Interação de um desenvolvedor com repositório centralizado



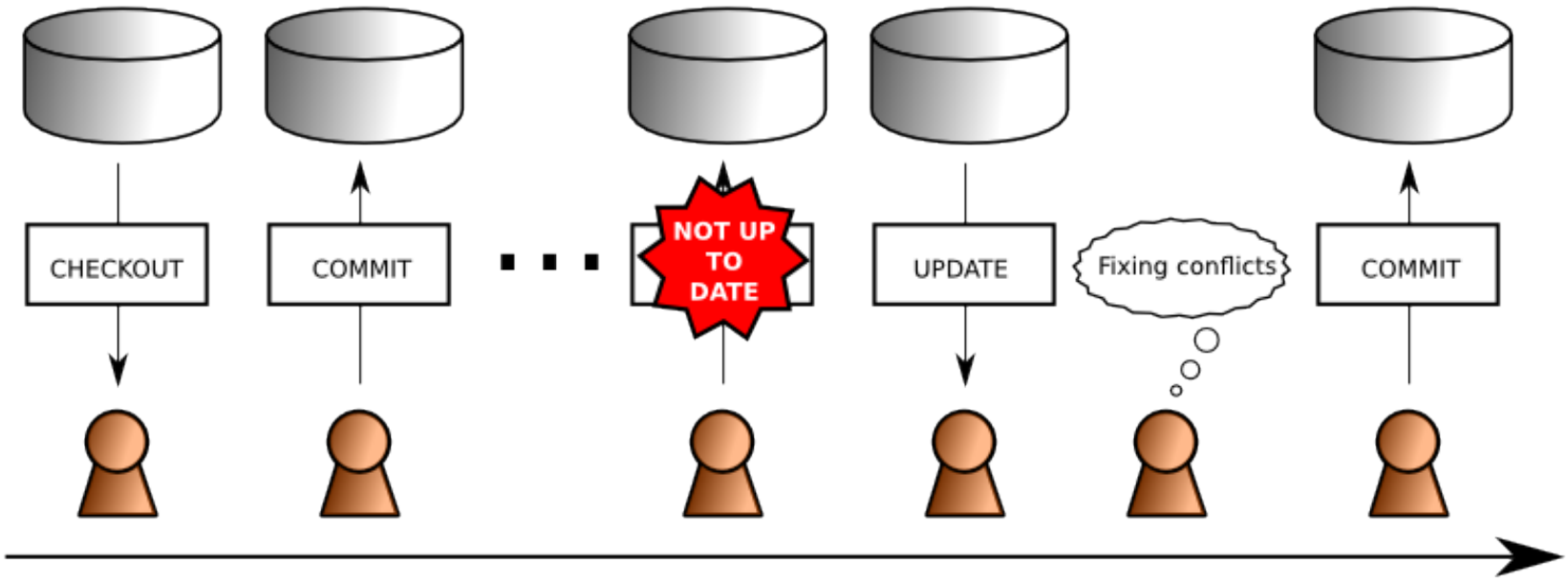
Interação de um desenvolvedor com repositório centralizado



Interação de um desenvolvedor com repositório centralizado



Interação de um desenvolvedor com repositório centralizado



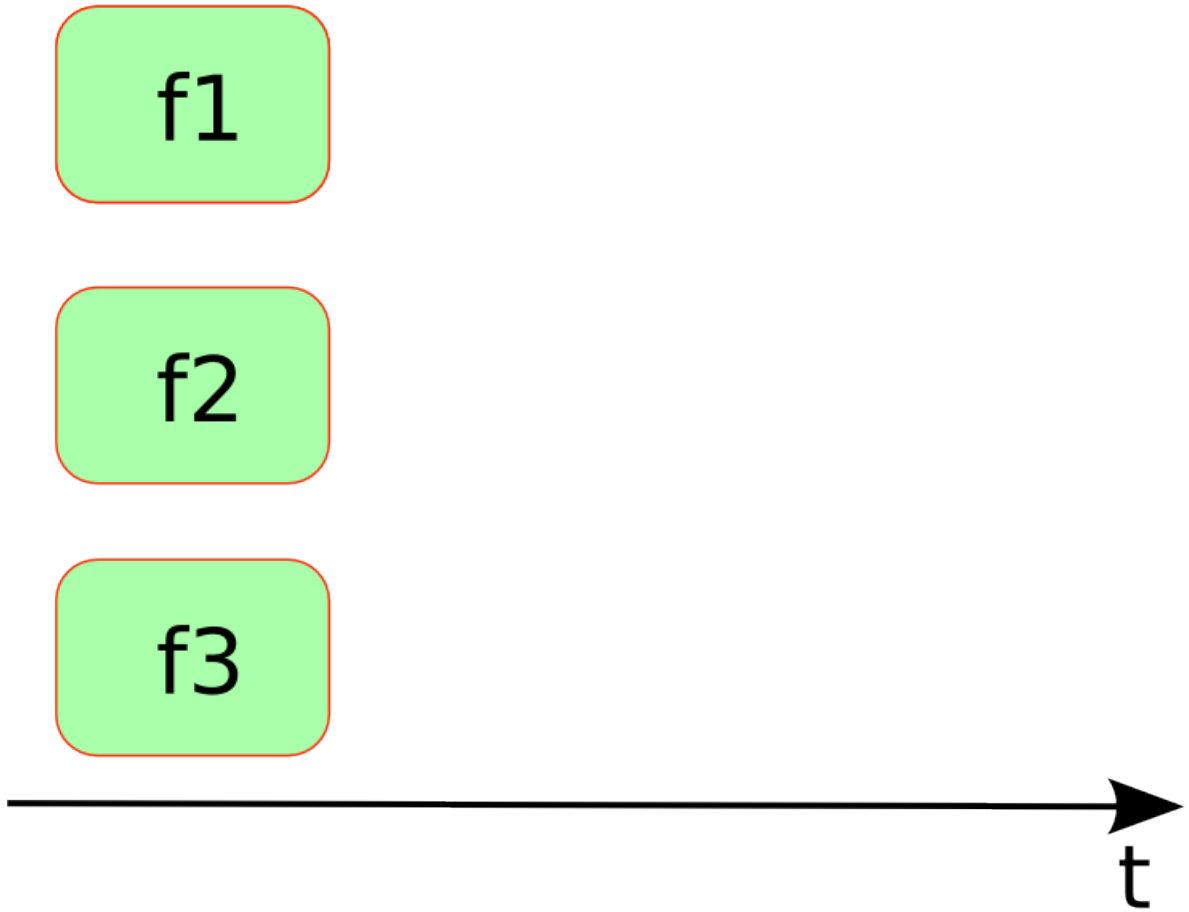
svn

- *"CVS done right"*
- `http://subversion.tigris.org/`

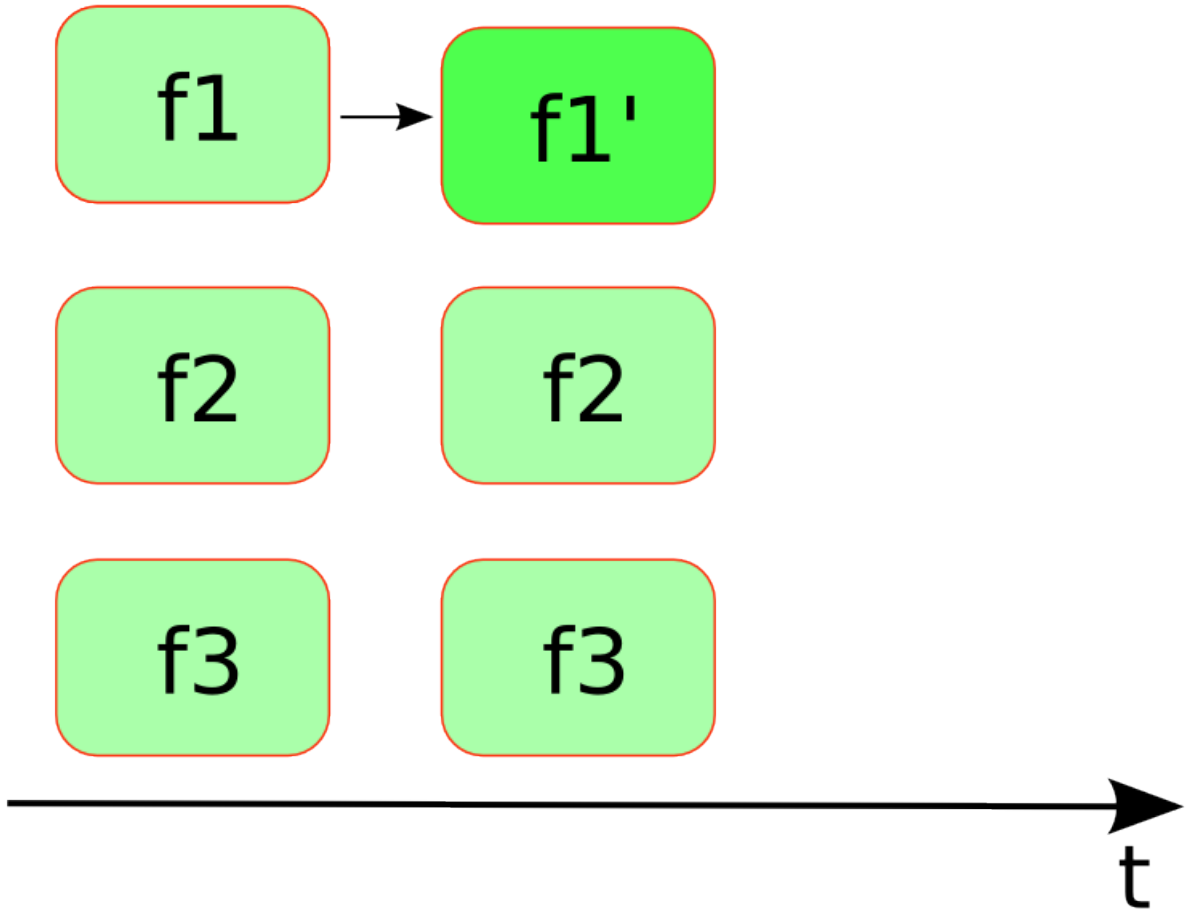
svn, um CVS melhorzinho

- Melhorou mesmo
  - Commits atômicos: controle de versão da árvore inteira.
- Manteve
  - Ainda depende-se de estar *on-line* para quase todas as operações.
  - Ainda depende-se de um repositório centralizado.

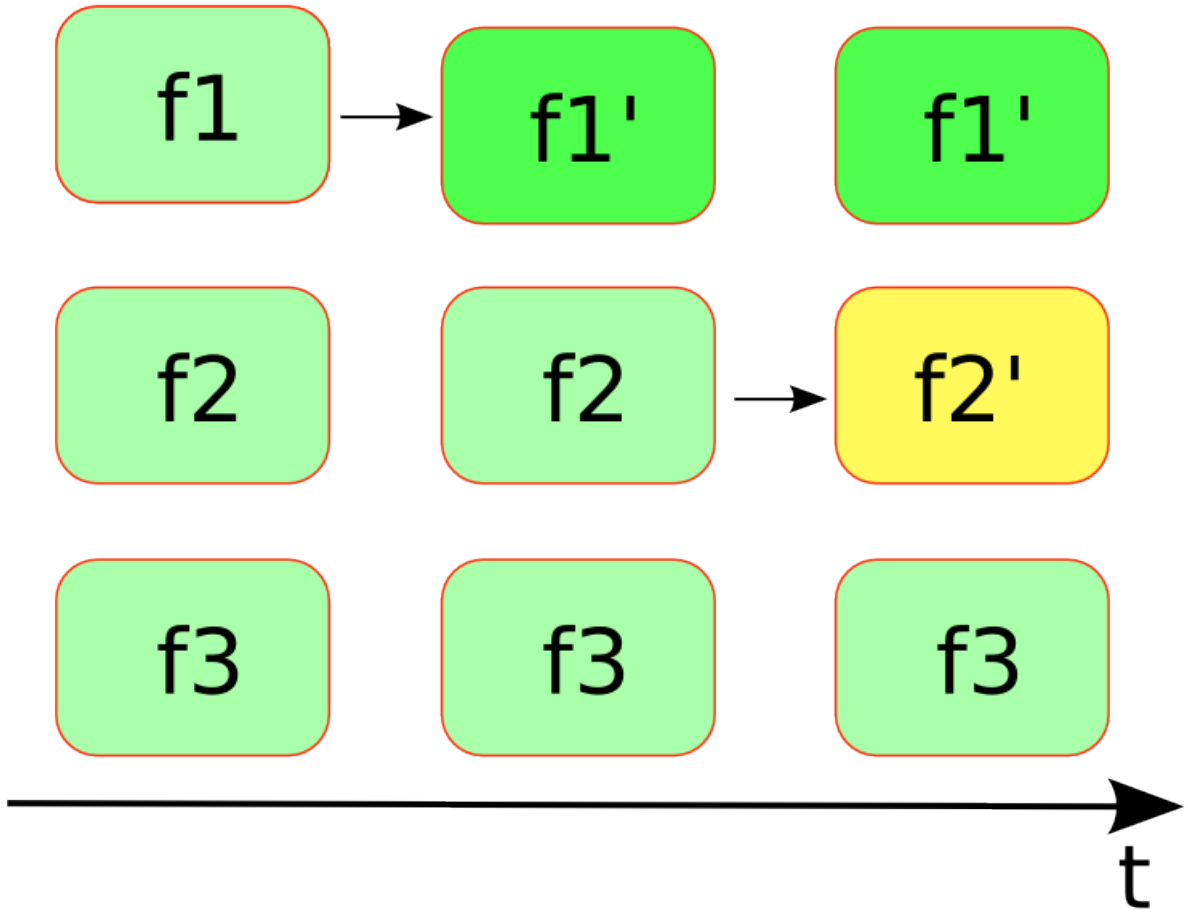
svn: exemplo



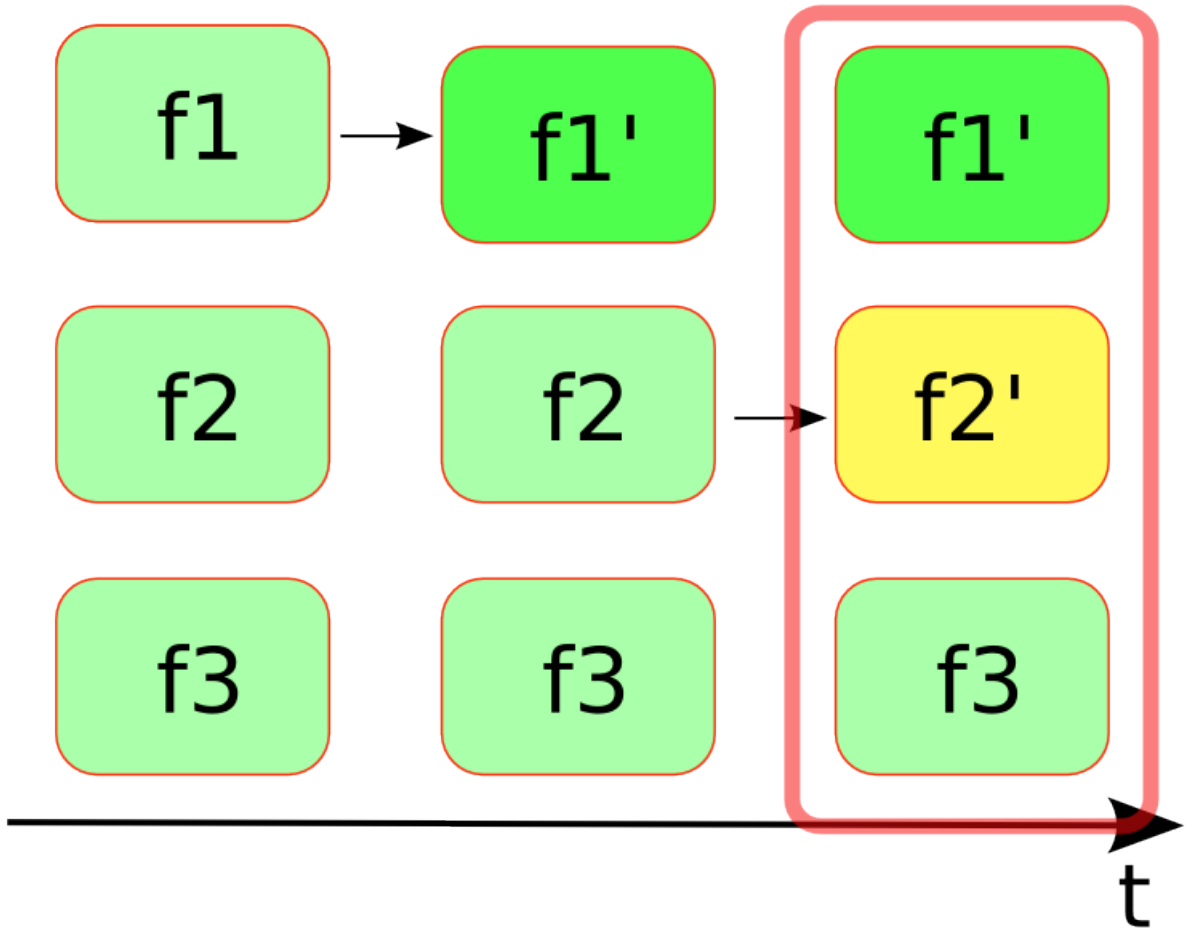
svn: exemplo



svn: exemplo



svn: exemplo



Problemas com controle de versão centralizado

Problemas com controle de versão centralizado

- Necessidade de estar sempre conectado.

Problemas com controle de versão centralizado

- Necessidade de estar sempre conectado.
- Desenvolvedor é forçado a resolver conflitos imediatamente.

Problemas com controle de versão centralizado

- Necessidade de estar sempre conectado.
- Desenvolvedor é forçado a resolver conflitos imediatamente.
- Repositório central é um ponto central de falha.

Problemas com controle de versão centralizado

- Necessidade de estar sempre conectado.
- Desenvolvedor é forçado a resolver conflitos imediatamente.
- Repositório central é um ponto central de falha.
- Necessidade de conceder permissão de escrita explicitamente emperra a colaboração.

Controle de versão Distribuído

Um sistema de controle de versão distribuído:

Controle de versão Distribuído

Um sistema de controle de versão distribuído:

- não depende de um repositório central.

Controle de versão Distribuído

Um sistema de controle de versão distribuído:

- não depende de um repositório central.
- permite que diferentes usuários evoluam em direções diferentes (*branches*) a partir de um mesmo ponto comum.

Controle de versão Distribuído

Um sistema de controle de versão distribuído:

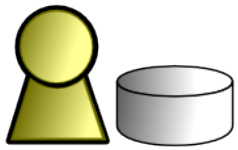
- não depende de um repositório central.
- permite que diferentes usuários evoluam em direções diferentes (*branches*) a partir de um mesmo ponto comum.
- oferece ferramentas para fazer *merge* de diferentes *branches* de volta ao *branch* principal.

Controle de versão Distribuído

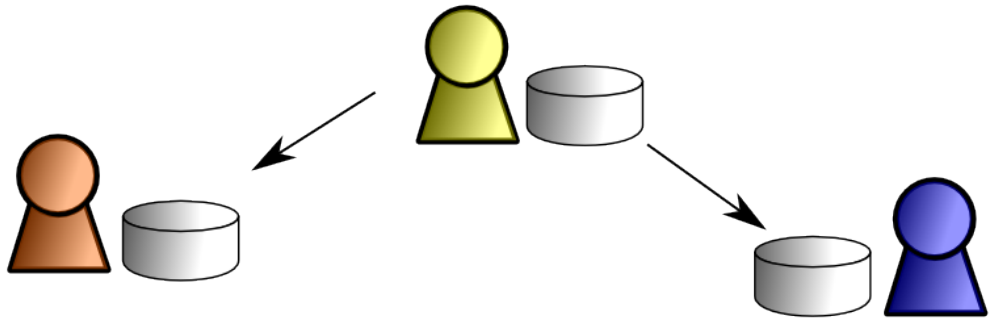
Um sistema de controle de versão distribuído:

- não depende de um repositório central.
- permite que diferentes usuários evoluam em direções diferentes (*branches*) a partir de um mesmo ponto comum.
- oferece ferramentas para fazer *merge* de diferentes *branches* de volta ao *branch* principal.
- (entre outras características)

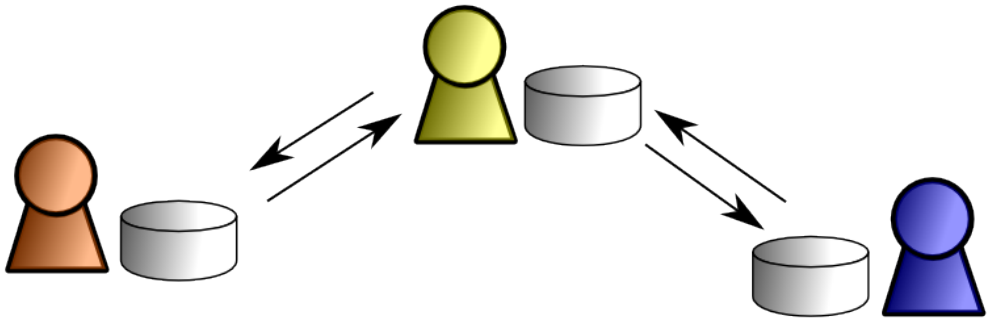
Controle de versão distribuído



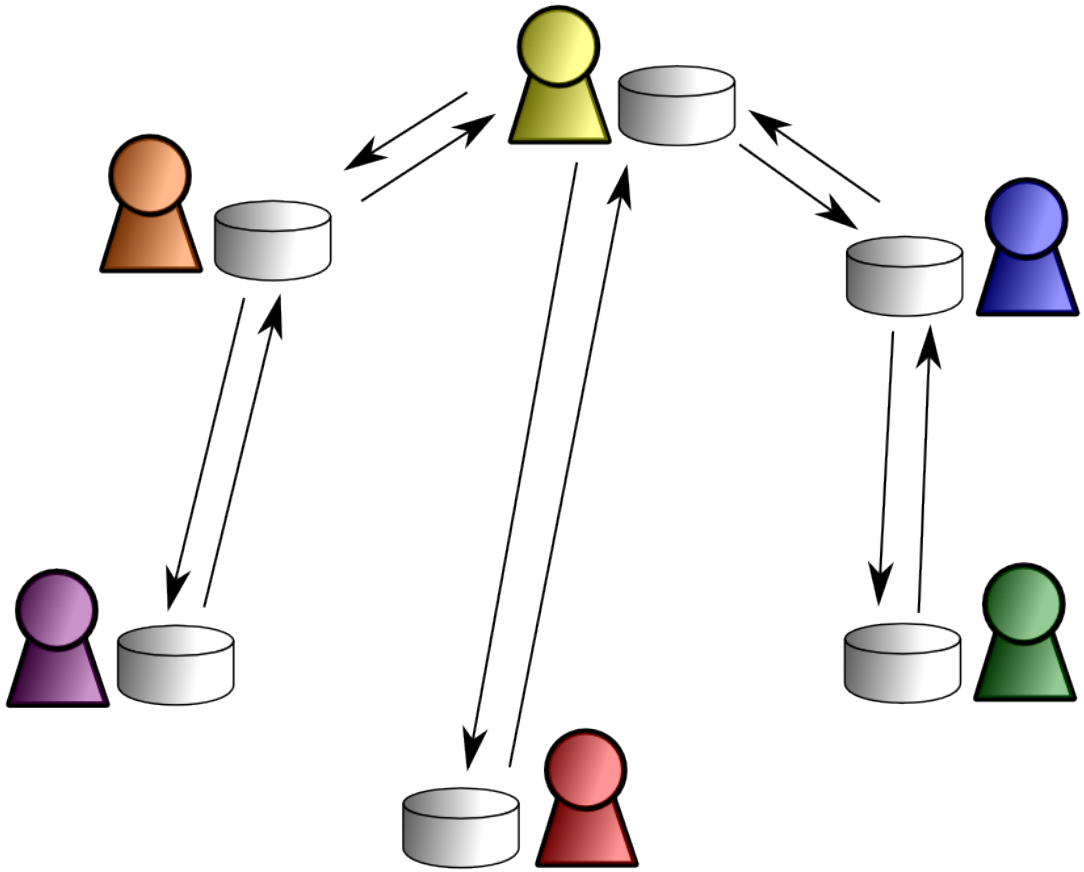
Controle de versão distribuído



Controle de versão distribuído



Controle de versão distribuído



Resumindo a história

Figura: Evolução dos VCS, do ponto de vista do usuário

Roteiro

Sobre o git

- `http://git-scm.com/`
- Pacotes disponíveis para diversos sistemas operacionais.
- Principal interface: programa `git` (linha de comando).

Iniciando um repositório `git`

- `git init`
  Inicializa o repositório, criando um diretório `.git` na raiz do diretório atual.

Iniciando um repositório git

- `git init`
  Inicializa o repositório, criando um diretório `.git` na raiz do diretório atual.
- `git add [ ARQ1 ARQ2 ...]`
  Informa ao `git` sobre arquivos que devem ser mantidos sob controle de versão.

Iniciando um repositório git

- `git init`
  Inicializa o repositório, criando um diretório `.git` na raiz do diretório atual.
- `git add [ ARQ1 ARQ2 ...]`
  Informa ao `git` sobre arquivos que devem ser mantidos sob controle de versão.
- `git commit`
  Confirma alterações nos arquivos mantidos sob controle de versão (ou que estão sendo incluídos).

init, add, commit

```
terceiro@morere:/tmp$ mkdir test
terceiro@morere:/tmp$ cd test/
terceiro@morere:/tmp/test$ git init
Initialized empty Git repository in /tmp/test/.git/
terceiro@morere:/tmp/test (master)$ edit README.txt
terceiro@morere:/tmp/test (master)$ edit test.c
terceiro@morere:/tmp/test (master)$ git add README.txt test.c
terceiro@morere:/tmp/test (master)$ git commit -m 'Adding initial version'
Created initial commit 3f42e5e: Adding initial version
 2 files changed, 6 insertions(+), 0 deletions(-)
 create mode 100644 README.txt
 create mode 100644 test.c
terceiro@morere:/tmp/test (master)$
```

Verificando trabalho realizado

- `git diff`
  Lista as diferenças entre o conteúdo atual do diretório e o último *commit*.

Verificando trabalho realizado

- `git diff`
  Lista as diferenças entre o conteúdo atual do diretório e o último *commit*.
- `git log`
  Lista todos os *commits* realizados no *branch* atual.

Verificando trabalho realizado

- `git diff`
  Lista as diferenças entre o conteúdo atual do diretório e o último *commit*.
- `git log`
  Lista todos os *commits* realizados no *branch* atual.
- `git show [COMMIT]`
  Mostra o conteúdo de `COMMIT`, ou do topo do *branch* atual se `COMMIT` é omitido.

git diff

```
terceiro@morere:/tmp/test (master)$ edit test.c
terceiro@morere:/tmp/test (master)$ git diff
diff --git a/test.c b/test.c
index 1394ce8..949743d 100644
--- a/test.c
+++ b/test.c
@@ -1,5 +1,6 @@
 #include <stdio.h>

 int main() {
+  printf("Hello, world!");
   return 0;
 }
terceiro@morere:/tmp/test (master)$ git add test.c
terceiro@morere:/tmp/test (master)$ git commit -m 'printing "Hello, world"'
Created commit de634bc: printing "Hello, world"
 1 files changed, 1 insertions(+), 0 deletions(-)
terceiro@morere:/tmp/test (master)$
```

git log

```
terceiro@morere:/tmp/test (master)$ git log
commit de634bc41062167b451f3eab8c22bc4424c1afe0
Author: Antonio Terceiro <terceiro@softwarelivre.org>
Date:   Tue Nov 4 19:57:59 2008 -0300

    printing "Hello, world"

commit 3f42e5ed6007bc320c1f0447eb0fb47501fab2cb
Author: Antonio Terceiro <terceiro@softwarelivre.org>
Date:   Tue Nov 4 19:42:56 2008 -0300

    Adding initial version
terceiro@morere:/tmp/test (master)$
```

git show

```
terceiro@morere:/tmp/test (master)$ git show de634bc41062167b451f3eab8c22bc4424c
1afe0
commit de634bc41062167b451f3eab8c22bc4424c1afe0
Author: Antonio Terceiro <terceiro@softwarelivre.org>
Date:   Tue Nov 4 19:57:59 2008 -0300

    printing "Hello, world"

diff --git a/test.c b/test.c
index 1394ce8..949743d 100644
--- a/test.c
+++ b/test.c
@@ -1,5 +1,6 @@
 #include <stdio.h>

 int main() {
+  printf("Hello, world!");
   return 0;
 }
terceiro@morere:/tmp/test (master)$
```

Ferramentas gráficas

- `gitk`.
- `giggle`.
- outras?

Branches

- `git branch [NOVOBRANCH BRANCHANTIGO]`
  Cria um novo *branch* com nome `NOVOBRANCH`, a partir do ponto onde está `BRANCHANTIGO`.

Branches

- `git branch [NOVOBRANCH BRANCHANTIGO]`
  Cria um novo *branch* com nome `NOVOBRANCH`, a partir do ponto onde está `BRANCHANTIGO`.
- `git branch`
  Lista os branches existentes.

Branches

- `git branch [NOVOBRANCH BRANCHANTIGO]`
  Cria um novo *branch* com nome `NOVOBRANCH`, a partir do ponto onde está `BRANCHANTIGO`.
- `git branch`
  Lista os branches existentes.
- `git checkout [BRANCH]`
  Muda para o *branch* `BRANCH`.

Branches

- `git branch [NOVOBRANCH BRANCHANTIGO]`
  Cria um novo *branch* com nome `NOVOBRANCH`, a partir do ponto onde está `BRANCHANTIGO`.
- `git branch`
  Lista os branches existentes.
- `git checkout [BRANCH]`
  Muda para o *branch* `BRANCH`.
- `git merge [BRANCH]`
  Combina no *branch* atual as mudanças que estão no *branch* `BRANCH` e que não estão no *branch* atual.

branch, merge

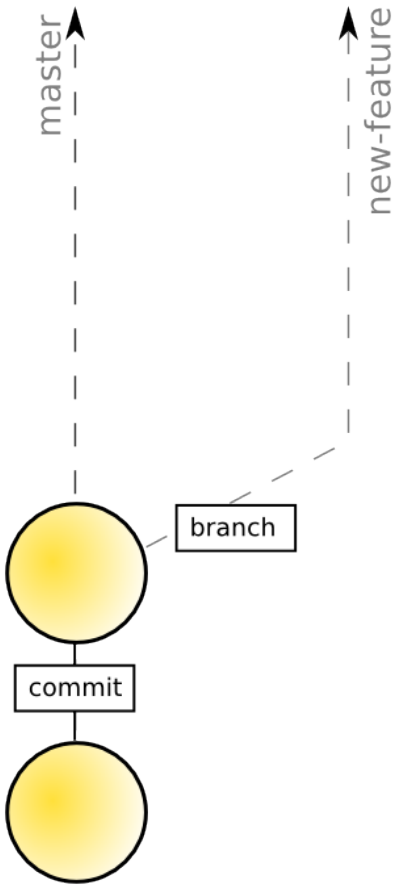
```
terceiro@morere:/tmp/test (master)$ git branch
* master
terceiro@morere:/tmp/test (master)$ git branch encapsulate-messages master
terceiro@morere:/tmp/test (master)$ git checkout encapsulate-messages
Switched to branch "encapsulate-messages"
terceiro@morere:/tmp/test (encapsulate-messages)$ edit test.c
terceiro@morere:/tmp/test (encapsulate-messages)$ git add test.c
terceiro@morere:/tmp/test (encapsulate-messages)$ git commit -m 'encapsulating m
essages in calls to the "say" function'
Created commit 22e5e15: encapsulating messages in calls to the "say" function
 1 files changed, 5 insertions(+), 1 deletions(-)
terceiro@morere:/tmp/test (encapsulate-messages)$ git checkout master
Switched to branch "master"
terceiro@morere:/tmp/test (master)$ git log master..encapsulate-messages
commit 22e5e151b4bf532ca9128bd71a99a65ebaf9d6dd
Author: Antonio Terceiro <terceiro@softwarelivre.org>
Date:   Tue Nov 4 21:06:02 2008 -0300

    encapsulating messages in calls to the "say" function
terceiro@morere:/tmp/test (master)$ git merge encapsulate-messages
Updating de634bc..22e5e15
Fast forward
 test.c |    6 +++++-
 1 files changed, 5 insertions(+), 1 deletions(-)
terceiro@morere:/tmp/test (master)$
```

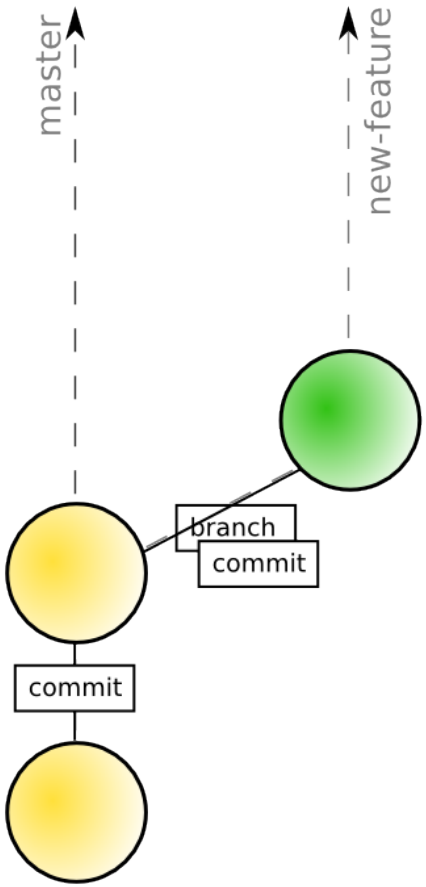
Branching e *merging*, graficamente

Branching e *merging*, graficamente

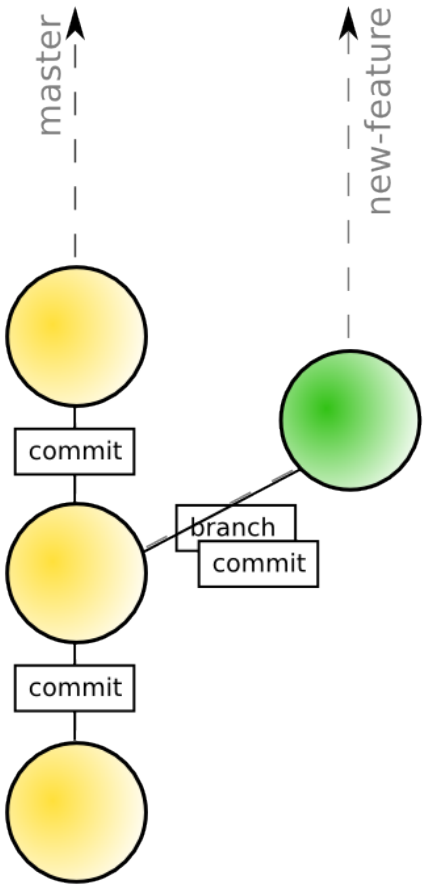
Branching e *merging*, graficamente



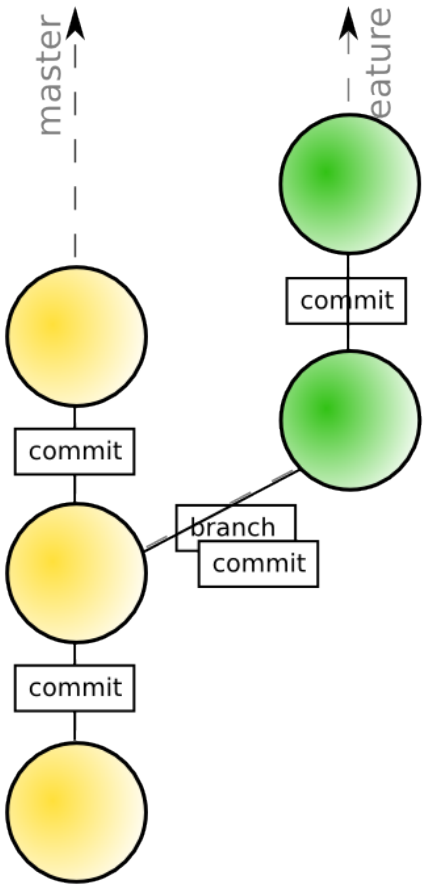
Branching e *merging*, graficamente



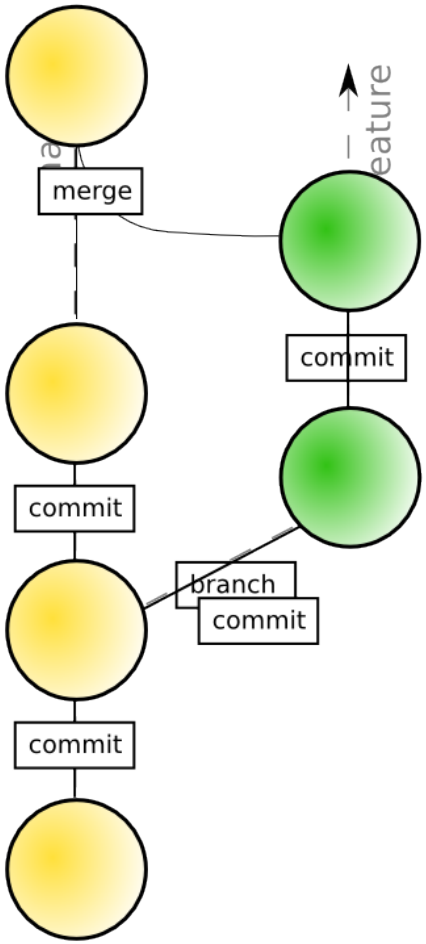
Branching e *merging*, graficamente



Branching e *merging*, graficamente



Branching e *merging*, graficamente



Os dois tipos de merge

- *merge* "tradicional": quando os dois *branches* já divergiram um o outro.
  - Acabamos de ver um desses.
  - Commit de *merge* tem dois pais, o que faz o histórico deixar de ser linear.

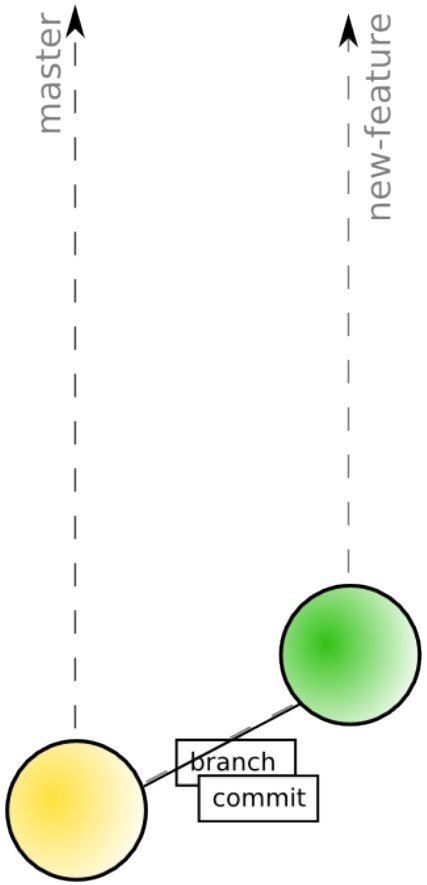
Os dois tipos de merge

- *merge* "tradicional": quando os dois *branches* já divergiram um o outro.
  - Acabamos de ver um desses.
  - Commit de *merge* tem dois pais, o que faz o histórico deixar de ser linear.
- *Fast forward*: quando o *branch* principal não tem atualizações em relação ao ponto a partir do qual o *branch* do qual está se fazendo *merge* foi criado.
  - Não é necessária a criação de outro *commit*.
  - A ponta do *branch* principal simplesmente é movida para a ponta do novo *branch*.
  - histórico de mantém linear.

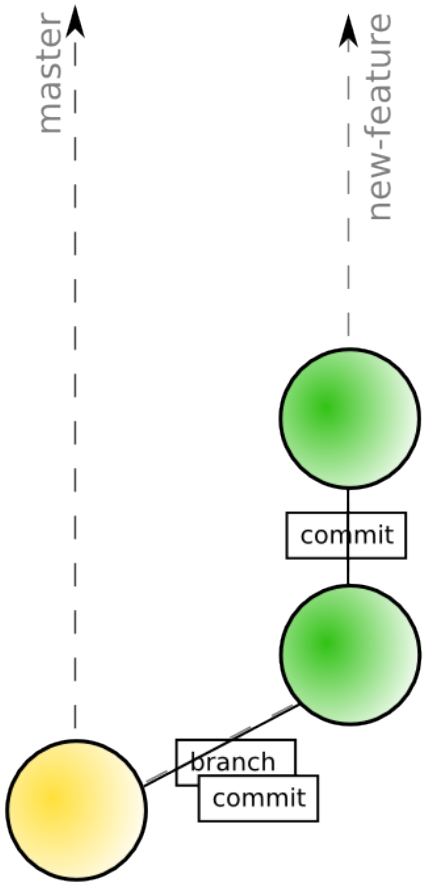
Fast forward

Fast forward

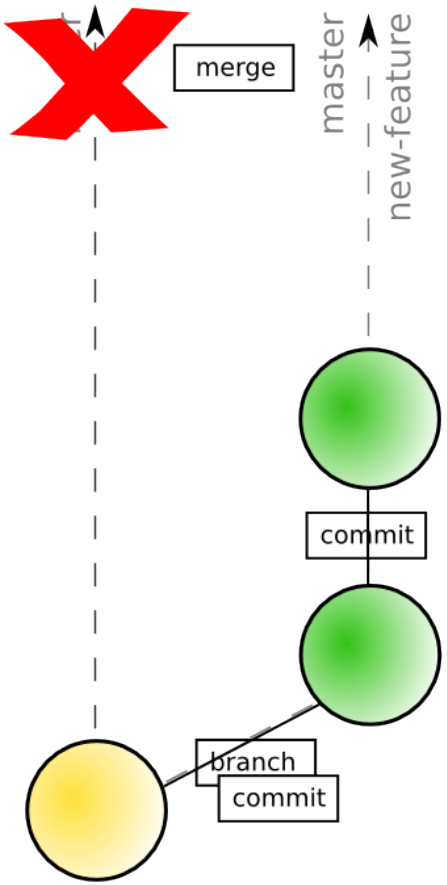
Fast forward



Fast forward



Fast forward



Rebase: reescrevendo o histórico

- `git rebase BRANCH`
  Reaplica as mudanças realizadas no *branch* atual a partir do ponto em que o *branch* `BRANCH` foi criado em relação ao estado atual do *branch* `BRANCH`, e move o branch atual para o último *commit* resultante.

Rebase: reescrevendo o histórico

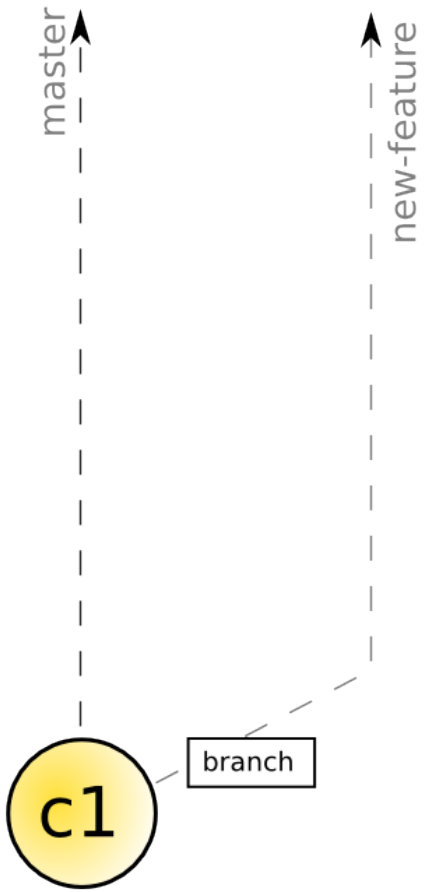
- `git rebase BRANCH`
  Reaplica as mudanças realizadas no *branch* atual a partir do ponto em que o *branch* `BRANCH` foi criado em relação ao estado atual do *branch* `BRANCH`, e move o branch atual para o último *commit* resultante.
- Usado para linearizar o histórico.

Rebase: reescrevendo o histórico

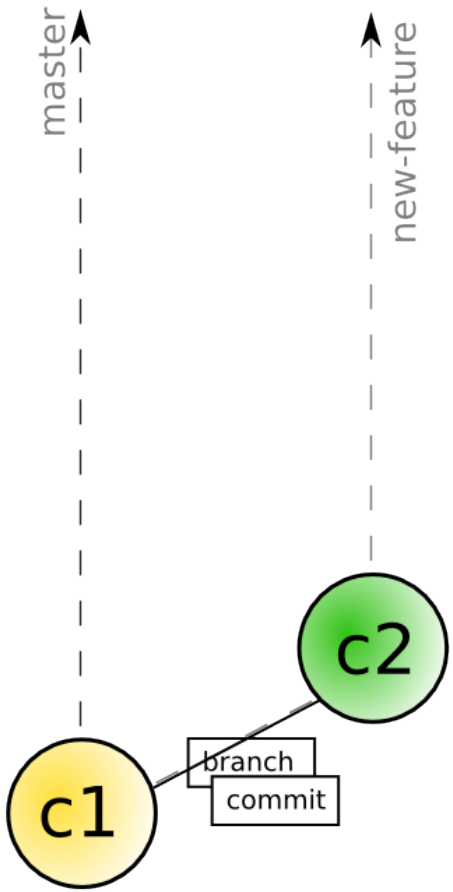
- `git rebase BRANCH`
  Reaplica as mudanças realizadas no *branch* atual a partir do ponto em que o *branch* `BRANCH` foi criado em relação ao estado atual do *branch* `BRANCH`, e move o branch atual para o último *commit* resultante.
- Usado para linearizar o histórico.
- Não deve ser usado se o estado do branch atual já foi publicado.

rebase, graficamente

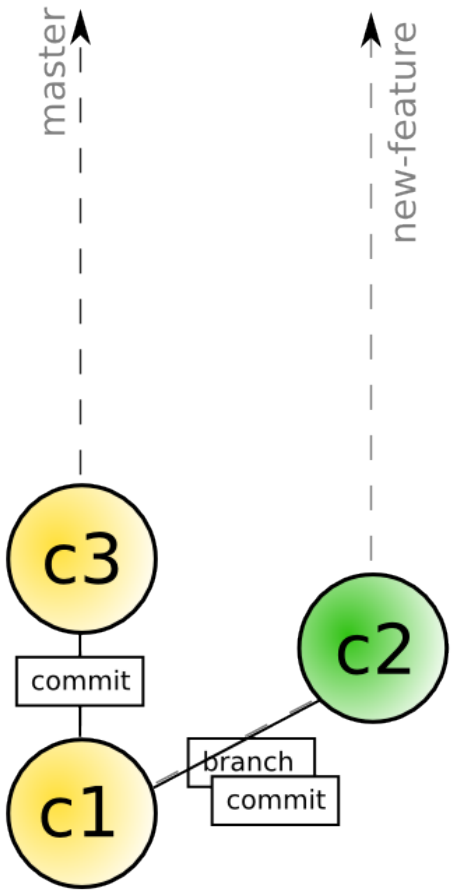
rebase, graficamente



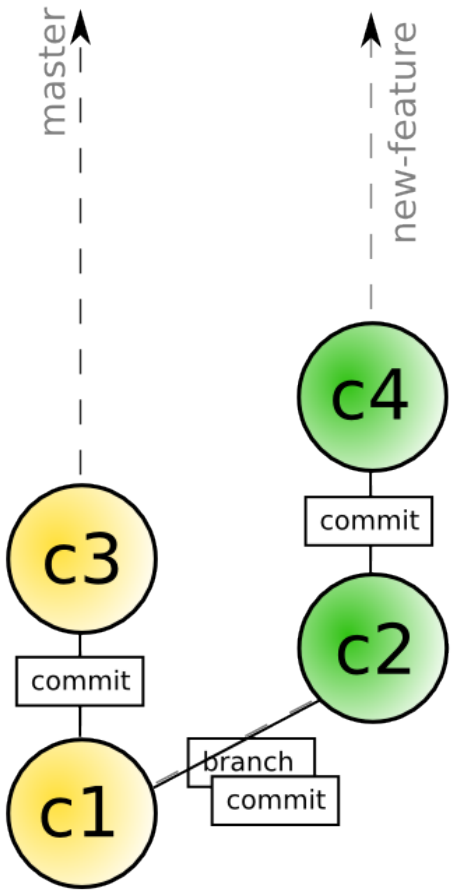
rebase, graficamente



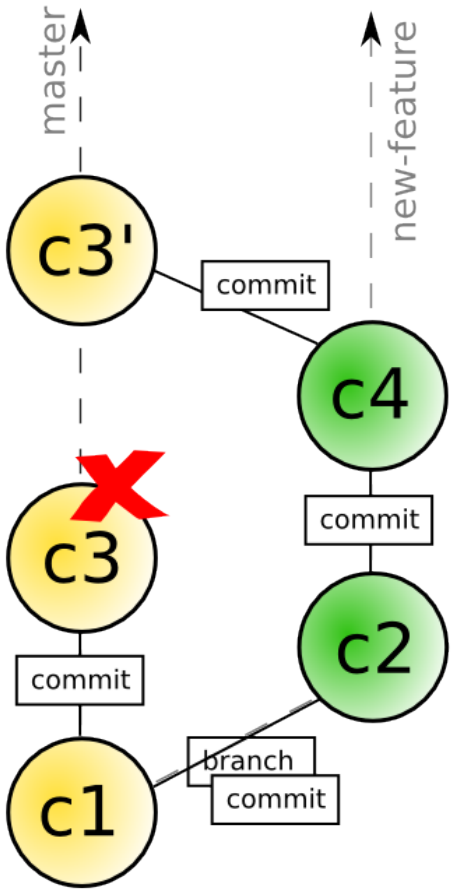
rebase, graficamente



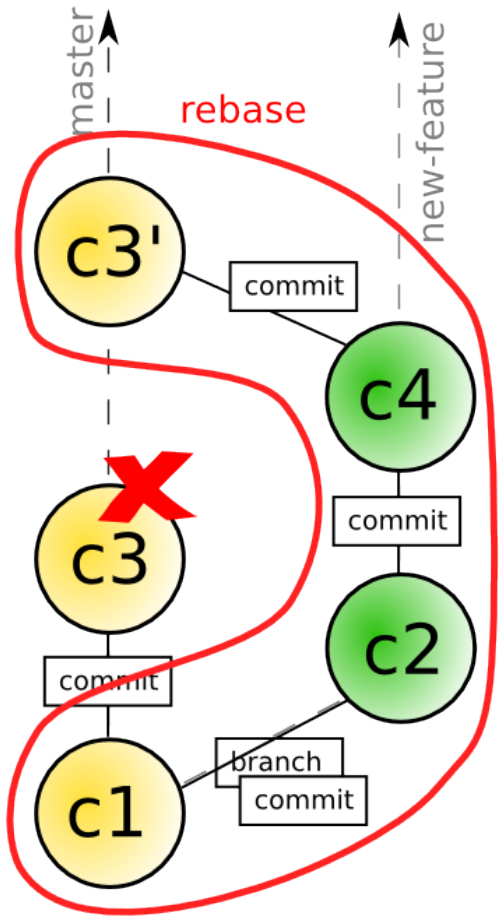
rebase, graficamente



rebase, graficamente



rebase, graficamente



Clonando um repositório

- `git clone URL`
  Faz um clone local de um repositório público localizado em `URL`.

Repositórios remotos em geral

- `git remote add NAME URL`
  Associa o nome `NAME` ao repositório remoto publicado em `URL`

Repositórios remotos em geral

- `git remote add NAME URL`
  Associa o nome `NAME` ao repositório remoto publicado em `URL`
- `git fetch NAME`
  Faz download dos objetos que estão no repositório remoto `NAME` mas que não estão no repositório local.

Repositórios remotos em geral

- `git remote add NAME URL`
  Associa o nome `NAME` ao repositório remoto publicado em `URL`
- `git fetch NAME`
  Faz download dos objetos que estão no repositório remoto `NAME` mas que não estão no repositório local.
- `git clone URL` é mais ou menos equivalente a:
  - `git init`
  - `git remote add origin URL`
  - `git fetch origin`
  - `git branch --track master origin/master`

Interagindo com repositórios remotos

- `git fetch` de tempos em tempos
- `git merge origin/master` para aplicar as mudanças no *branch* `master` remoto (*"upstream"*) ao *branch* (`master`?) local.

Interagindo com repositórios remotos

- `git fetch` de tempos em tempos
- `git merge origin/master` para aplicar as mudanças no *branch* `master` remoto (*"upstream"*) ao *branch* (`master`?) local.
- `git pull` faz as duas coisas automaticamente:
  - baixa os objetos presentes no repositório remoto de onde o *branch* local atual foi derivado.
  - Aplica as mudanças necessárias para fazer o *branch* local atual igual ao *branch* remoto original.

Interagindo com repositórios remotos

- `git fetch` de tempos em tempos
- `git merge origin/master` para aplicar as mudanças no *branch* `master` remoto (*"upstream"*) ao *branch* (`master`?) local.
- `git pull` faz as duas coisas automaticamente:
  - baixa os objetos presentes no repositório remoto de onde o *branch* local atual foi derivado.
  - Aplica as mudanças necessárias para fazer o *branch* local atual igual ao *branch* remoto original.

  É mais ou menos equivalente a:
  - `git fetch origin`
  - `git merge origin/master`

Publicando um repositório

Publicando um repositório

- `git remote add public [user@]server:/path/to/repo.git`
  Deve existir um repositório em `/path/to/repo.git` no servidor remoto `server`, e `user` deve ter acesso *ssh* à máquina.

Publicando um repositório

- `git remote add public [user@]server:/path/to/repo.git`
  Deve existir um repositório em `/path/to/repo.git` no servidor remoto `server`, e `user` deve ter acesso *ssh* à máquina.
- `git push public [master BRANCH1 BRANCH2 ...| --all]`
  Faz upload de todos os objetos presentes nos *branches* locais `master BRANCH1 BRANCH2 ...` que também não estejam no repositório `public`, e atualiza esses *branches* no repositório remoto para refletir seu estado no repositório local.

Publicando um repositório

- `git remote add public [user@]server:/path/to/repo.git`
  Deve existir um repositório em `/path/to/repo.git` no servidor remoto `server`, e `user` deve ter acesso *ssh* à máquina.
- `git push public [master BRANCH1 BRANCH2 ...| --all]`
  Faz upload de todos os objetos presentes nos *branches* locais `master BRANCH1 BRANCH2 ...` que também não estejam no repositório `public`, e atualiza esses *branches* no repositório remoto para refletir seu estado no repositório local.
- `git push` por *default* empurra para o repositório "`origin`".

clone, pull

```
terceiro@morere:/tmp$ git clone git://github.com/terceiro/test.git
Initialized empty Git repository in /tmp/test/.git/
remote: Counting objects: 10, done.
remote: Compressing objects: 100% (8/8), done.
Receiving objects: 100% (10/10), done.
remote: Total 10 (delta 0), reused 0 (delta 0)
terceiro@morere:/tmp$ cd test
terceiro@morere:/tmp/test (master)$ edit README.txt
terceiro@morere:/tmp/test (master)$ git add README.txt
terceiro@morere:/tmp/test (master)$ git commit -m 'adding explanation in README
file'
Created commit 8b0caa4: adding explanation in README file
 1 files changed, 2 insertions(+), 0 deletions(-)
terceiro@morere:/tmp/test (master)$ git pull
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From git://github.com/terceiro/test
   f075991..fb97551  master     -> origin/master
Merge made by recursive.
 COPYING |    4 ++++
 1 files changed, 4 insertions(+), 0 deletions(-)
 create mode 100644 COPYING
terceiro@morere:/tmp/test (master)$
```

push

```
terceiro@morere:/tmp/test2 (master)$ git commit -a
Created commit fb97551: Adding COPYING file with a public domain statement\
 1 files changed, 4 insertions(+), 0 deletions(-)
 create mode 100644 COPYING
terceiro@morere:/tmp/test2 (master)$ git push
Counting objects: 4, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 468 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:terceiro/test.git
   f075991..fb97551  master -> master
terceiro@morere:/tmp/test2 (master)$
```

remote, fetch

```
terceiro@morere:/tmp/test2 (master)$ git remote add terceiro /tmp/test
terceiro@morere:/tmp/test2 (master)$ git fetch terceiro
remote: Counting objects: 8, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta remote: 0), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
From /tmp/test
 * [new branch]      master     -> terceiro/master
terceiro@morere:/tmp/test2 (master)$ git log master..terceiro/master --pretty=on
eline
128e15bac74a325c8e6744f89efc627c07488227 Merge branch 'master' of git://github.c
8b0caa4101ca73bc2f7c914a5ea019e533193a4d adding explanation in README file
terceiro@morere:/tmp/test2 (master)$
```

Roteiro

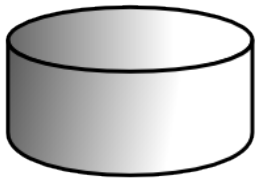
Objetivos

- Apresentar fluxos de trabalho comumente utilizados com o auxílio de sistemas de controle de versão.
- Identificar quais desses fluxos podem ser implementados com
  - sistemas de controle de versão centralizados.
  - sistemas de controle de versão distribuídos.
- Discutir aplicabilidade desses fluxos a diferentes contextos, suas vantagens e desvantagens.

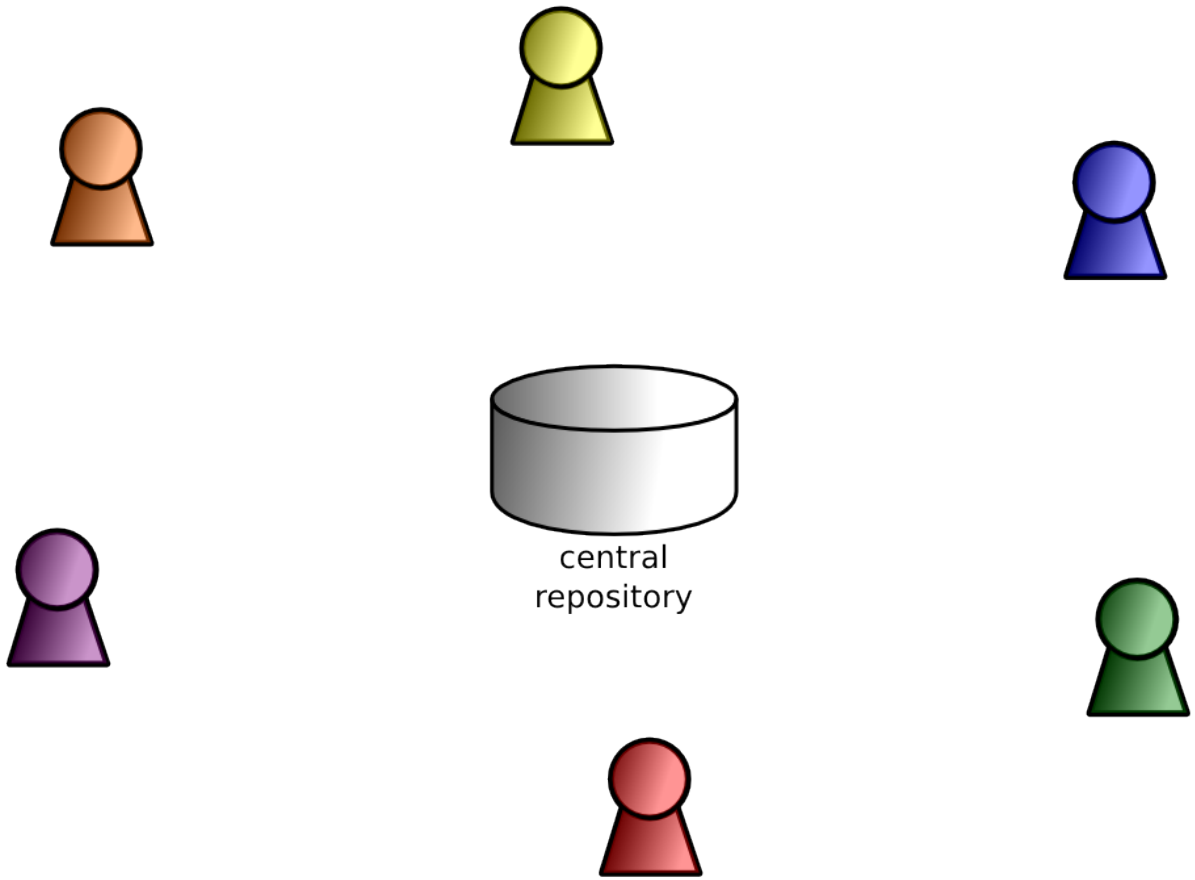
3 diferentes fluxos de trabalho

- Repositório central
- Gerente de Integração
- Ditador e Tenentes

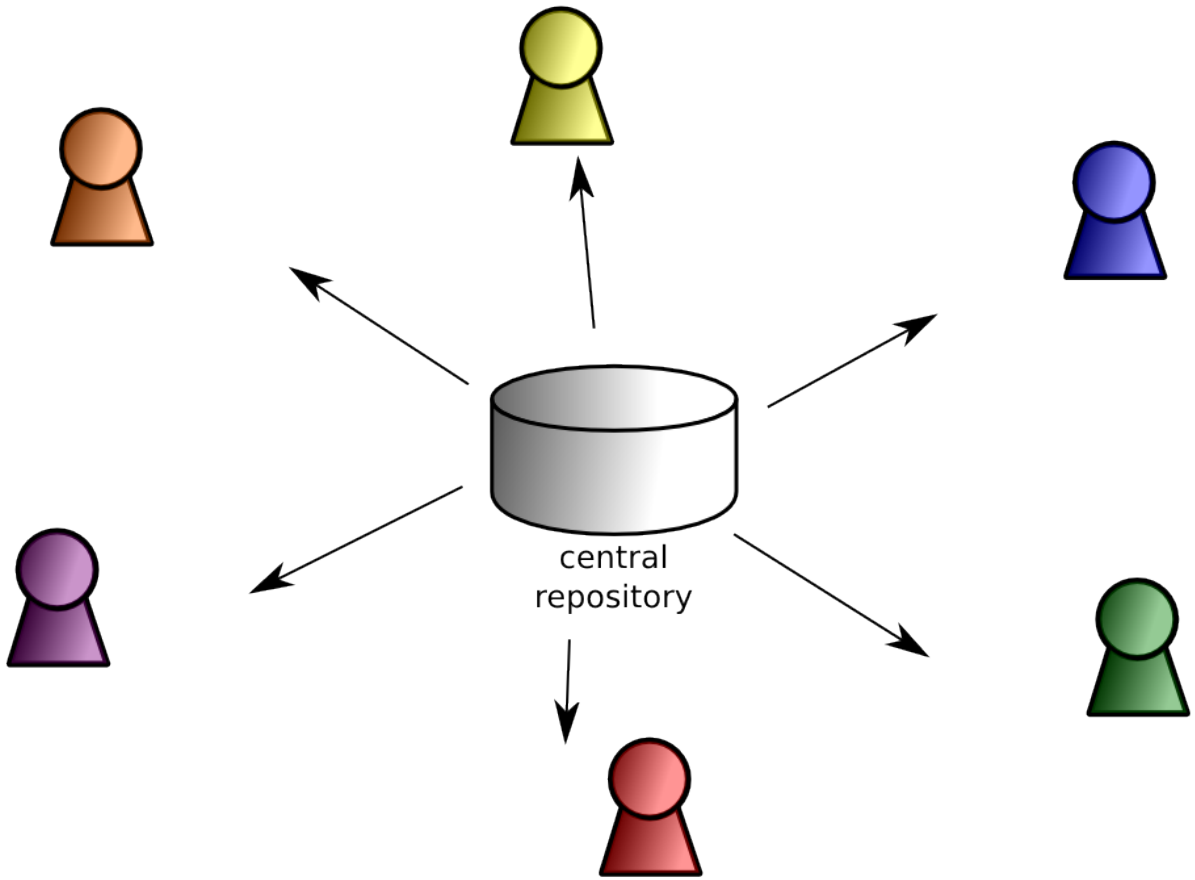
Repositório central



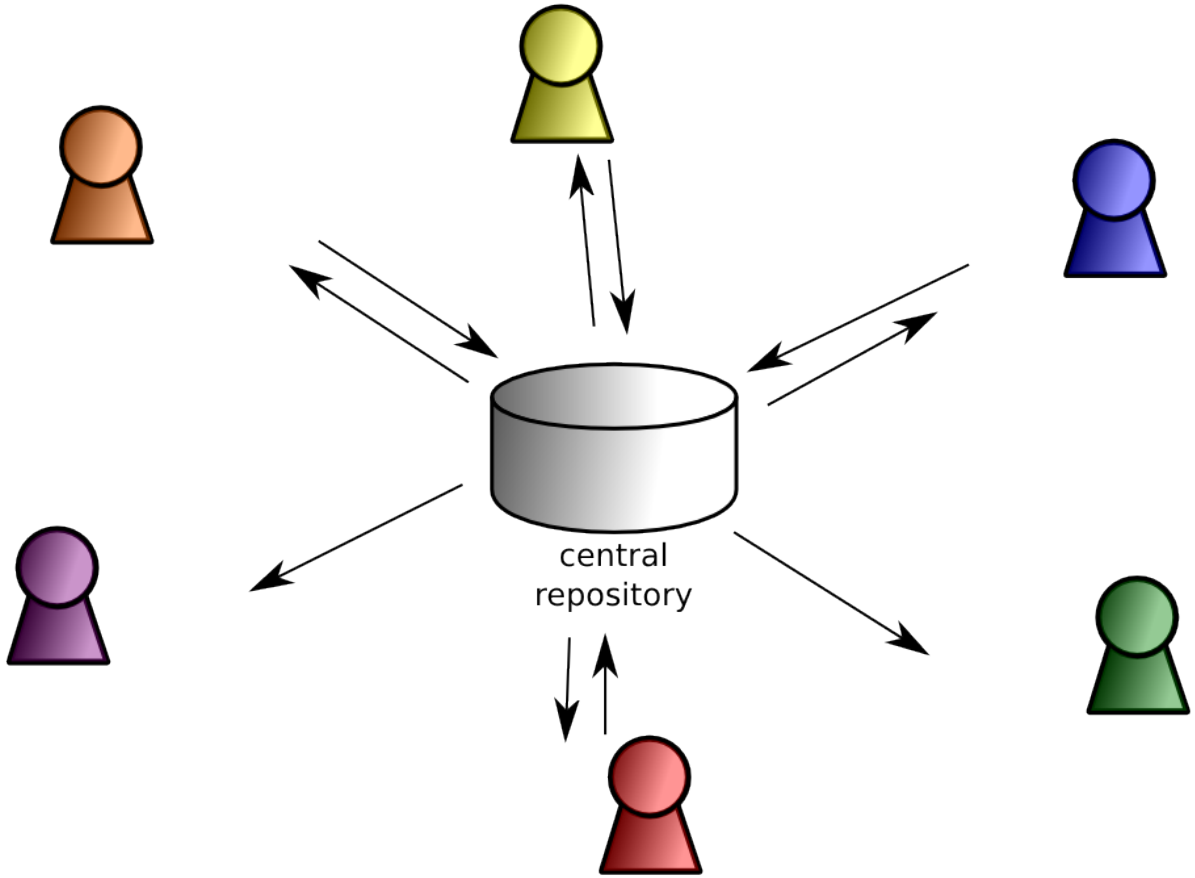
Repositório central



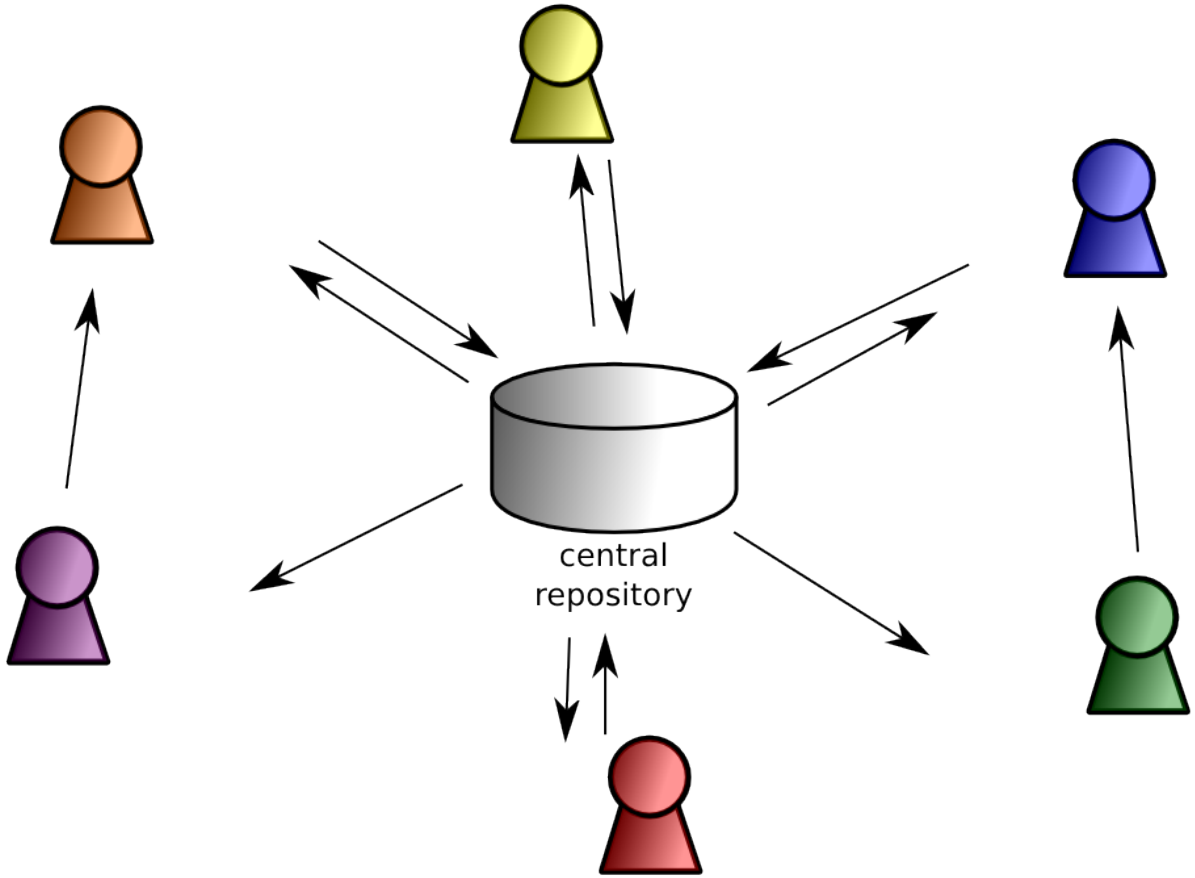
Repositório central



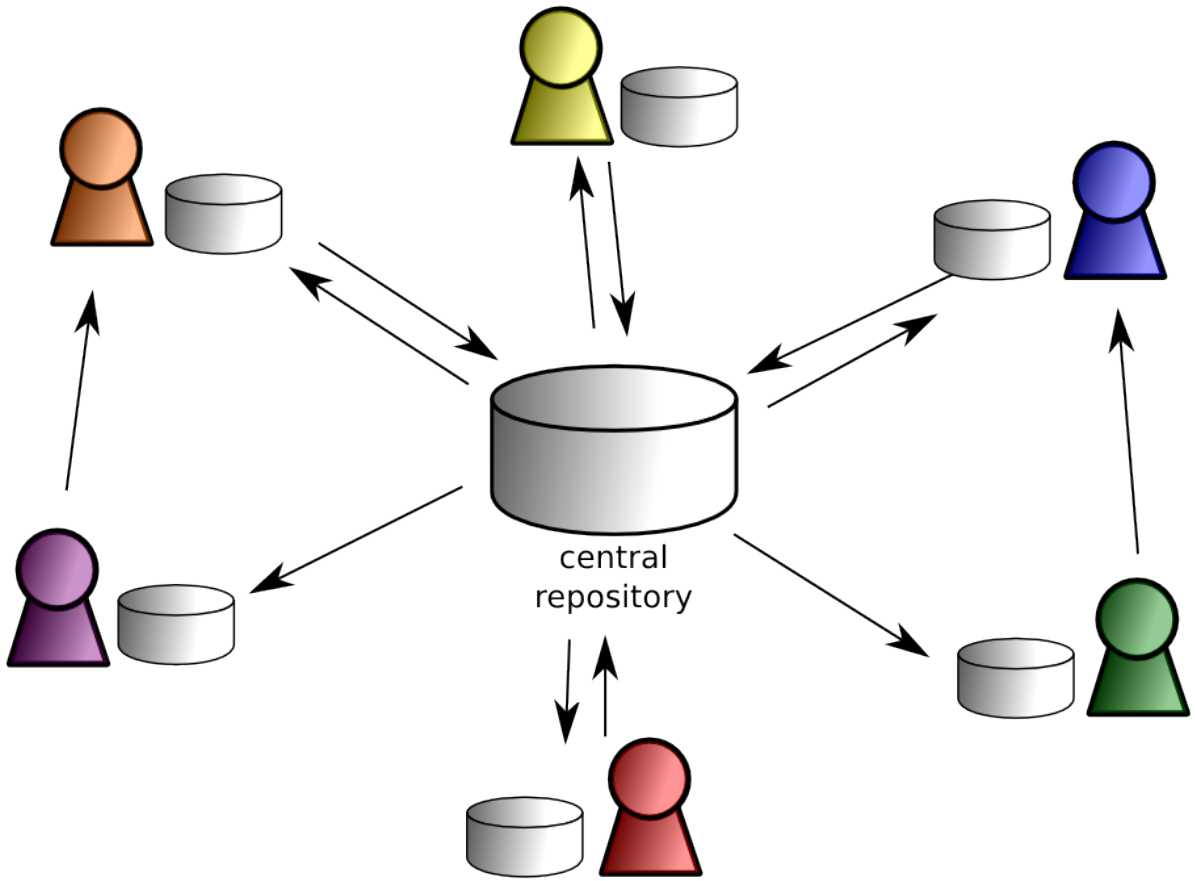
Repositório central



Repositório central



Repositório central



Repositório central – discussão

- Aplicável com controle de versão centralizado e distribuído
- Vantagens

- Desvantagens

Repositório central – discussão

- Aplicável com controle de versão centralizado e distribuído
- Vantagens
  - Paralelismo.
- Desvantagens

Repositório central – discussão

- Aplicável com controle de versão centralizado e distribuído
- Vantagens
  - Paralelismo.
  - “Simplicidade”.
- Desvantagens

Repositório central – discussão

- Aplicável com controle de versão centralizado e distribuído
- Vantagens
  - Paralelismo.
  - “Simplicidade”.
- Desvantagens
  - Contribuições diretas dependem de permissão explícita de escrita no repositório.

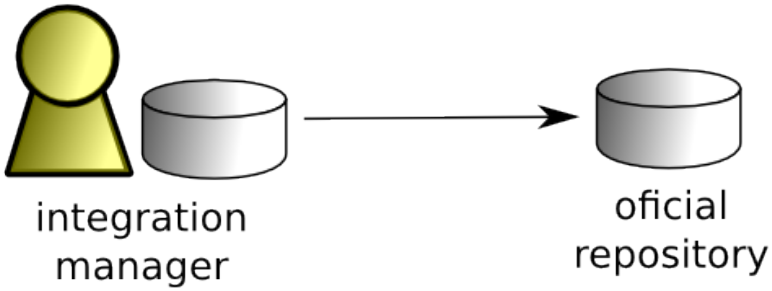
Repositório central – discussão

- Aplicável com controle de versão centralizado e distribuído
- Vantagens
  - Paralelismo.
  - "Simplicidade".
- Desvantagens
  - Contribuições diretas dependem de permissão explícita de escrita no repositório.
  - Sem um sistema de controle de versão distribuído, os contribuidores têm dificuldades em trabalhar em mais de uma funcionalidade/bug *ao mesmo tempo*.

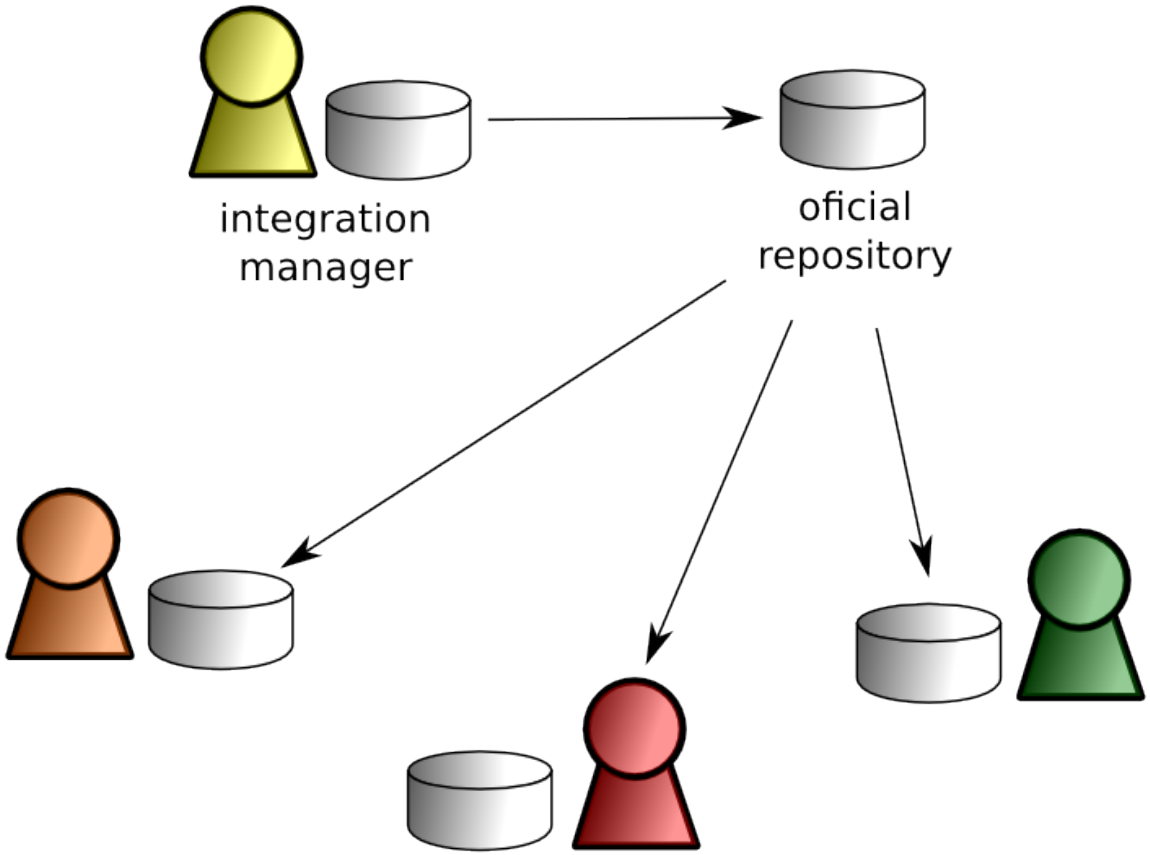
Repositório central – discussão

- Aplicável com controle de versão centralizado e distribuído
- Vantagens
  - Paralelismo.
  - "Simplicidade".
- Desvantagens
  - Contribuições diretas dependem de permissão explícita de escrita no repositório.
  - Sem um sistema de controle de versão distribuído, os contribuidores têm dificuldades em trabalhar em mais de uma funcionalidade/bug *ao mesmo tempo*.
  - Repositório central é um ponto de falha.

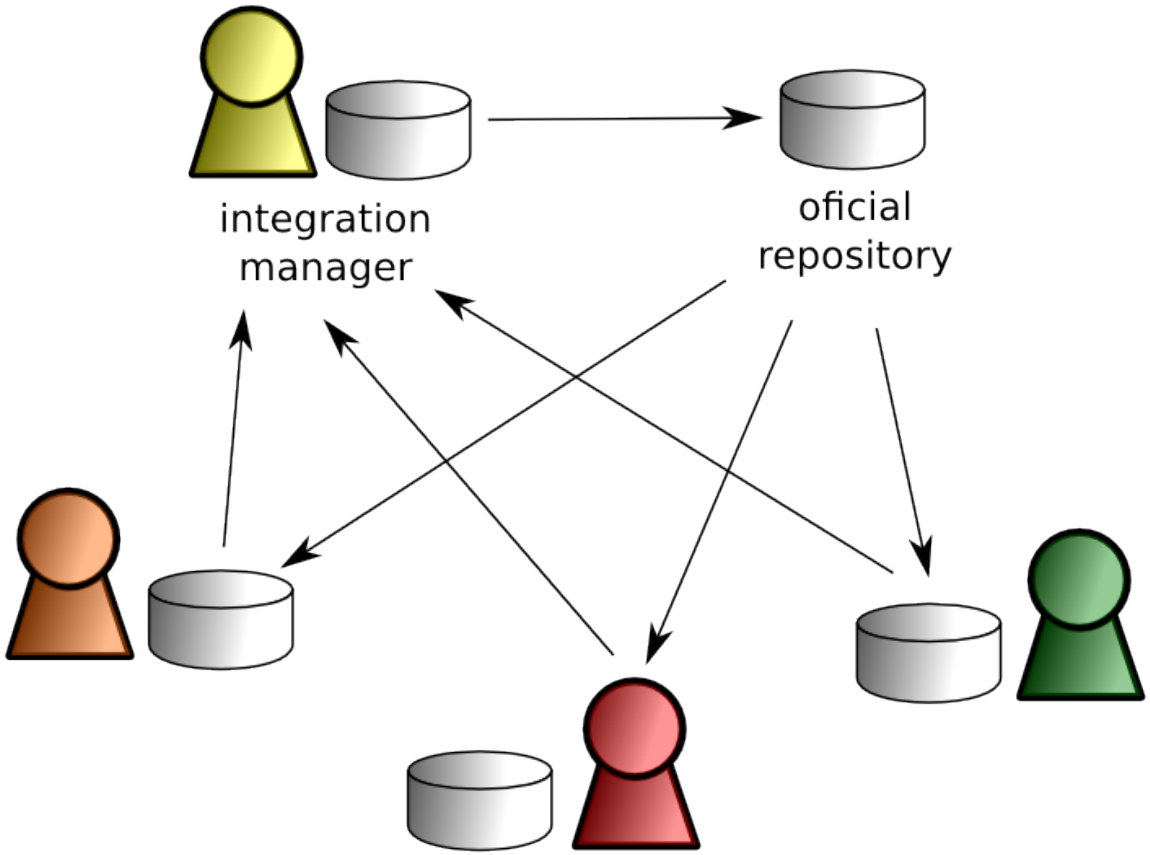
Gerente de Integração



Gerente de Integração



Gerente de Integração



Gerente de Integração – discussão

- Difícil de aplicar sem controle de versão distribuído.
- Mais interação social entre os desenvolvedores.
- A depender do contexto, o gerente de integração pode ser uma máquina.
- Vantagens

- Desvantagens

Gerente de Integração – discussão

- Difícil de aplicar sem controle de versão distribuído.
- Mais interação social entre os desenvolvedores.
- A depender do contexto, o gerente de integração pode ser uma máquina.
- Vantagens
  - Atividade explícita de inspeção/revisão/teste.
- Desvantagens

Gerente de Integração – discussão

- Difícil de aplicar sem controle de versão distribuído.
- Mais interação social entre os desenvolvedores.
- A depender do contexto, o gerente de integração pode ser uma máquina.
- Vantagens
  - Atividade explícita de inspeção/revisão/teste.
  - Maior controle sobre a base de código e sobre o design do software.
- Desvantagens

Gerente de Integração – discussão

- Difícil de aplicar sem controle de versão distribuído.
- Mais interação social entre os desenvolvedores.
- A depender do contexto, o gerente de integração pode ser uma máquina.
- Vantagens
  - Atividade explícita de inspeção/revisão/teste.
  - Maior controle sobre a base de código e sobre o design do software.
- Desvantagens
  - Gerente de Integração é um ponto de falha.

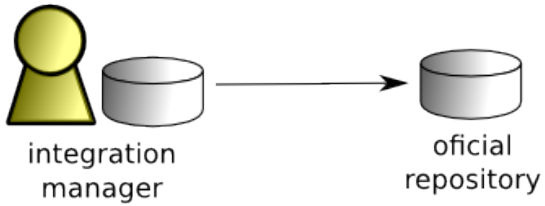
Gerente de Integração – discussão

- Difícil de aplicar sem controle de versão distribuído.
- Mais interação social entre os desenvolvedores.
- A depender do contexto, o gerente de integração pode ser uma máquina.
- Vantagens
  - Atividade explícita de inspeção/revisão/teste.
  - Maior controle sobre a base de código e sobre o design do software.
- Desvantagens
  - Gerente de Integração é um ponto de falha.
  - Problemas de escala se o gerente de integração for humano.

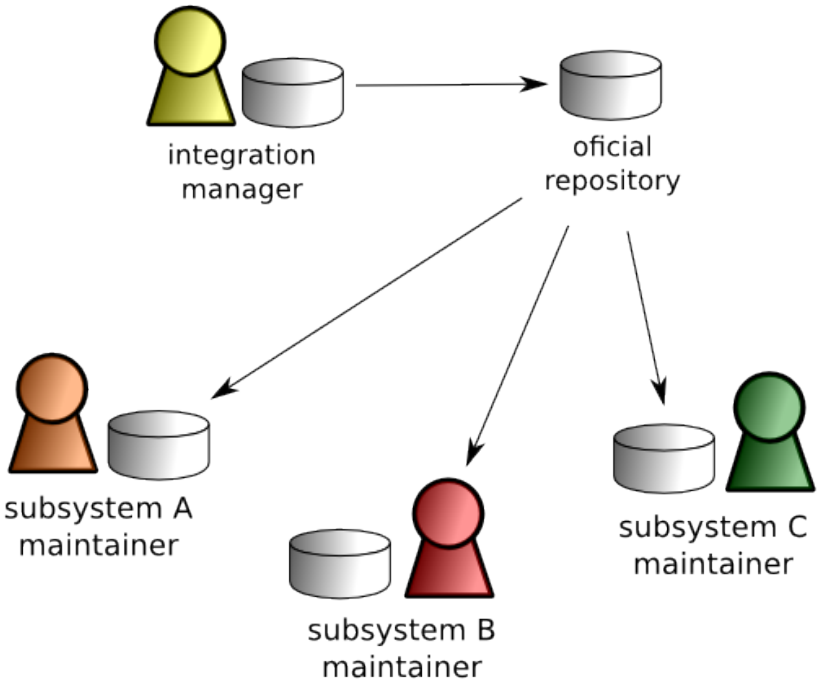
Gerente de Integração – discussão

- Difícil de aplicar sem controle de versão distribuído.
- Mais interação social entre os desenvolvedores.
- A depender do contexto, o gerente de integração pode ser uma máquina.
- Vantagens
  - Atividade explícita de inspeção/revisão/teste.
  - Maior controle sobre a base de código e sobre o design do software.
- Desvantagens
  - Gerente de Integração é um ponto de falha.
  - Problemas de escala se o gerente de integração for humano.
  - Pode ser difícil para equipes inexperientes.

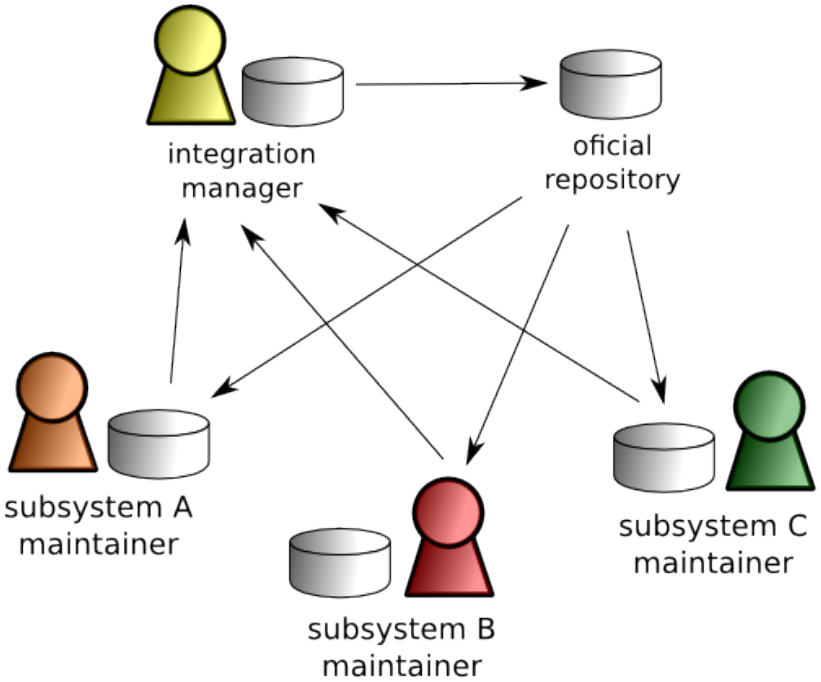
Ditador e Tenentes



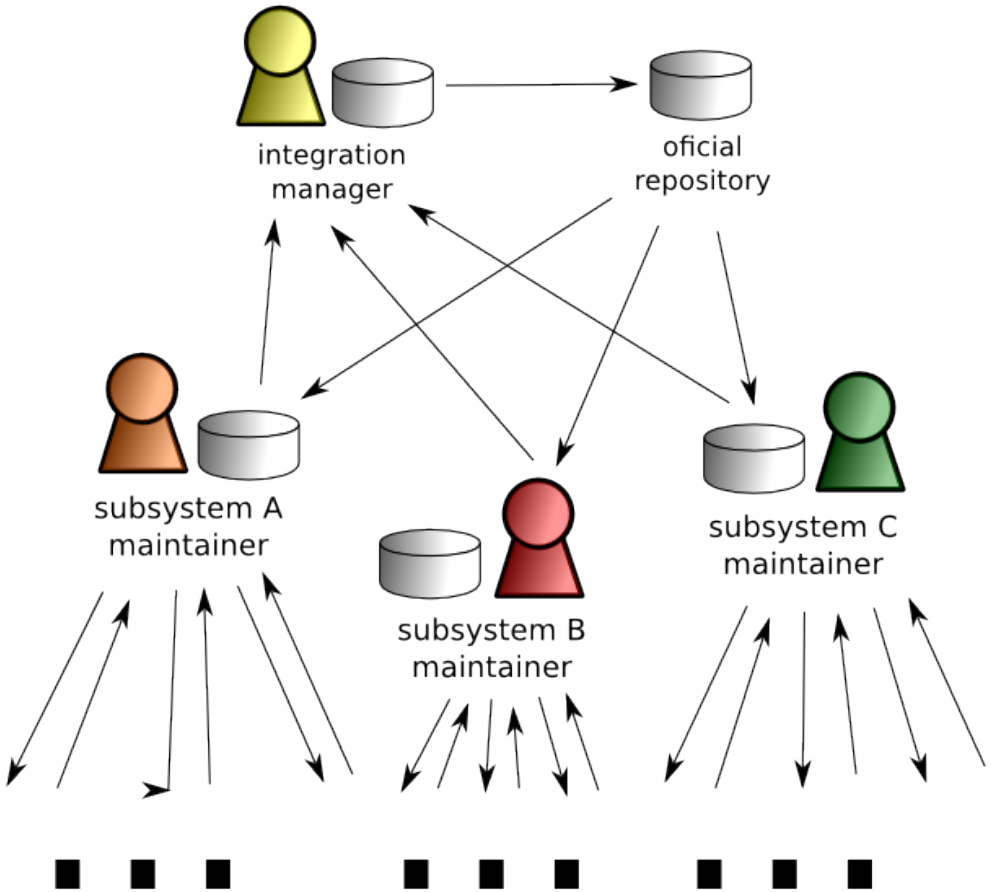
Ditador e Tenentes



Ditador e Tenentes



Ditador e Tenentes



Ditador e Tenentes – discussão

- Uma variação recursiva de "Gerente de integração".
- Praticamente impossível sem um sistema de controle de versão distribuído.
- Vantagens

- Desvantagens

Ditador e Tenentes – discussão

- Uma variação recursiva de "Gerente de integração".
- Praticamente impossível sem um sistema de controle de versão distribuído.
- Vantagens
  - Melhor escalabilidade para projetos grandes.
- Desvantagens

Ditador e Tenentes – discussão

- Uma variação recursiva de "Gerente de integração".
- Praticamente impossível sem um sistema de controle de versão distribuído.
- Vantagens
  - Melhor escalabilidade para projetos grandes.
  - Menos pontos de falha
- Desvantagens

Ditador e Tenentes – discussão

- Uma variação recursiva de "Gerente de integração".
- Praticamente impossível sem um sistema de controle de versão distribuído.
- Vantagens
  - Melhor escalabilidade para projetos grandes.
  - Menos pontos de falha
- Desvantagens
  - ...

Roteiro

Sistemas de controle de versão

- É fundamental ter o histórico do desenvolvimento de um projeto.
- Uso conjunto com outras ferramentas pode fornecer ainda mais informação útil:
  - integração com ferramentas de gestão de projetos (*bug trackers*)
  - ferramentas de integração contínua

Workflows em projetos de software

Escolha depende de vários fatores

Workflows em projetos de software

Escolha depende de vários fatores

- Cultura e contexto da organização.

Workflows em projetos de software

Escolha depende de vários fatores

- Cultura e contexto da organização.
- Recursos.

Workflows em projetos de software

Escolha depende de vários fatores

- Cultura e contexto da organização.
- Recursos.
- Demandas dos usuários.

Workflows em projetos de software

Escolha depende de vários fatores

- Cultura e contexto da organização.
- Recursos.
- Demandas dos usuários.
- Demandas dos pares.

Para saber mais: git

- `http://www.git-scm.org/`: manuais, tutoriais
- GitCasts: `http://www.gitcasts.com/`
- `git COMMMAND --help`
  ou
  `man git-COMMAND`

Hospedagem de repositórios git

- Projetos de software livre:
  - `http://gitorius.org/`
  - `http://repo.or.cz/`
  - `http://github.com/`
- Projetos privados: `http://github.com/`
  - planos por número de repositórios, número de colaboradores, espaço disponível etc.

Discussão

Contato: terceiro@dcc.ufba.br