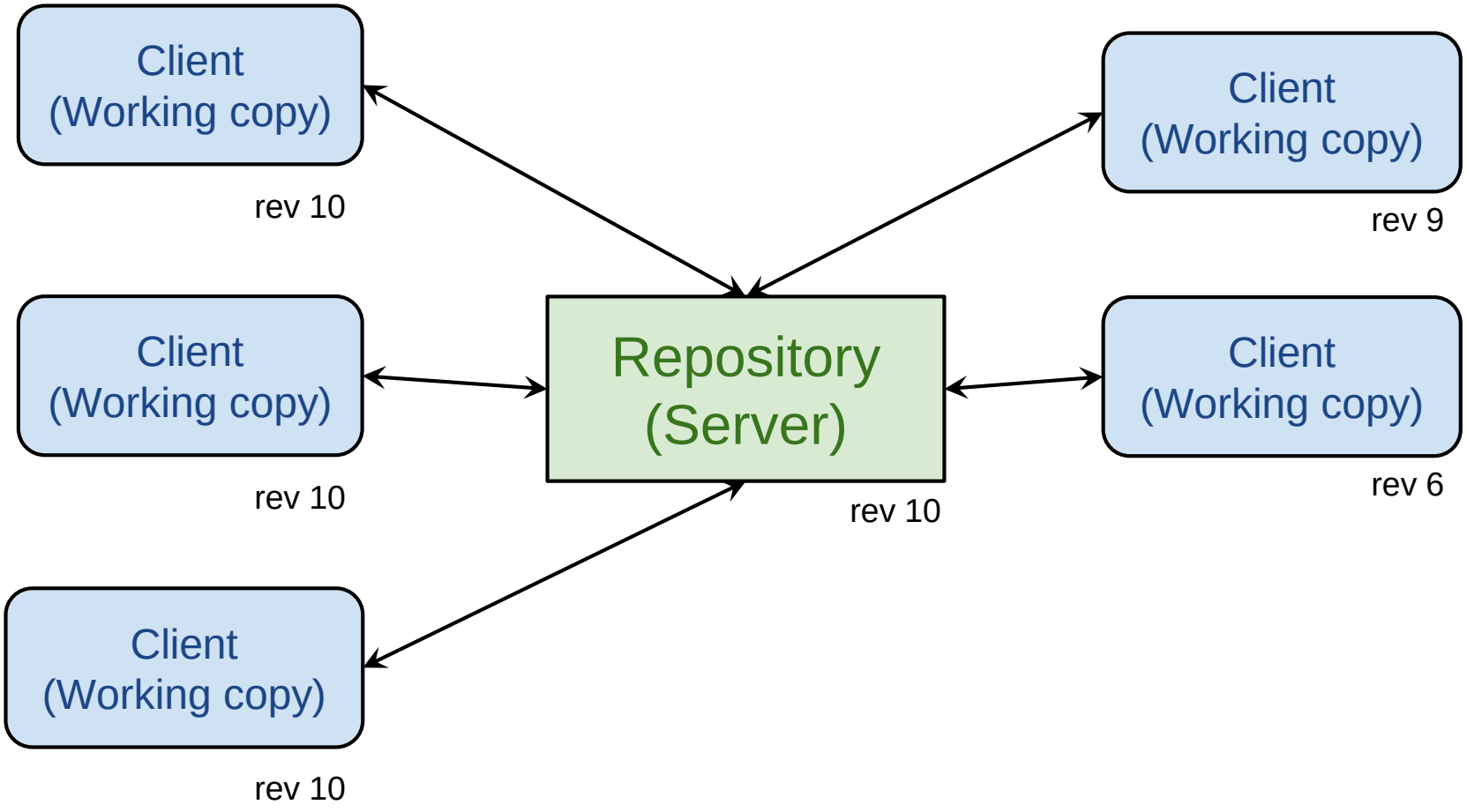
Управление производственным процессом разработки программного обеспечения

Распределенные системы контроля версий

Centralized VCS



Достоинства и недостатки централизации

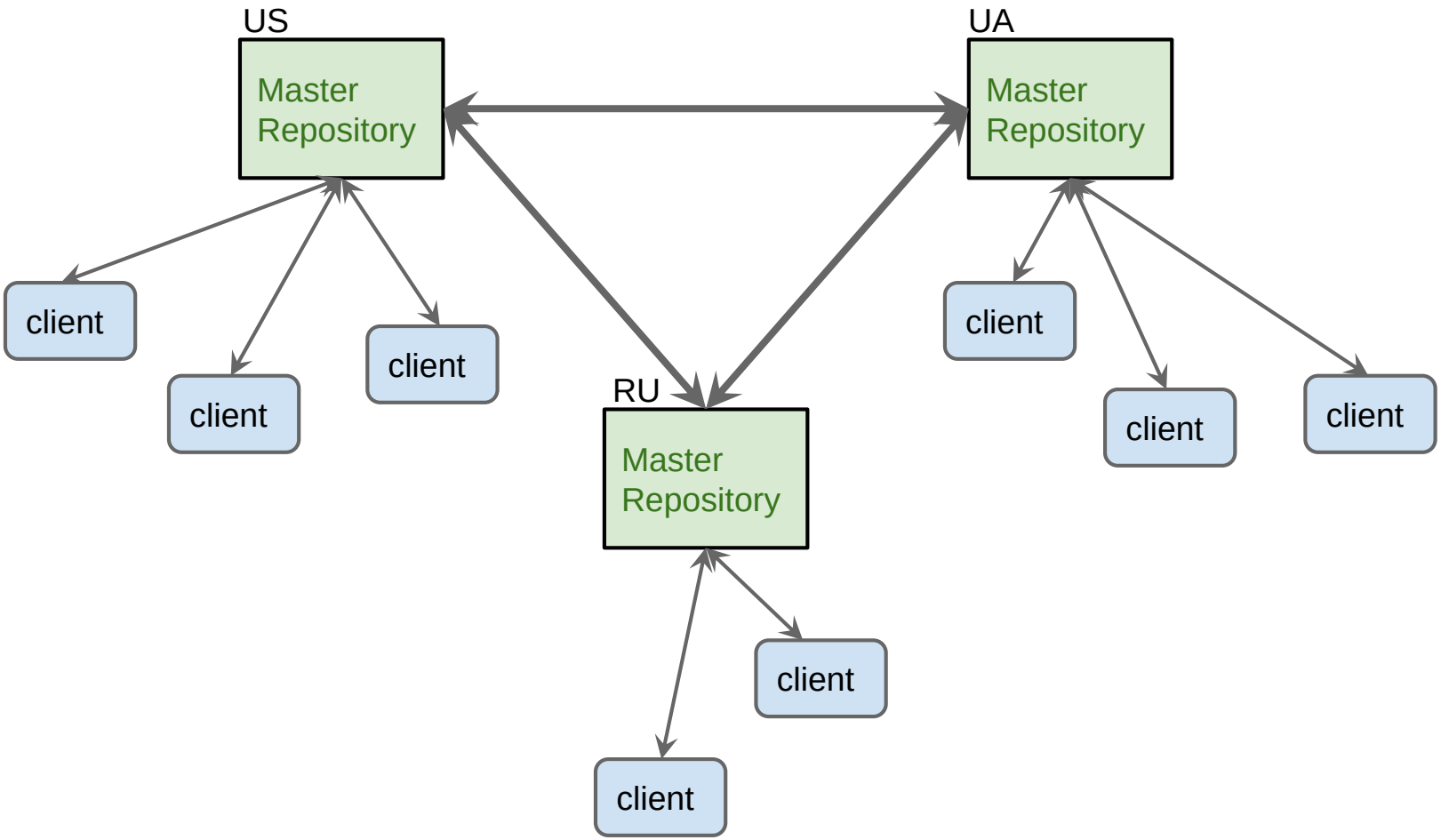
Достоинства:

- Всегда есть выделенная "эталонная" версия кода
- Централизованное администрирование
- Привычный workflow

Недостатки:

- Единая точка отказа — сервер
- Любые изменения влияют на всех пользователей
- Требуется достаточно гибкая система прав
- В очень больших или распределенных проектах централизованные механизмы работают плохо.
- Репозитории уровня корпорации трудно администрировать

Master-master репликация



Master-master репликация

Появилась довольно давно как надстройка над централизованными системами контроля версий (DCVS, SVK).

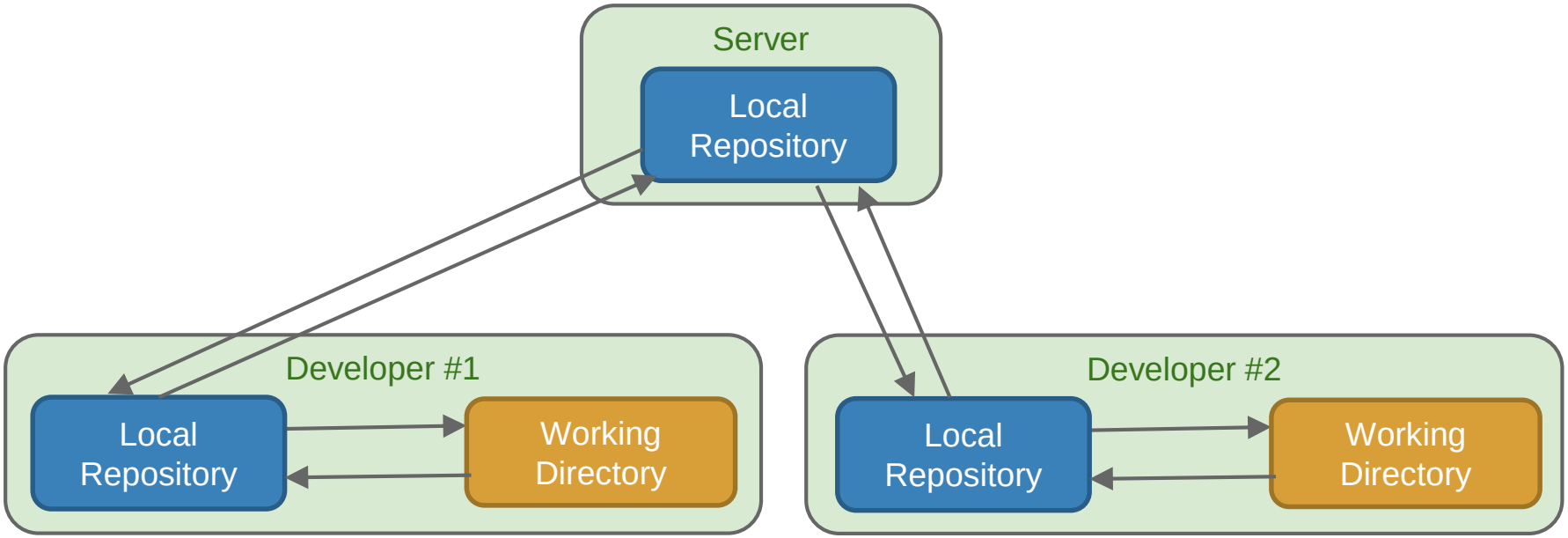
Работает, но не решает всех проблем:

- довольно сложно в администрировании
- все еще возможны конфликты правок
- все еще требуется связь с локальным сервером

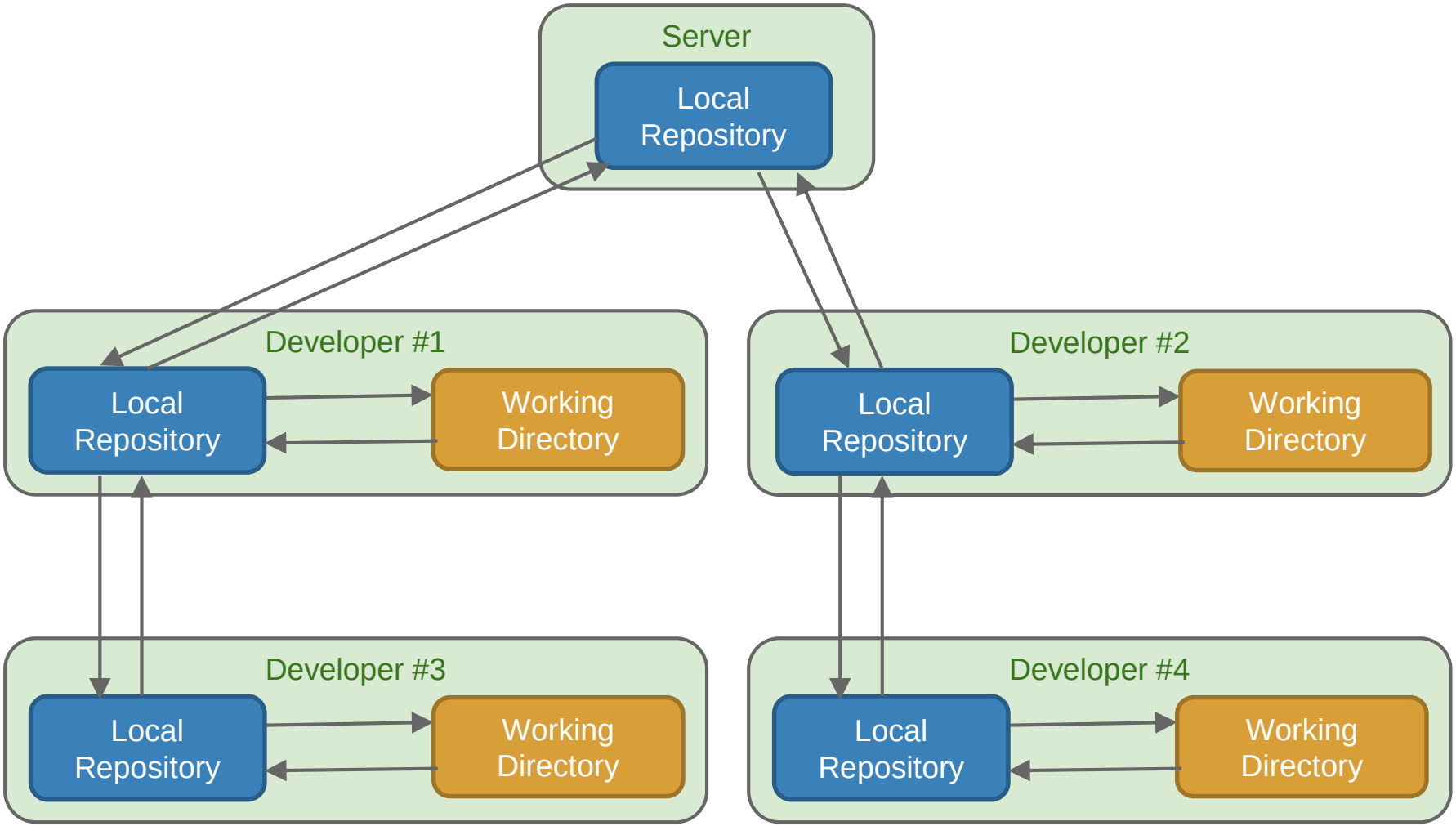
Мотивация проекта Linux kernel

- Очень большое число участников
- Сложности с управлением правами
- Собственные репозитории у участников проекта (таких как RedHat или Alt Linux)
- Очень большой объем исходного кода
- Сборка артефактов отделена от разработки

Distributed workflow

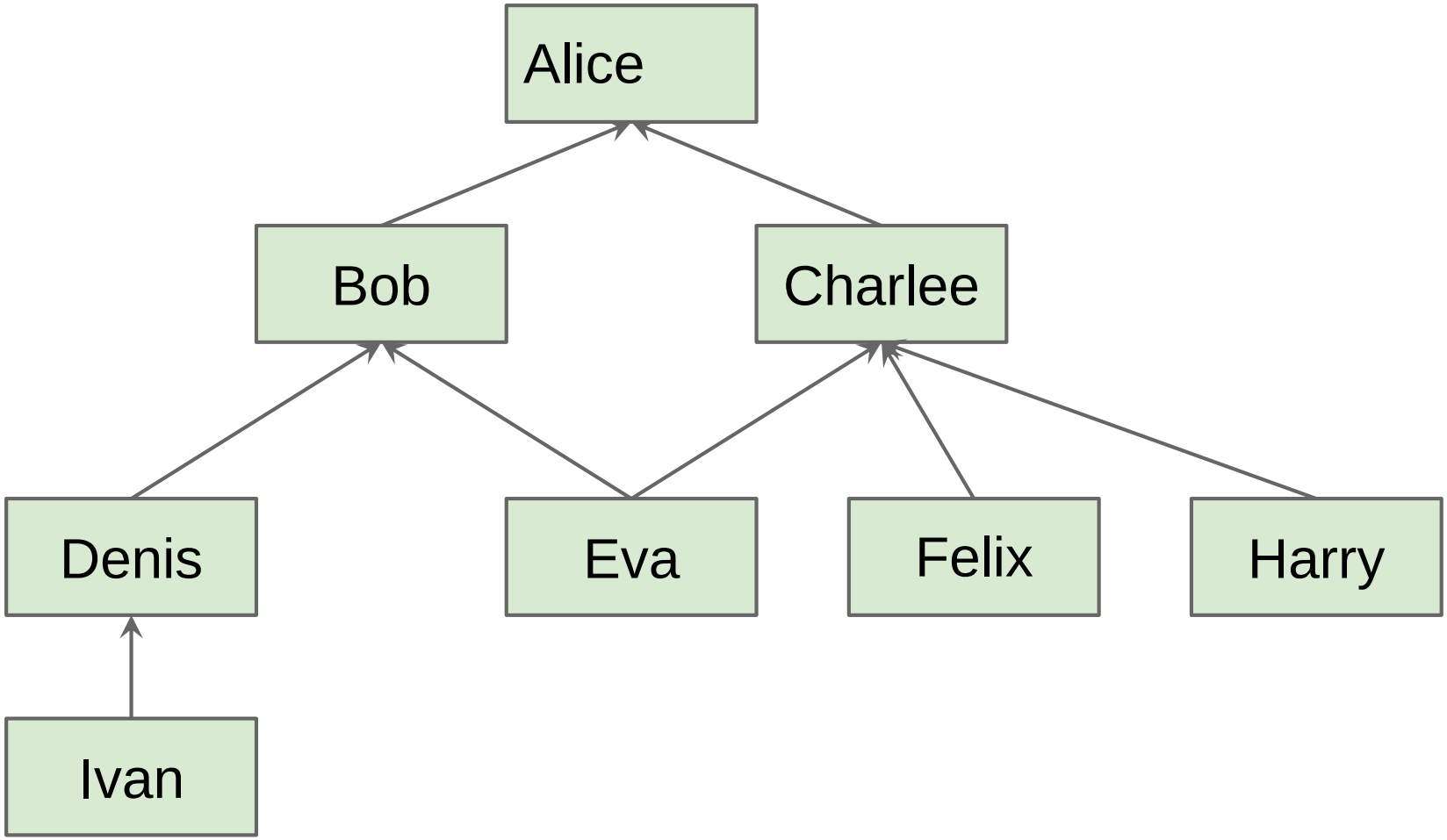


Distributed workflow

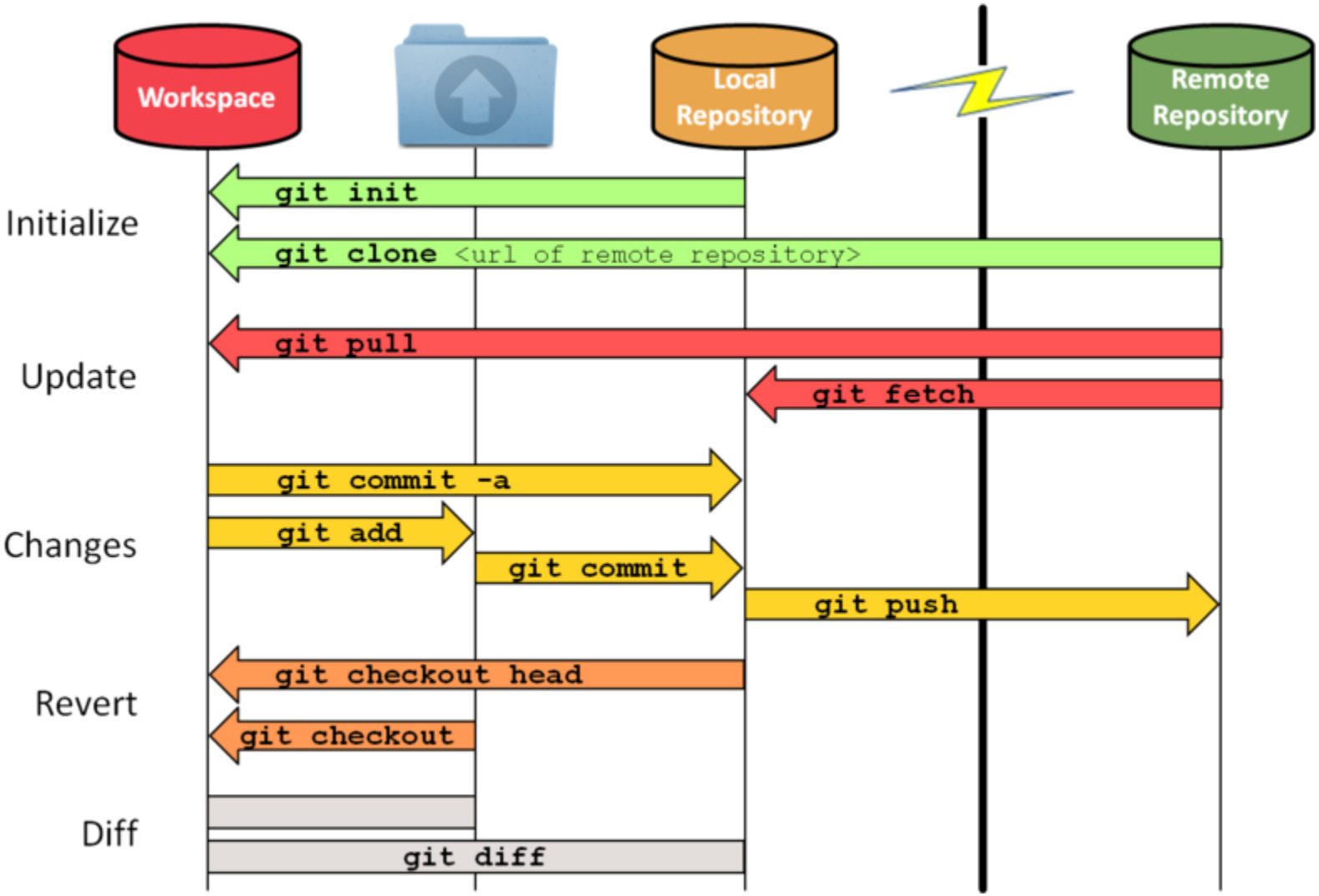


Сеть доверия

Trust network



Жизненный цикл git



Работа с индексом (stage)

Команда **add** добавляет измененные файлы в stage.

```
$git add filename
```

Команда **rm** помечает файл в stage как удаленный.

```
$git rm filename
```

Команда **reset** сбрасывает текущий stage.

```
$git reset HEAD filename
```

Работа с локальным репозиторием

Команда **commit** сохраняет текущий stage в локальный репозиторий.

```
$git commit
$git commit --amend
```

Команда **branch** создает/удаляет новую ветку.

```
$git branch
* master
$git branch test
$git branch
* master
  test
```

Команда **checkout** переключает рабочую копию на другую ветку.

```
$git checkout test
Switched to branch 'test'
```

Работа с удаленным репозиторием

Команда **clone** клонирует репозиторий и создает рабочую копию.

```
$git clone git@github.com:kothic/kothic-js.git
```

Cloning into kothic-js...
remote: Counting objects: 1771, done.
remote: Compressing objects: 100% (993/993), done.
remote: Total 1771 (delta 840), reused 1689 (delta 759)
Receiving objects: 100% (1771/1771), 6.36 MiB | 492 KiB/s, done.
Resolving deltas: 100% (840/840), done.

Команда **push** отправляет изменения в удаленный репозиторий.

```
$git push origin master
```

Команда **pull** забирает изменения указанной ветки из удаленного репозитория и сливает их в текущую ветку.

```
$git pull origin master
```

Команда **fetch** забирает все изменения из удаленного репозитория.

```
$git fetch
```

Временное хранилище (stash)

Команда **stash** помещает изменения из stage во временное хранилище и сбрасывает рабочую копию.

```
$git stash
Saved working directory and index state WIP on test:
7376ecb Performance test
HEAD is now at 7376ecb Performance test

$git status
Saved working directory and index state WIP on test:
7376ecb Performance test
HEAD is now at 7376ecb Performance test

$git stash apply
# On branch test
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   INSTALL
#
```

git: что внутри

- Контентно-адресуемое key-value хранилище
- В качестве ключа используется SHA-1 хэш от содержимого
- Указатели на выделенные точки в репозитории

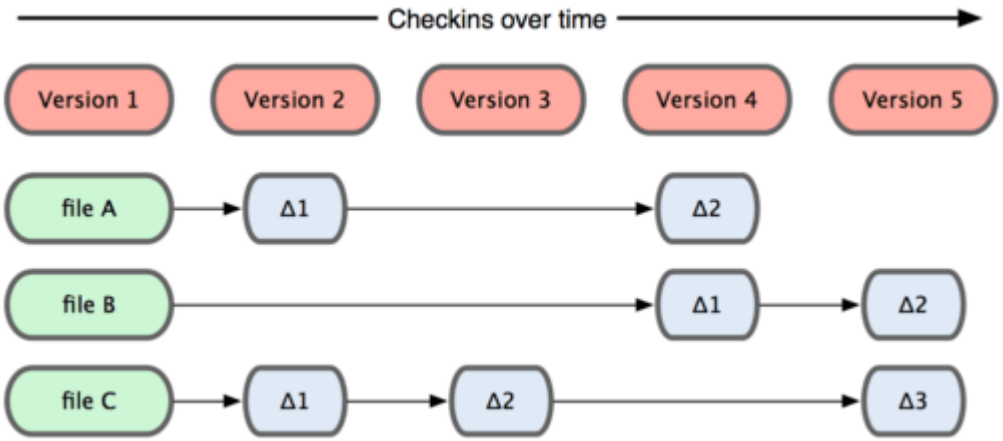
Identity management

Поскольку в распределенной среде нет центрального сервера который может назначать уникальные номера ревизий, в качестве идентификаторов ревизий используются хэши.

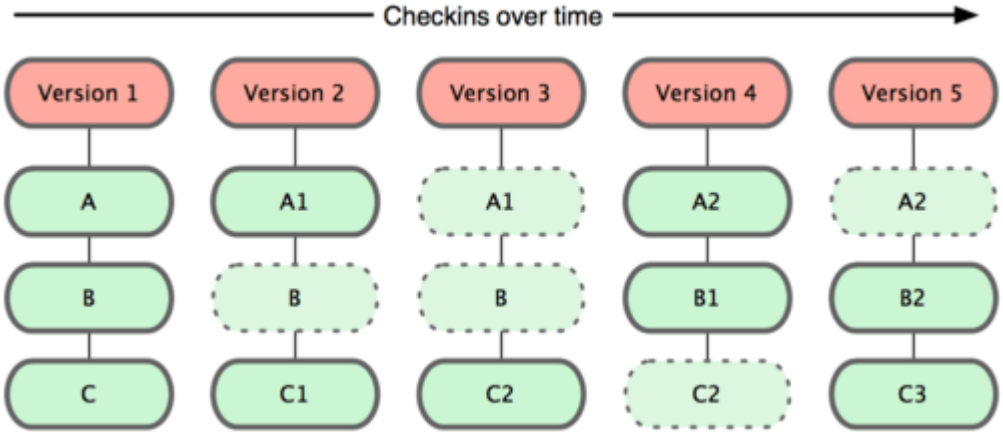
```
$git log --pretty="%h - %s"
7376ecb - Performance test
e8d3877 - Added style from openstreetmap.org Added mapcss
target to Makefile
2406a45 - Merge branch 'master' of
github.com:kothic/kothic-js
478ef8e - recompile
d30b6e8 - faster thin lines rendering
c295b70 - Added .gitignore
4956337 - Checked style with jslint
```

Хранение данных

SVN



git

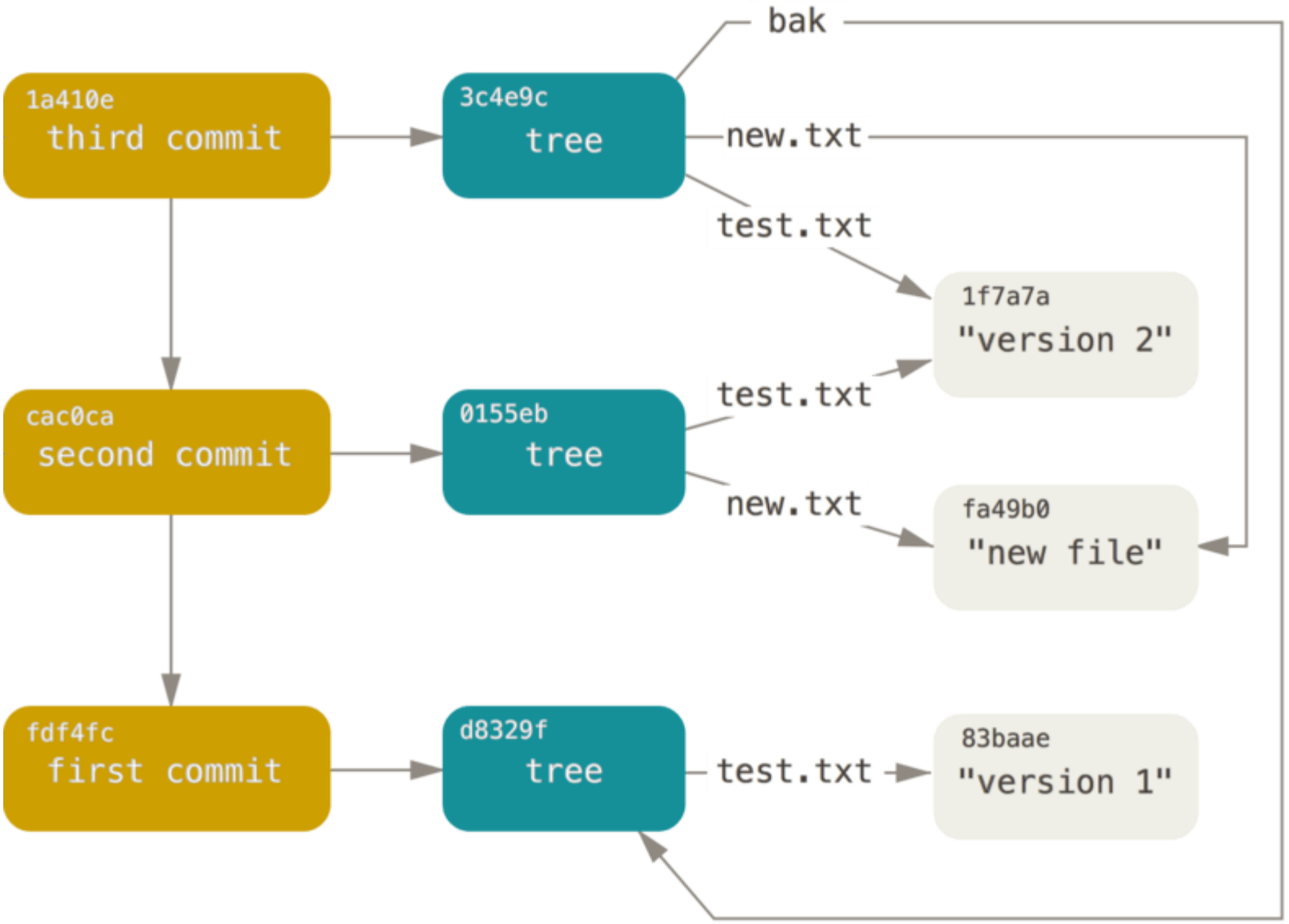


Объекты

- blob (содержимое файлов)
- tree (папки и имена файлов со ссылками на соответствующие объекты)
- commit (tree + метаданные + родительские коммиты)

Ветки это просто указатели на определенные коммиты

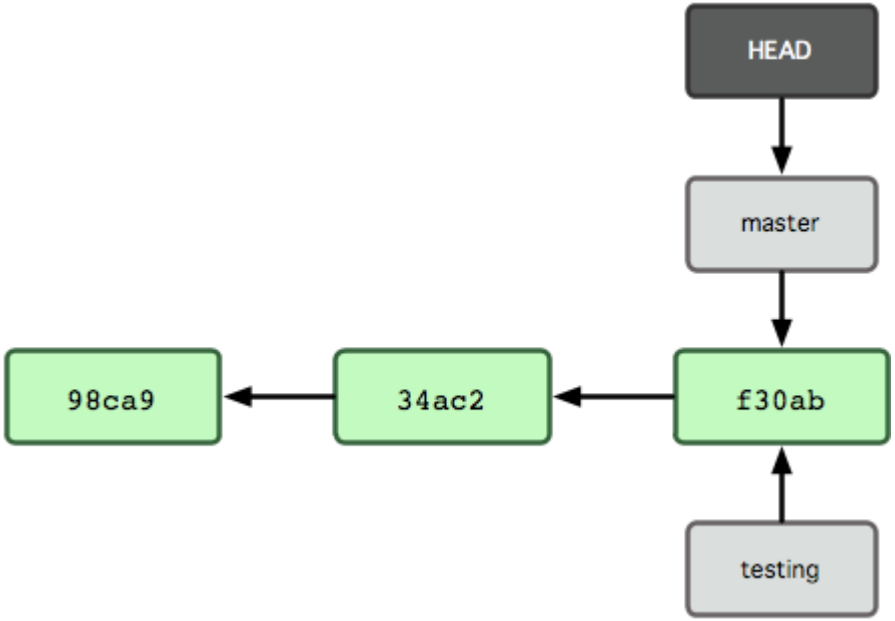
Хранение информации в git



Ветвление

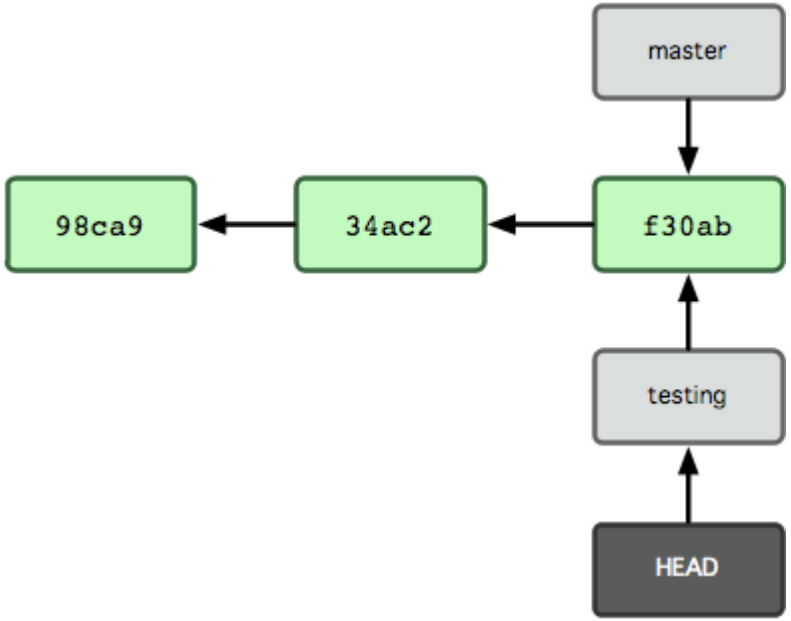
git clone http://gitlab.dev.ccfit.nsu.ru/students2016/example.git

git branch testing



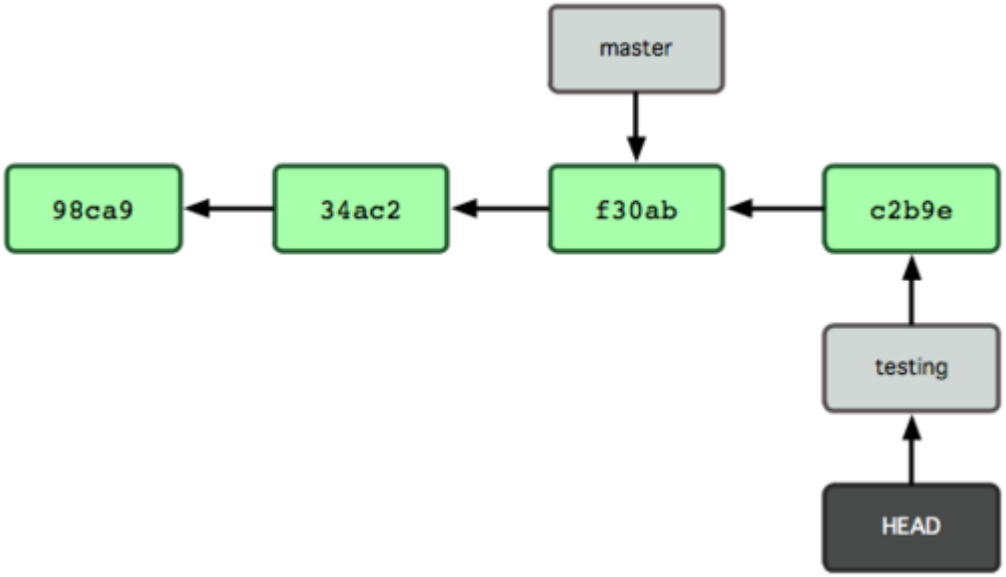
Ветвление

git checkout testing



Ветвление

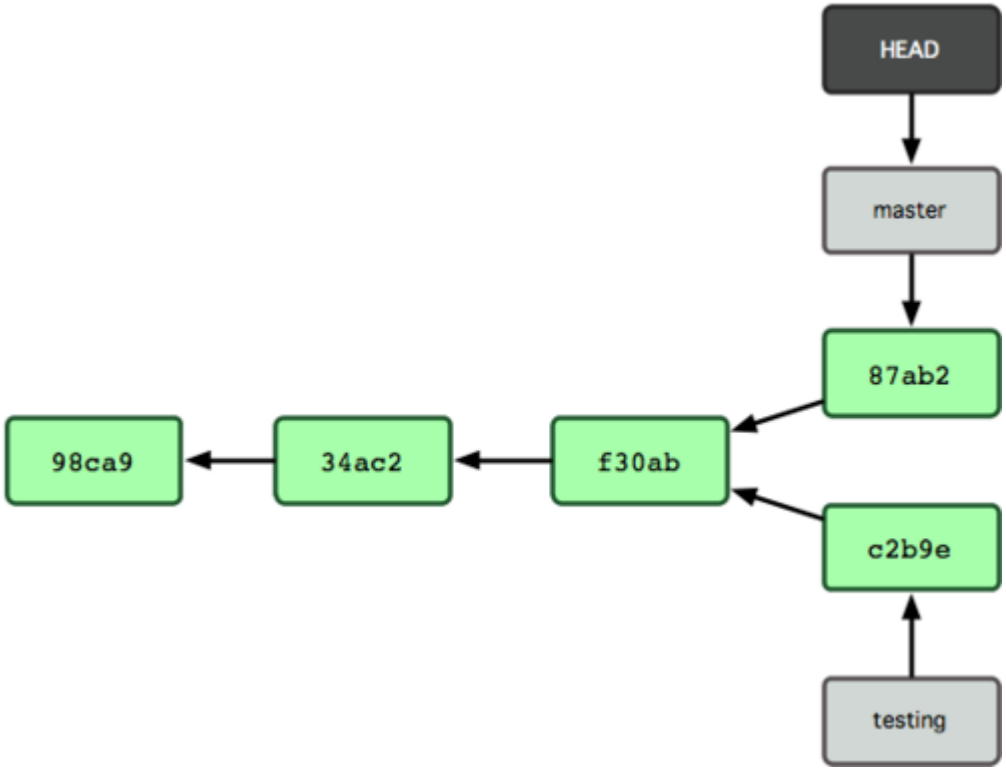
git commit –a –m ‘change in the file’



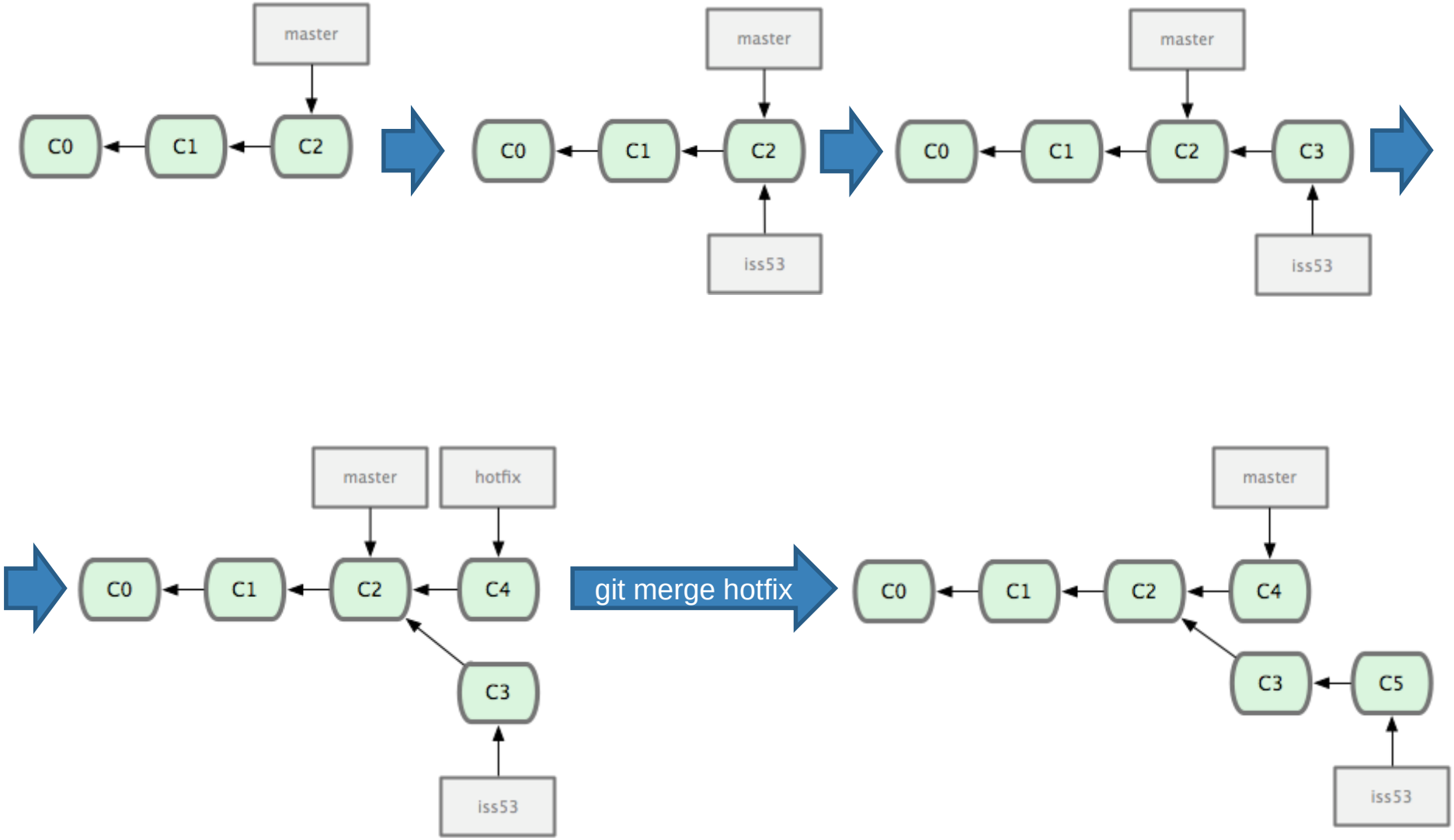
Ветвление

git checkout master

git commit –a –m ‘another change in the file’

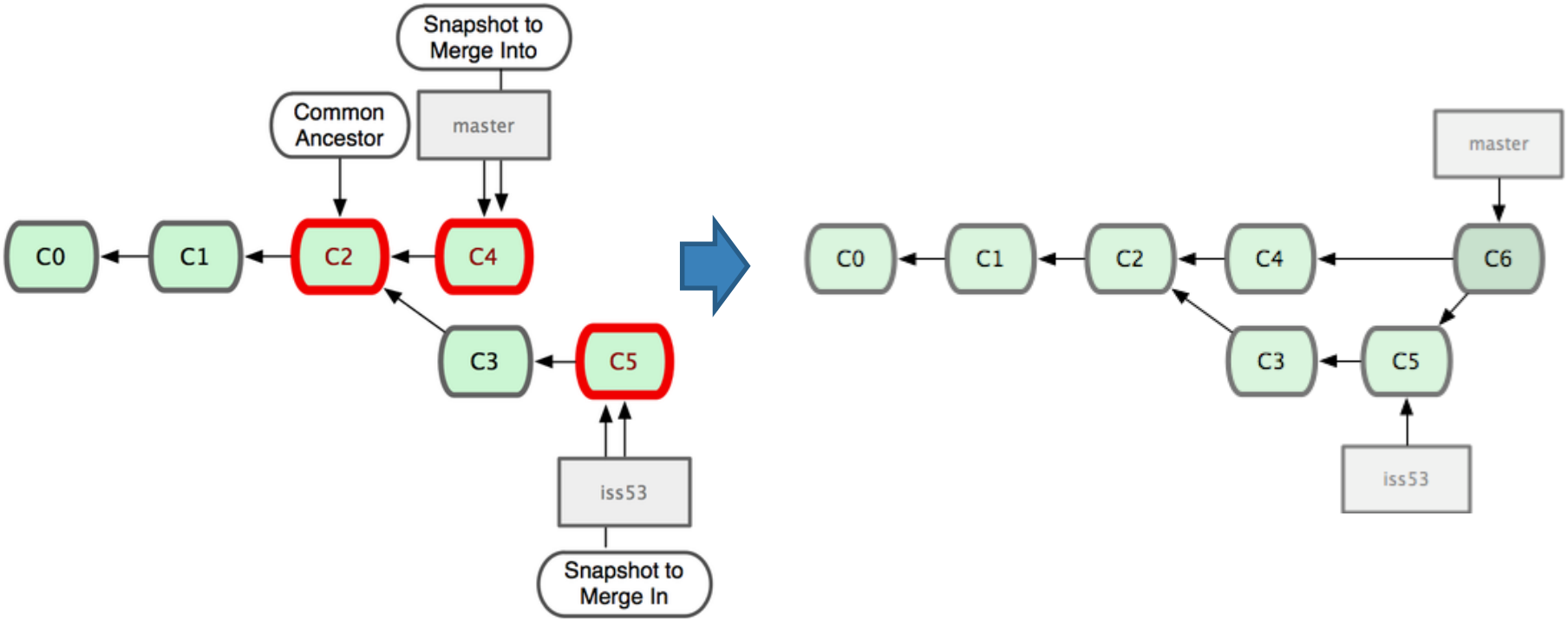


Слияние веток

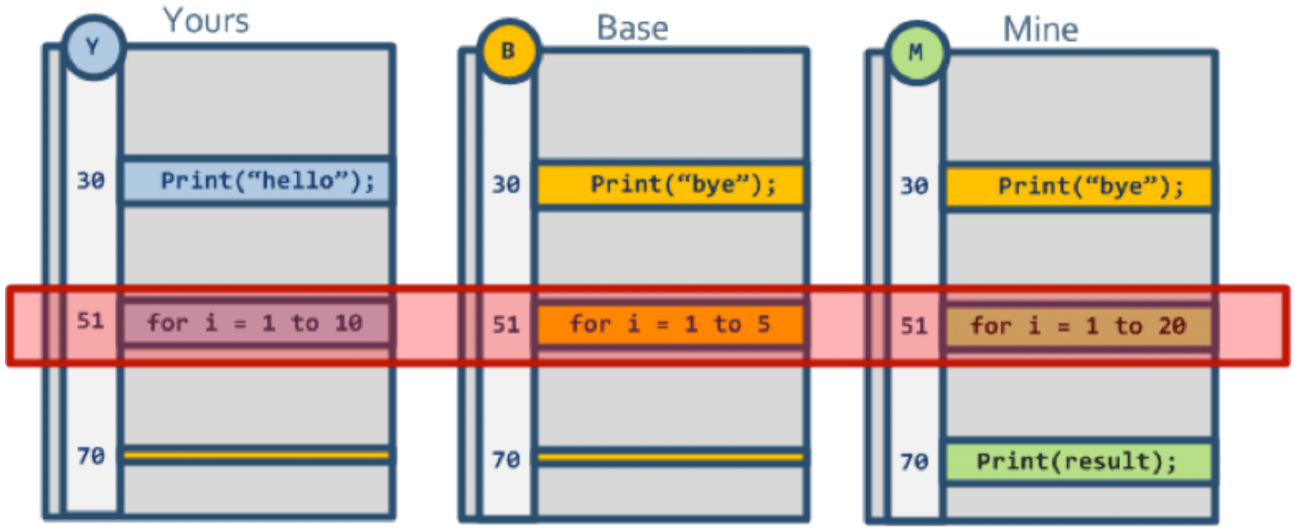


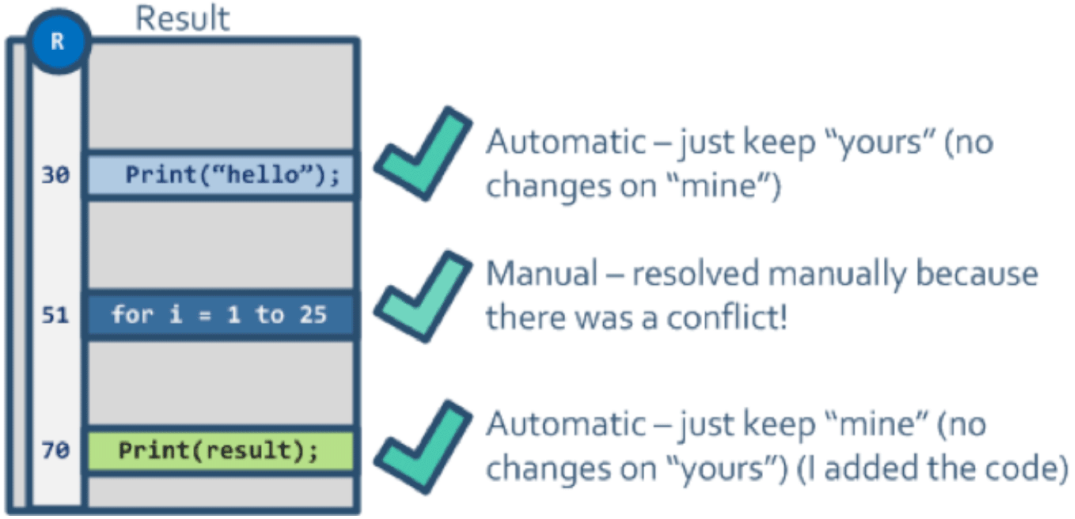
Слияние веток

git merge iss53

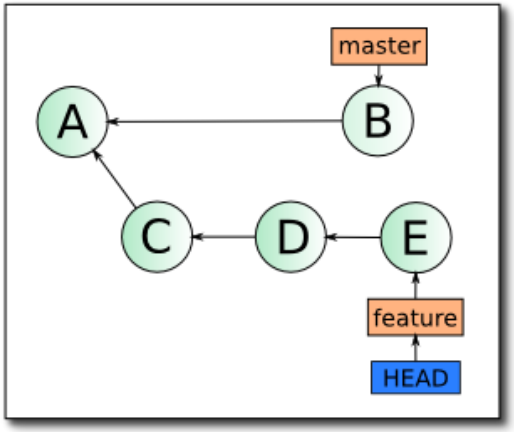


3-way merge

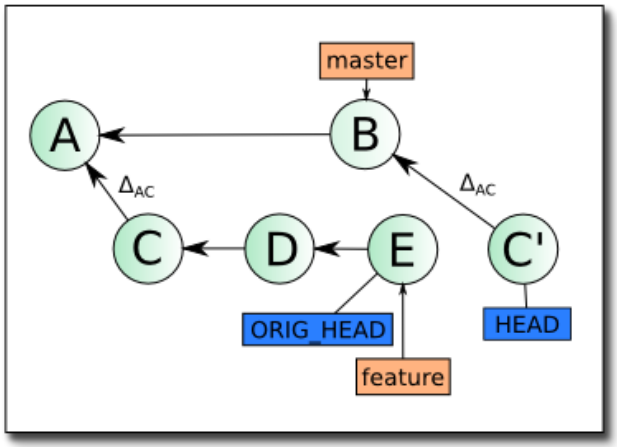

Yours
Y
30
Print("hello");
51
for i = 1 to 10
70
Base
B
30
Print("bye");
51
for i = 1 to 5
70
Mine
M
30
Print("bye");
51
for i = 1 to 20
70
Print(result);


Result
R
30
Print("hello");
51
for i = 1 to 25
70
Print(result);
Automatic – just keep "yours" (no changes on "mine")
Manual – resolved manually because there was a conflict!
Automatic – just keep "mine" (no changes on "yours") (I added the code)

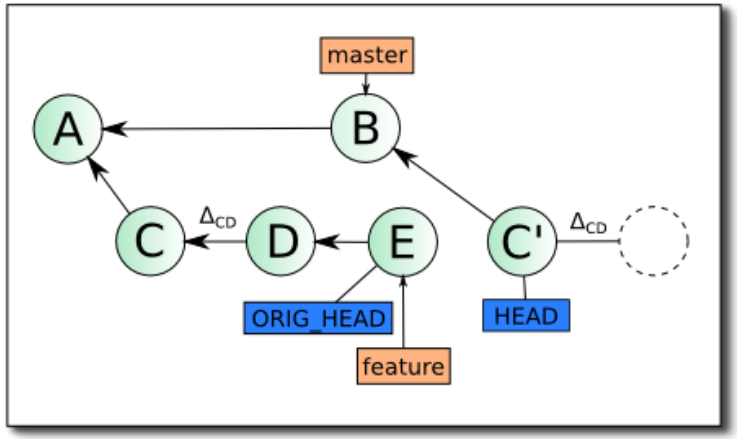
Rebase

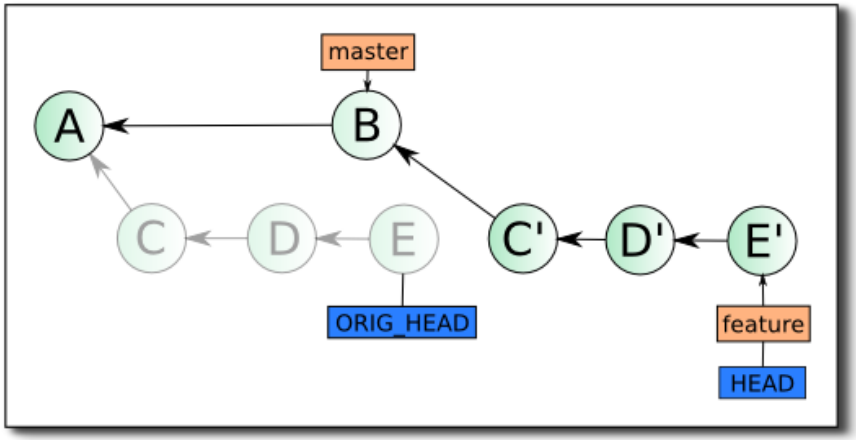


git checkout feature
git rebase master

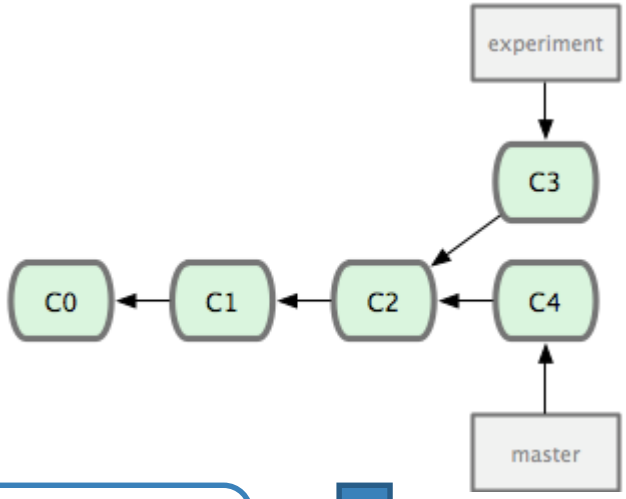


- git rebase --abort
- git rebase --continue
- git rebase --skip

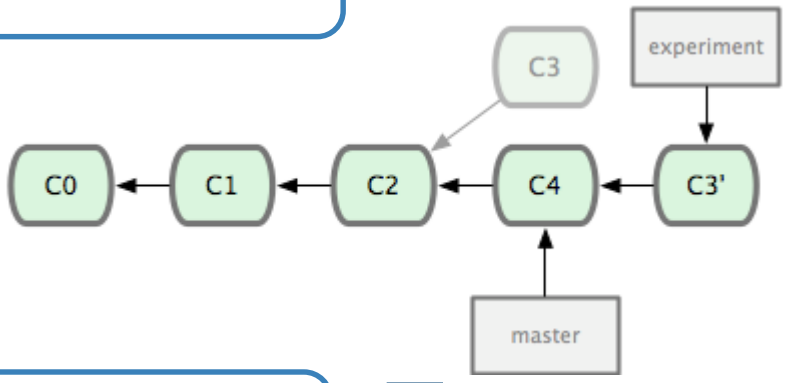




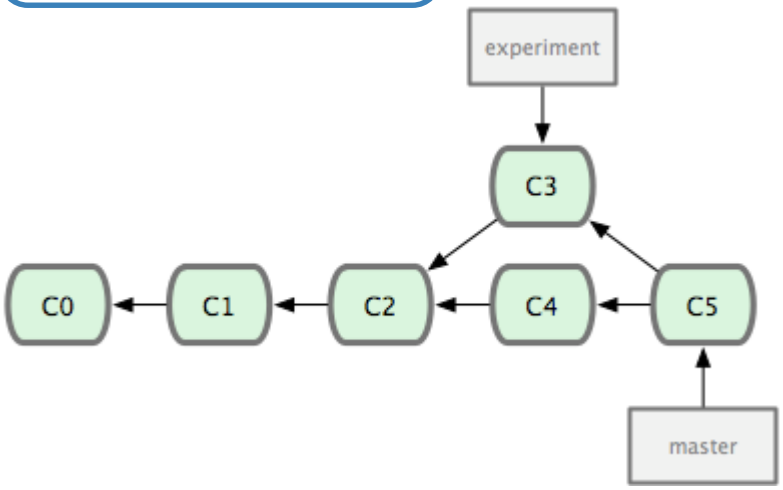
Merge vs Rebase



git checkout experiment
git rebase master

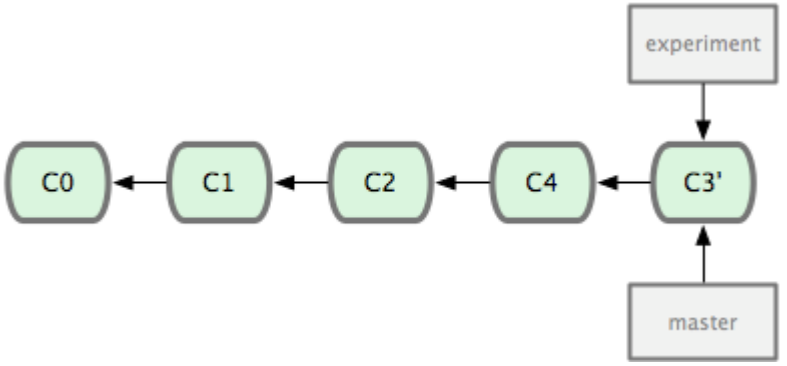


git checkout master
git merge experiment

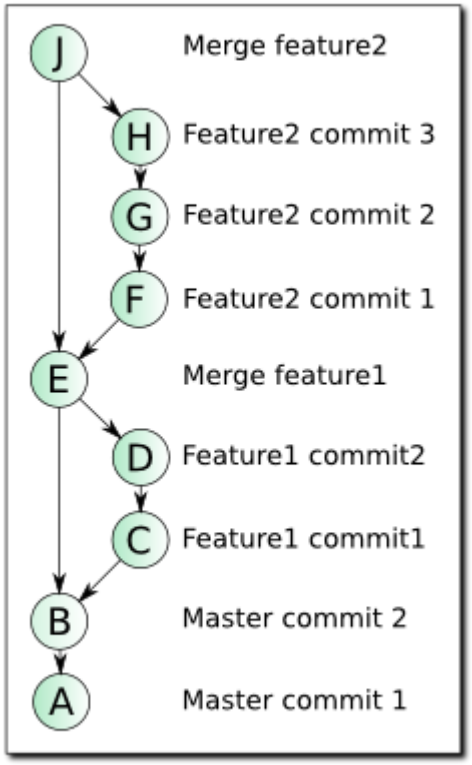


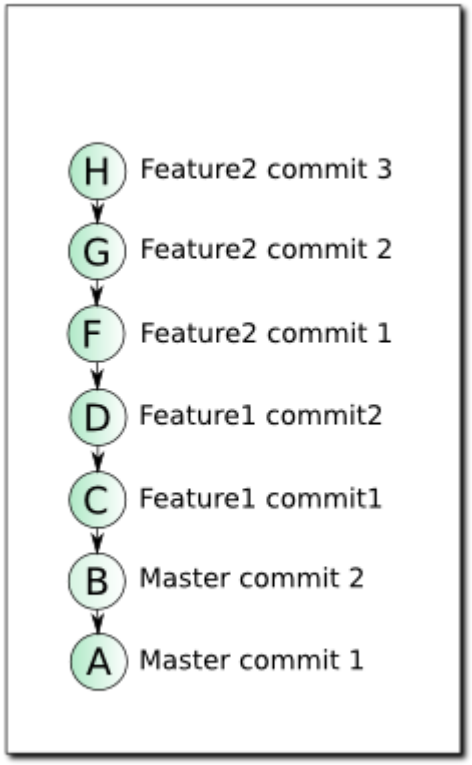
git checkout master
git merge experiment

Fast-forward

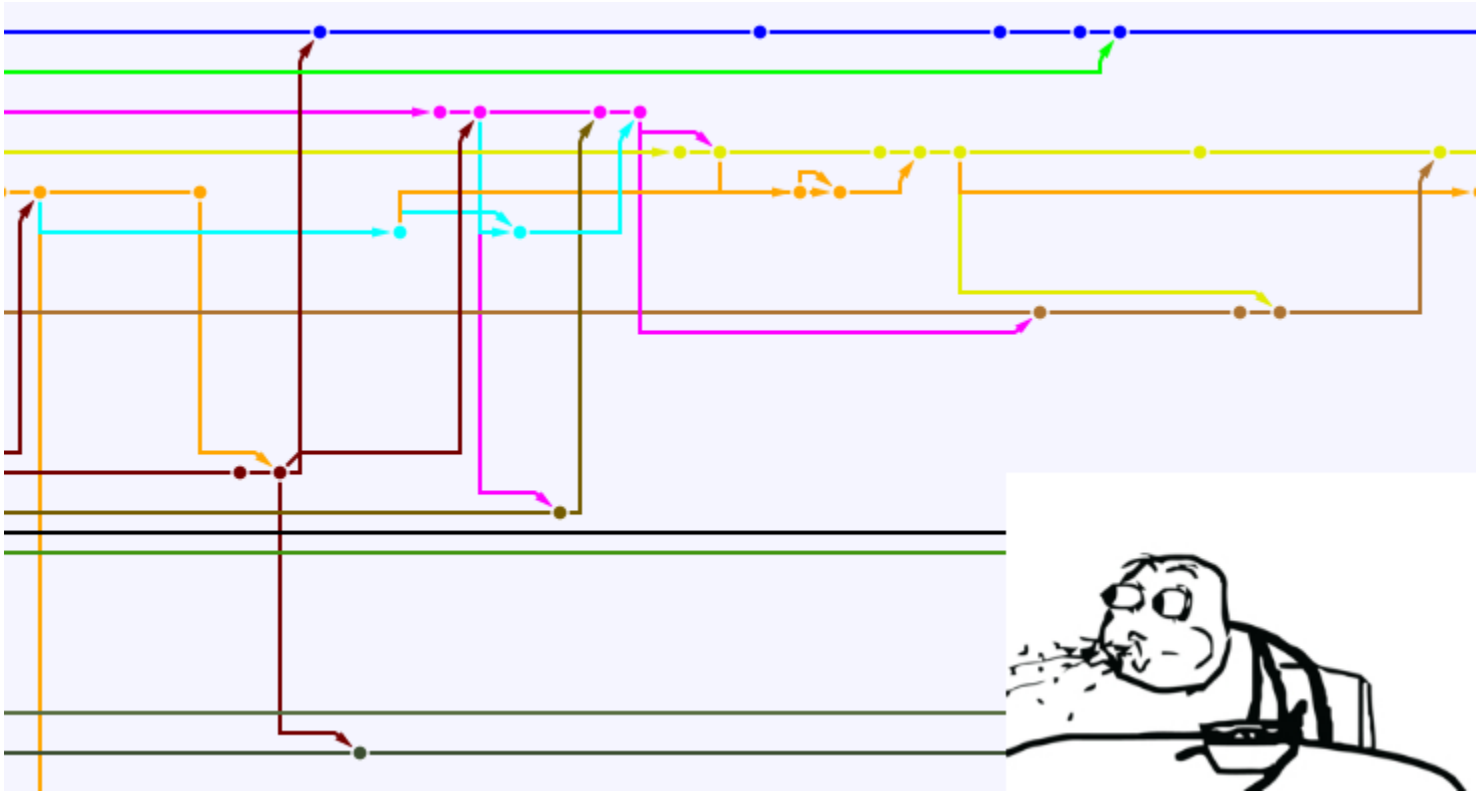


Merge vs Rebase





Merge vs Rebase



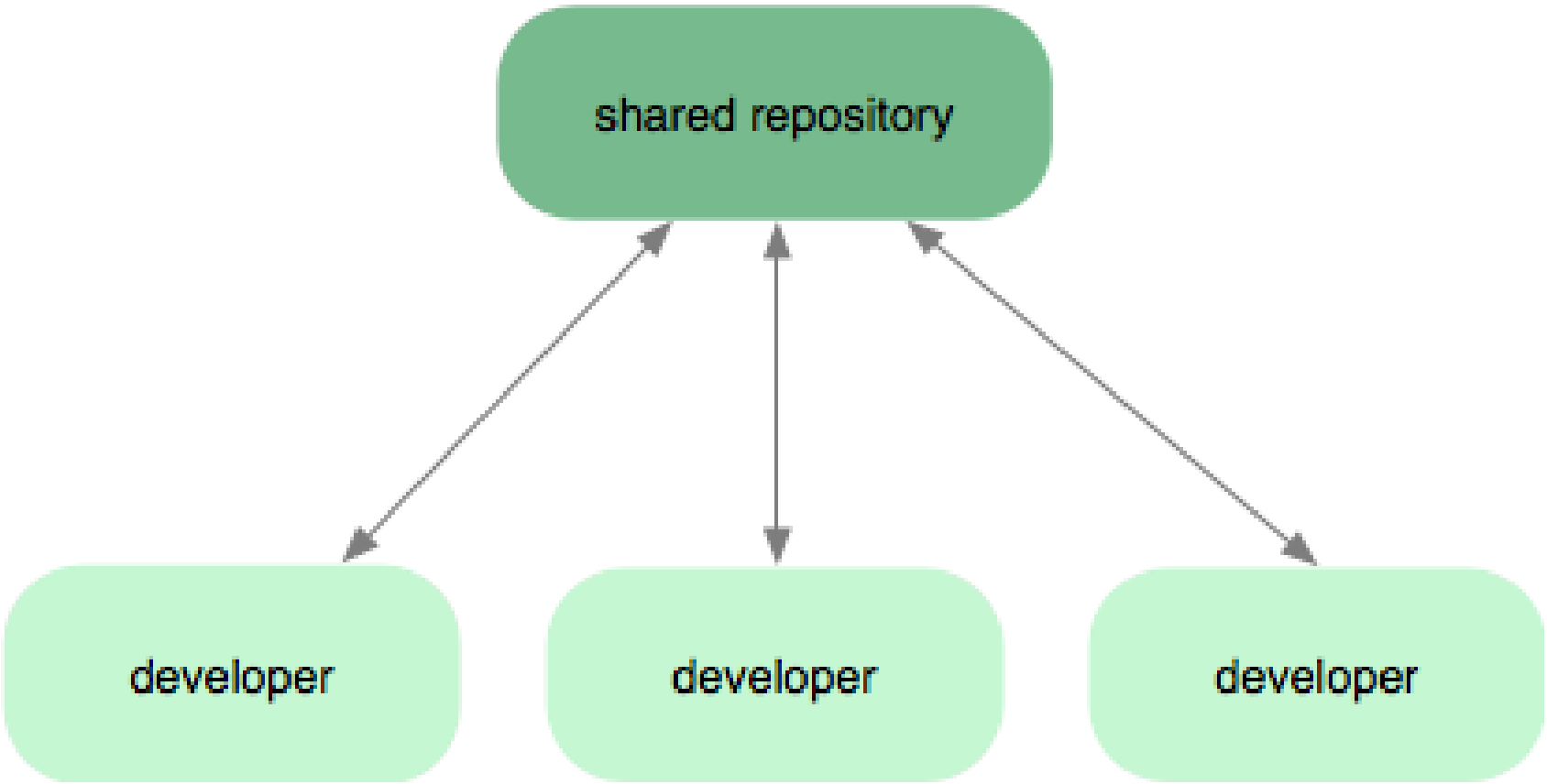
Merge vs Rebase

Rebase - возможность почистить историю приватных веток от запутанных слияний и множества мелких коммитов

Rebase Policy – опасность работы с публичными ветками, другие разработчики вынуждены будут проходить цепочку новых коммитов каждый раз

Merge Policy – простота использования, запутанность истории коммитов

Subversion-style workflow

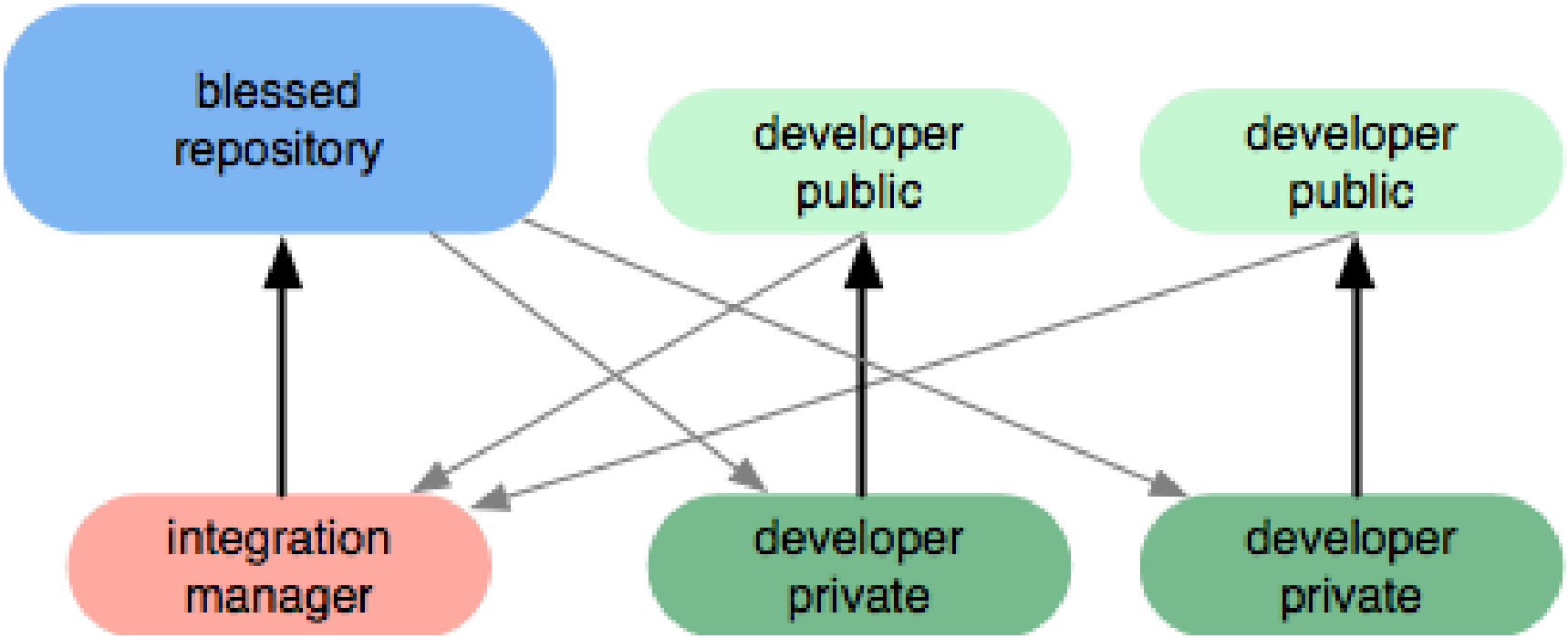


Git workflow

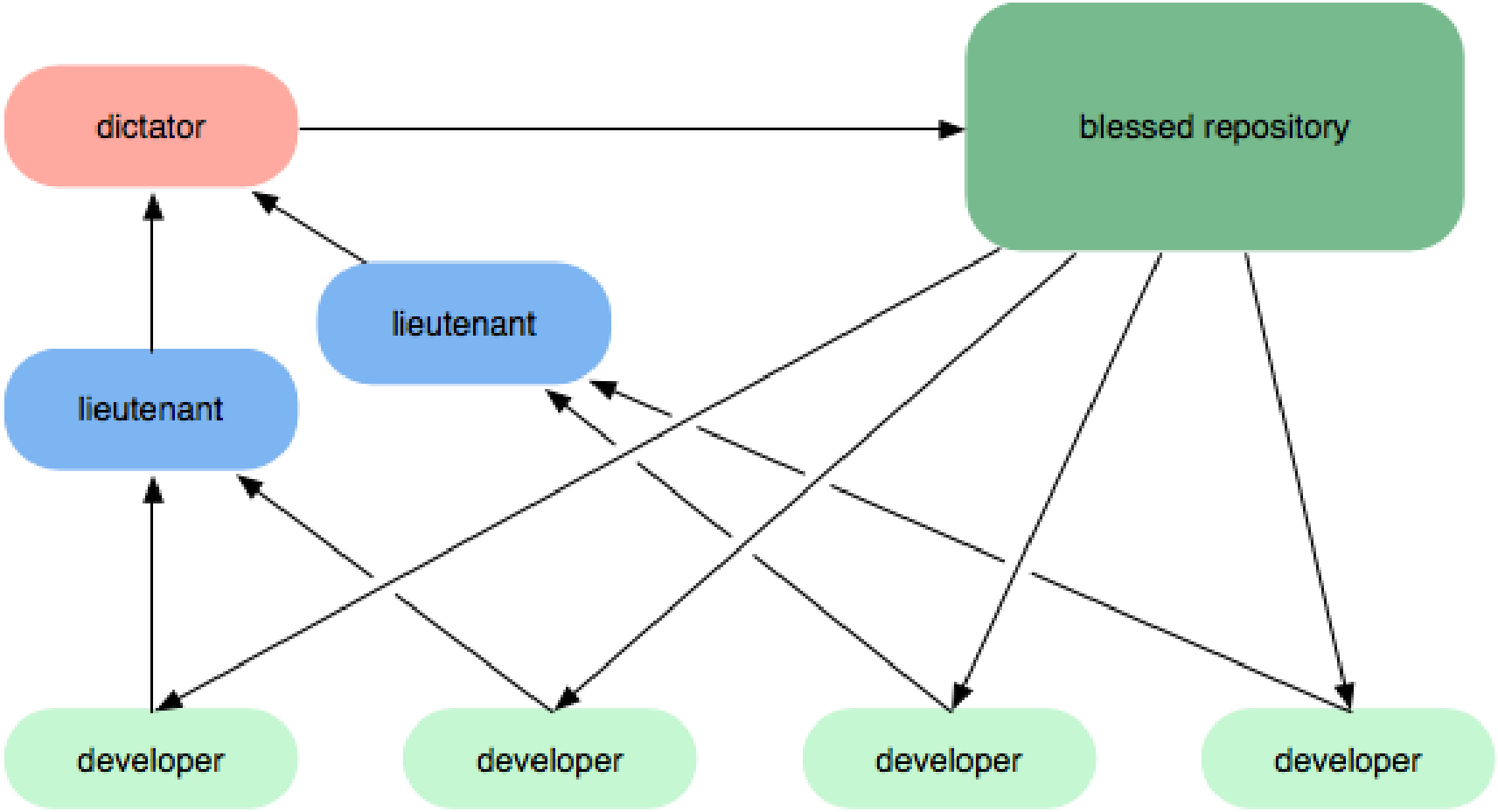
Поскольку git умеет изменять локальную историю, широко распространен такой workflow:

1. При разработке изменения коммитятся очень часто и довольно грязно.
2. Как только разработка завершена делается rebase к master
3. Все коммиты сбрасываются и изменения остаются только в рабочей копии
4. С помощью команд **add -i** и **commit** изменения группируются по смыслу
5. Уже причесанные изменения залиаются в удаленный репозиторий.

Integration manager workflow



Dictator and Lieutenants workflow



Достоинства DVCS

- Нет выделенного центра (технически)
- Вся история дублируется в каждом локальном репозитории
- Нет необходимости в постоянном соединении
- Нет необходимости в управлении правами
- Поддерживает любой workflow
- Каждый разработчик работает в своей песочнице
- Простое создание и слияние веток
- Локальные операции работают быстро
- Позволяет откладывать решение о коммите до момента push’а на сервер

Недостатки DVCS

- Все еще требуются бэкапы
- Нет сквозных номеров ревизий
- Достаточно сложны в использовании, по крайней мере git

Рекомендуемая литература

- http://git-scm.com/book
- Mercurial: The Definitive Guide
- Why Git is Better than X
- Why Switch to Bazaar?
- GitHub