

Системы управления версиями

Введение

Исторически – разработка ПО многими разработчиками происходит посредством листов рассылки – аналога форума.

Разработчики делятся своими наработками в формате патчей, которые затем интегрируются автором проекта

```
1 diff --git a/gcc/config/arc/arc-arches.def b/gcc/config/arc/arc-arches.def
2 index f24babb..5fd45cd 100644
3 --- a/gcc/config/arc/arc-arches.def
4 +++ b/gcc/config/arc/arc-arches.def
5 @@ -40,7 +40,7 @@
6 .
7 .ARC ARCH ("arcem", .em, .FL_MPYOPT_1_6 | .FL_DIVREM | .FL_CD | .FL_NORM |
8 . . . . . | .FL_BS | .FL_SWAP | .FL_FPUS | .FL_SPFP | .FL_DPFP . . . . . \
9 . . . . . | .FL_SIMD | .FL_FPUDA, .0)
10 + . . . . . | .FL_SIMD | .FL_FPUDA | .FL_QUARK, .0)
11 .ARC ARCH ("archs", .hs, .FL_MPYOPT_7_9 | .FL_DIVREM | .FL_NORM | .FL_CD |
12 . . . . . | .FL_ATOMIC | .FL_LL64 | .FL_BS | .FL_SWAP . . . . . \
13 . . . . . | .FL_FPUS | .FL_FPUD, . . . . . \
14 diff --git a/gcc/config/arc/arc-c.def b/gcc/config/arc/arc-c.def
15 index 4cfd7b6..fd64376 100644
16 --- a/gcc/config/arc/arc-c.def
17 +++ b/gcc/config/arc/arc-c.def
18 @@ -58,6 +58,7 @@
19 .ARC_C_DEF (" _ARC_FPU_DP_DIV ", .TARGET_FP_DP_SQRT)
20 .ARC_C_DEF (" _ARC_FPU_SP_FMA ", .TARGET_FP_SP_FUSED)
21 .ARC_C_DEF (" _ARC_FPU_DP_FMA ", .TARGET_FP_DP_FUSED)
22 .ARC_C_DEF (" _ARC_FPU_ASSIST ", .TARGET_FP_DP_AX)
23 + .ARC_C_DEF (" _ARC_FPX_QUARK ", .TARGET_FPX_QUARK)
```

Фрагмент патча для компилятора gcc

Введение

Ситуация, в которой электронный документ за время своего существования претерпевает ряд изменений, достаточно типична. При этом часто бывает важно иметь не только последнюю версию, но и несколько предыдущих. В простейшем случае можно просто хранить несколько вариантов документа, нумеруя их соответствующим образом. Такой способ неэффективен (приходится хранить несколько практически идентичных копий), требует повышенного внимания и дисциплины и часто ведёт к ошибкам, поэтому были разработаны средства для автоматизации этой работы.

Следует отметить, что большинство систем контроля версий работают с абстрактными документами-файлами. Конечно, наиболее удобно версионировать простые текстовые файлы, но большинство систем контроля версий неплохо справляется и с бинарными файлами, такими как изображения.

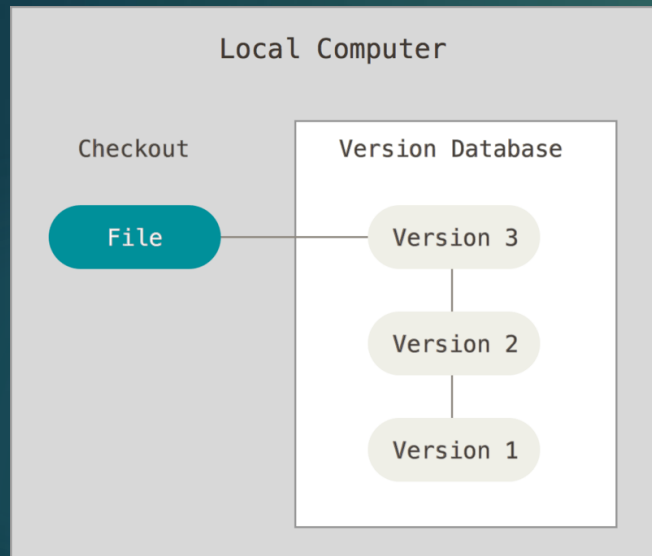
Задачи, решаемые системами контроля версий

- ▶ Позволяют хранить все состояния проекта на протяжении разработки
- ▶ Позволяют нескольким разработчикам комфортно работать над проектом одновременно
- ▶ Защищают проект от случайно (или не случайного) удаления
- ▶ Позволяют определить автора тех или иных изменений
- ▶ Позволяют параллельно разрабатывать несколько проектов с одной и той же кодовой базой (например: библиотека и несколько приложений основанных на ней)

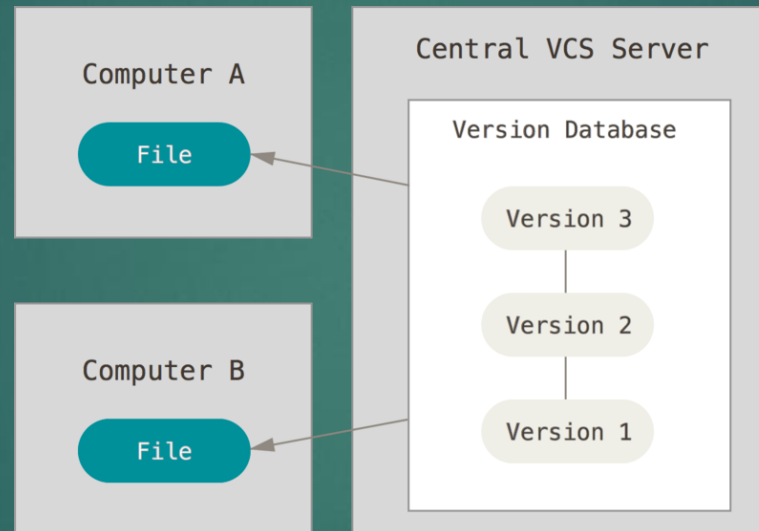
Классификация

Системы контроля версий можно классифицировать по принадлежности к нескольким категориям.

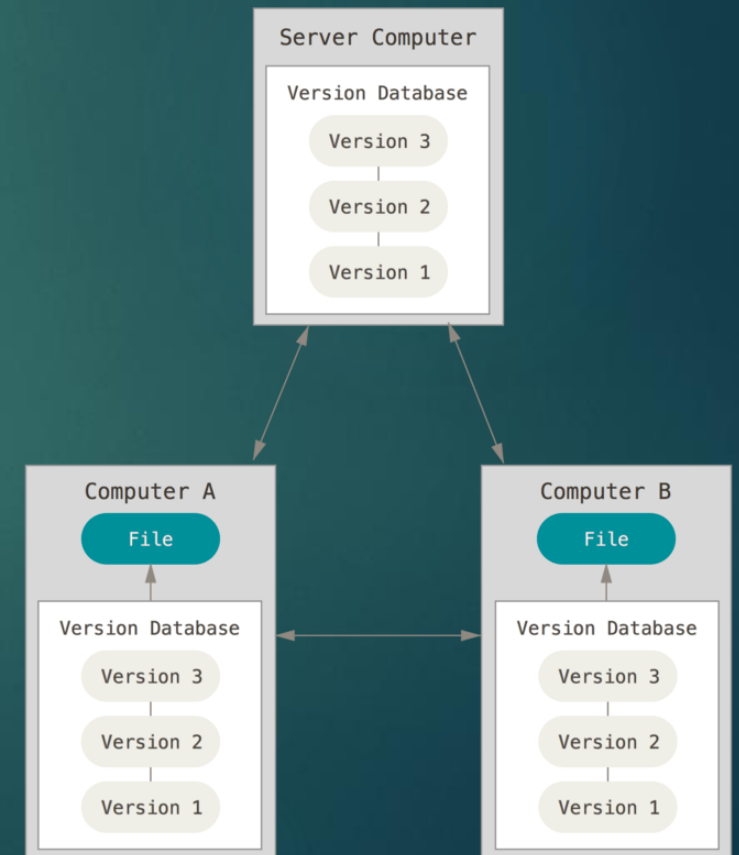
1) Локальные



2) Централизованные



3) Распределенные (децентрализованные)



Наиболее значимые VCS

GIT

Модель хранения данных: распределенная;

Год релиза: 2005;

Разработчики: Linus Torvalds, Junio Hamano и др.

Приоритеты разработчиков

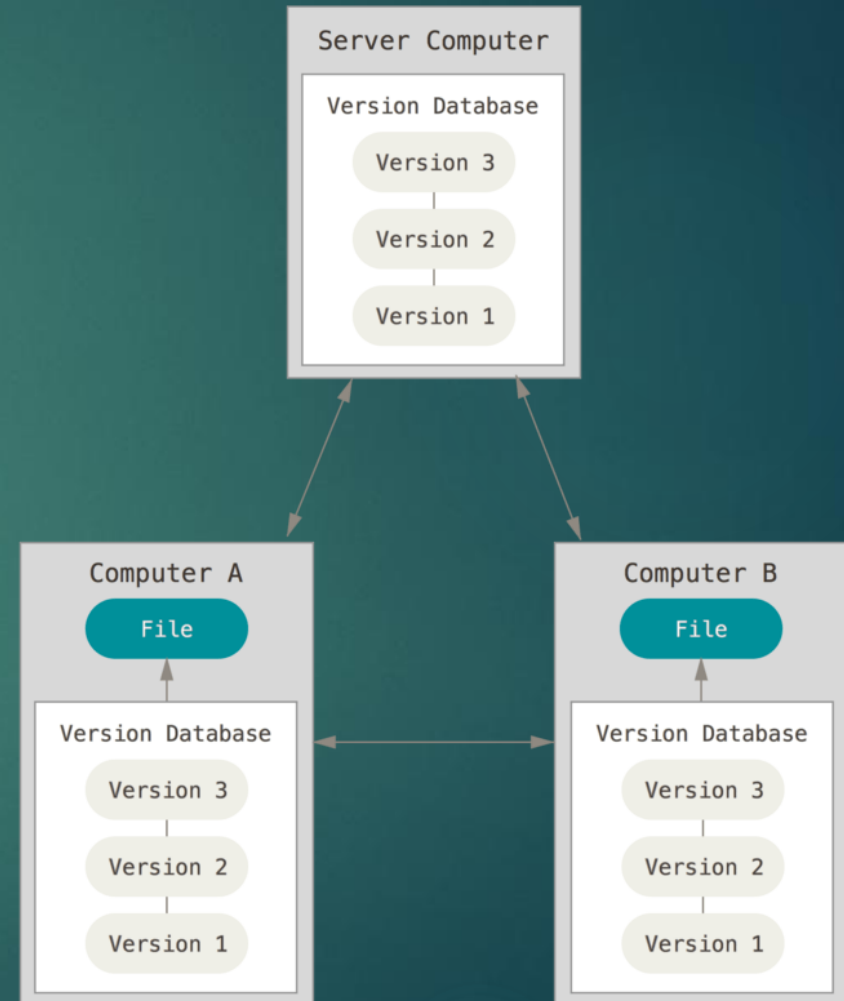
- ▶ Производительность
- ▶ Простая кодовая база самой VCS;
- ▶ Ориентированность на нелинейную разработку
- ▶ Децентрализованность
- ▶ Возможность использования в больших проектах (Linux Kernel)



Прочие системы контроля версий

- ▶ BitKeeper
- ▶ StarTeam
- ▶ AccuRev SCM
- ▶ ClearCase
- ▶ Code Co-op
- ▶ Codeville
- ▶ CVSNT
- ▶ Darcs
- ▶ Dimensions CM
- ▶ Endavor
- ▶ Fossil
- ▶ GNU arch
- ▶ IC Manage
- ▶ MKS Integrity
- ▶ Monotone
- ▶ Perforce
- ▶ Plastic SCM
- ▶ PVCS
- ▶ Rational Team Concert
- ▶ Revision Control System
- ▶ SCM Anywhere
- ▶ Source Code Control System
- ▶ Surround SCM
- ▶ MKS Integrity
- ▶ SVK
- ▶ Team Foundation Server (TFS)
- ▶ Synergy
- ▶ Vault
- ▶ Veracity
- ▶ Vesta
- ▶ Visual SourceSafe (VSS)

Работа с GIT



Работа с GIT: Децентрализованность

GIT подразумевает несколько вариантов распределения. В самом простом варианте – он работает аналогично SVN – есть один центральный репозиторий в с которым взаимодействуют все разработчики.

Git не позволяет работать с удаленным репозиторием напрямую. Вместо этого разработчики работают с локальным репозиторием, который затем могут синхронизировать с удаленным



Схема с простым общим репозиторием

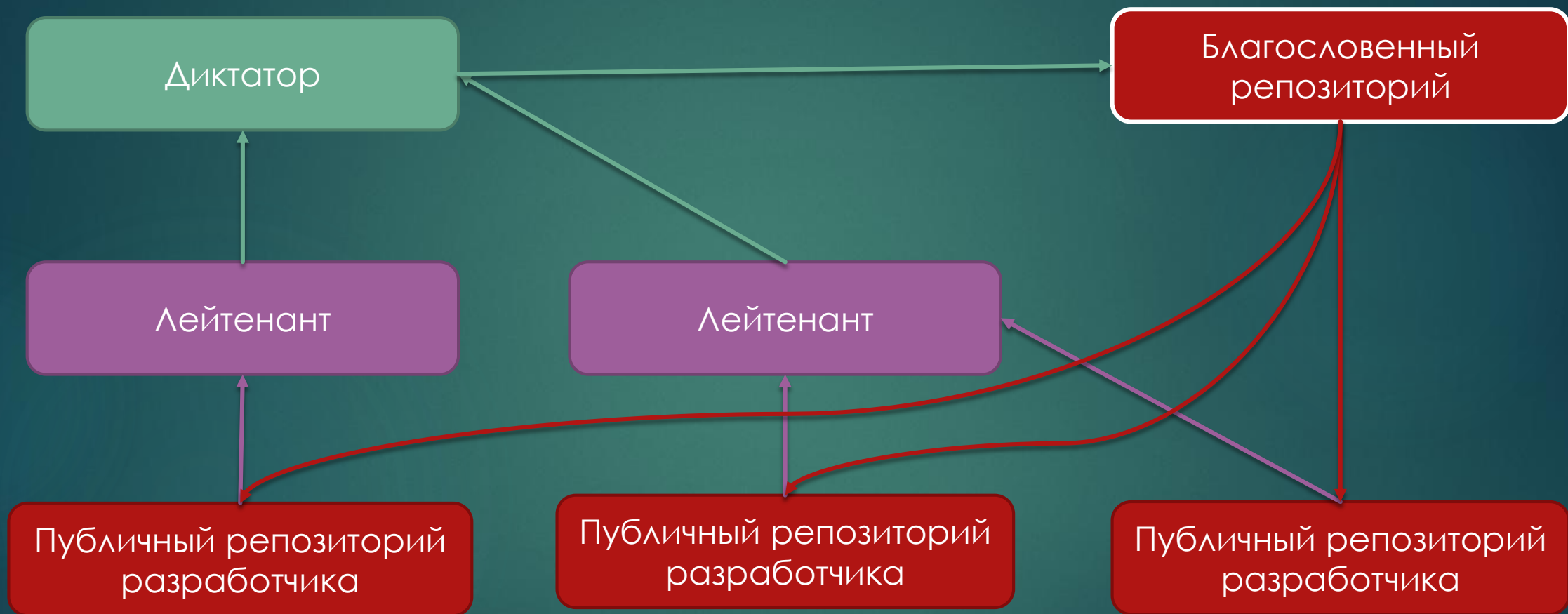
Работа с GIT: Децентрализованность

Помимо простой схемы с одним централизованным репозиторием GIT подразумевает так же и другие варианты



Схема с независимой непрерывной интеграцией

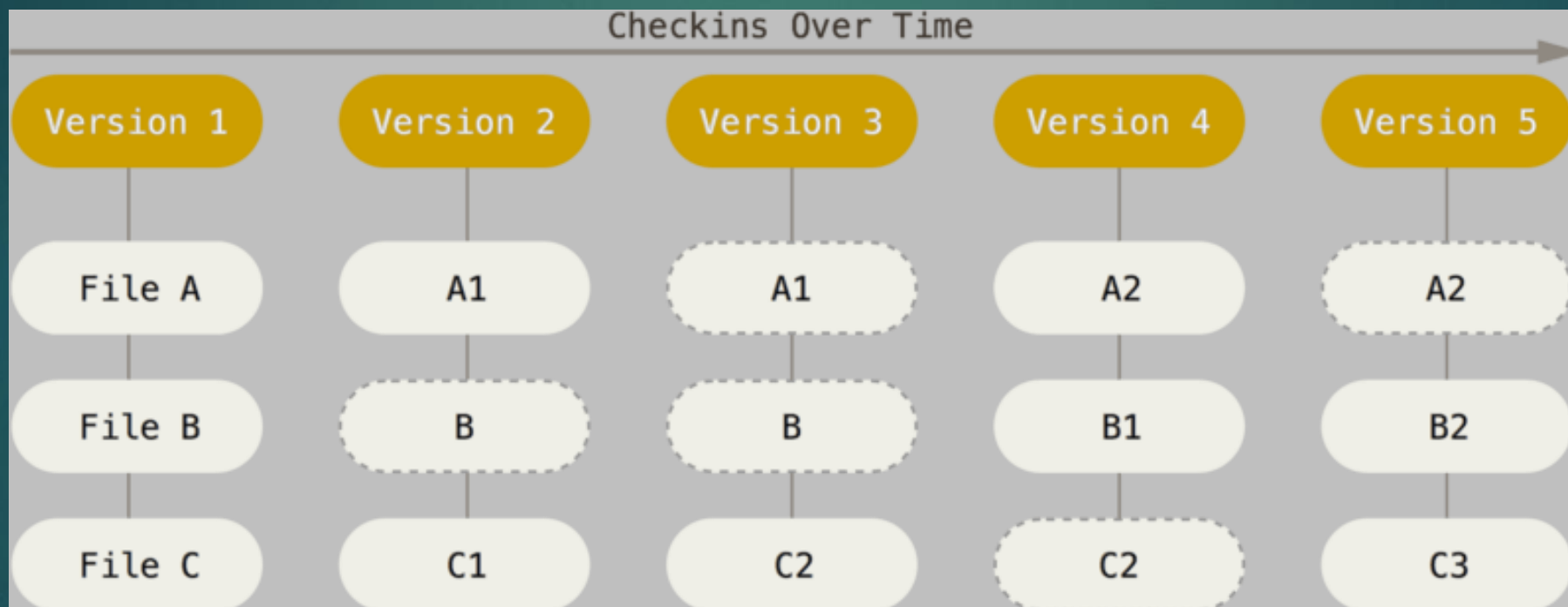
Работа с GIT: Децентрализованность



Иерархическая схема с великодушным диктатором

Работа с GIT: Ревизии

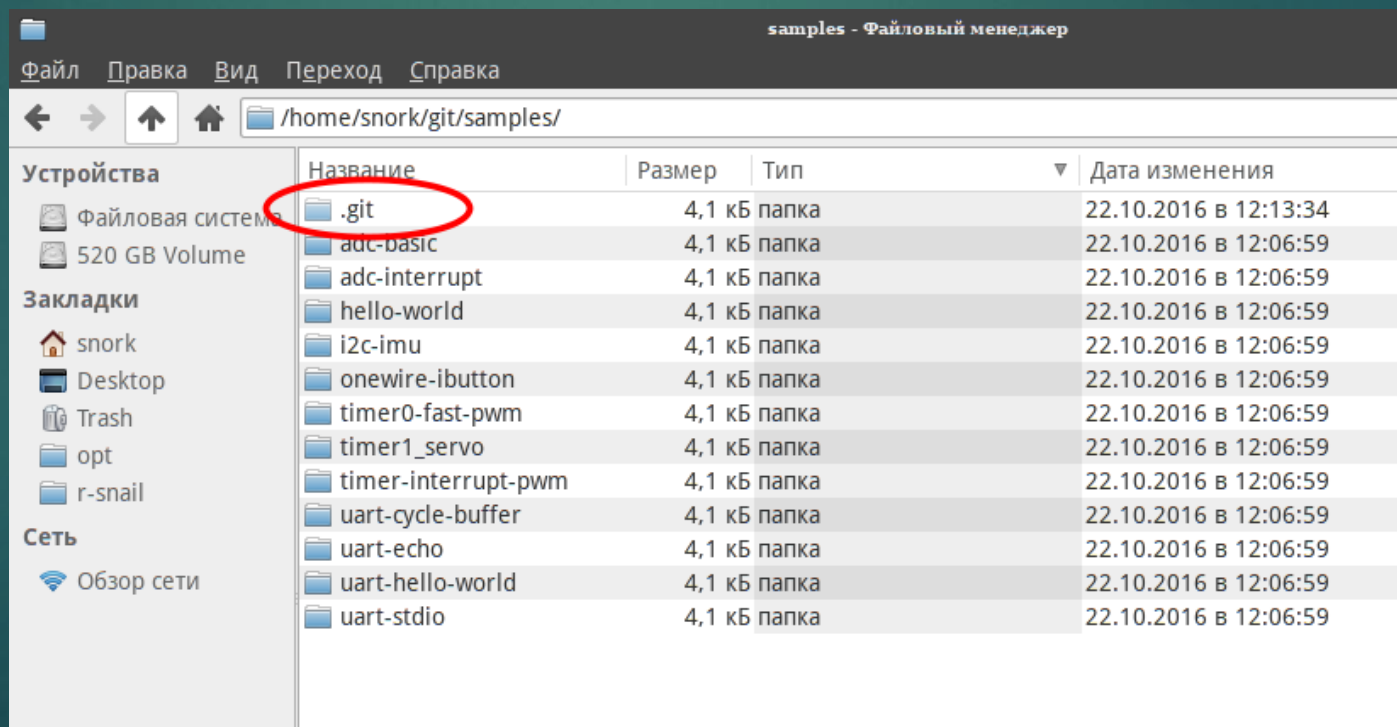
GIT хранит все изменения произошедшие с содержимым репозитория в виде наборов изменений называемых коммитами. Все коммиты имеют уникальные идентификаторы.



Ревизии в GIT

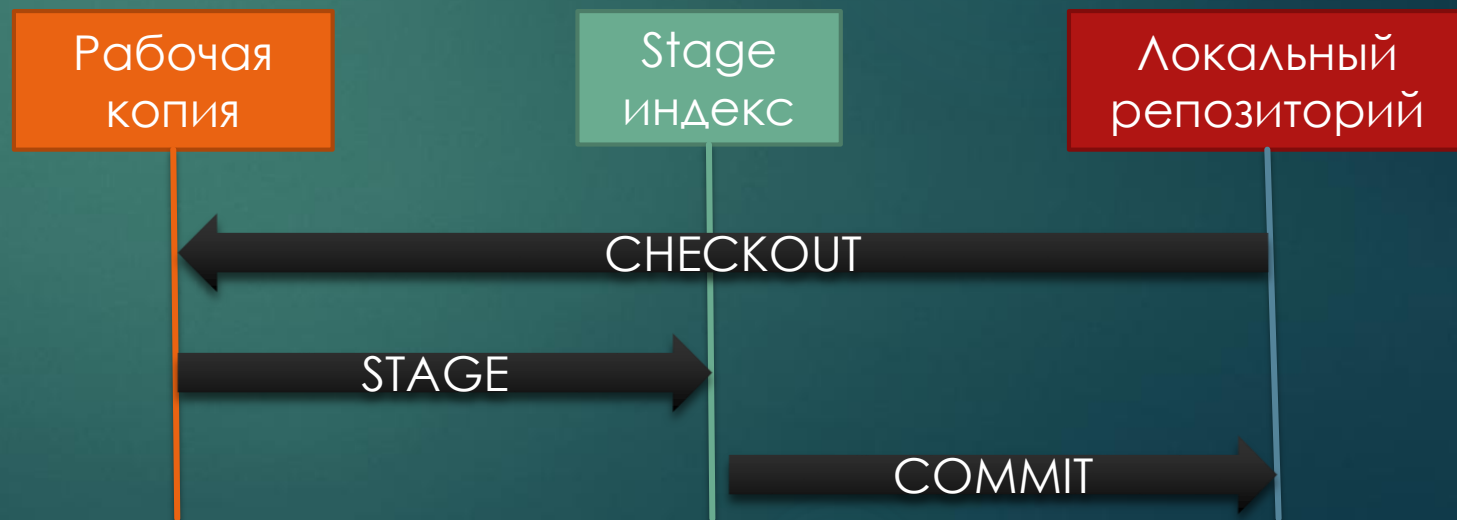
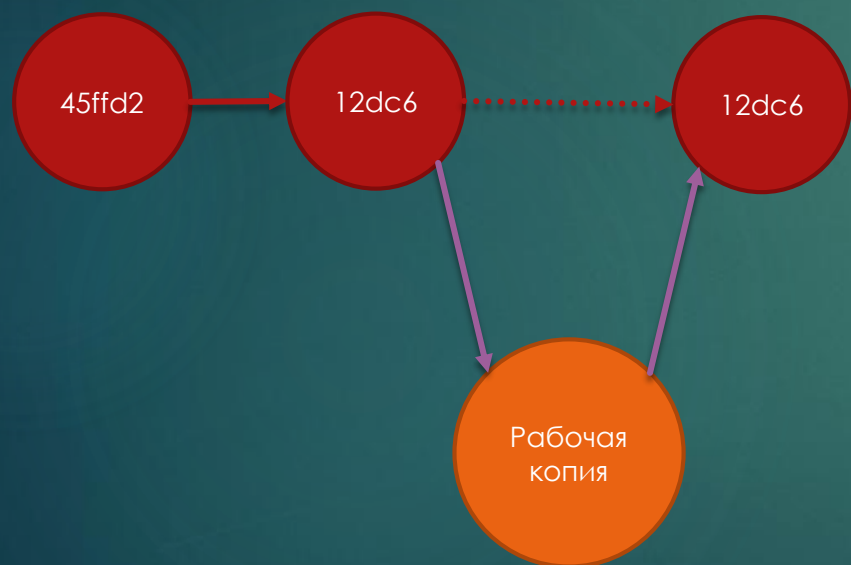
Работа с GIT: Взаимодействие с локальным репозиторием

Локальный репозиторий это простая папка на компьютере разработчика. Для локального репозитория существует только одна рабочая копия. Она и находится в папке репозитория. Сам же репозиторий скрыт в подпапке с именем `.git`



Git – внесение изменений

Для того, чтобы закрепить новый набор изменений в локальном репозитории нужно получить состояние «после выбранного коммита» в рабочую копию, внести в нее изменения и закрепить их как новый набор изменений в репозитории.



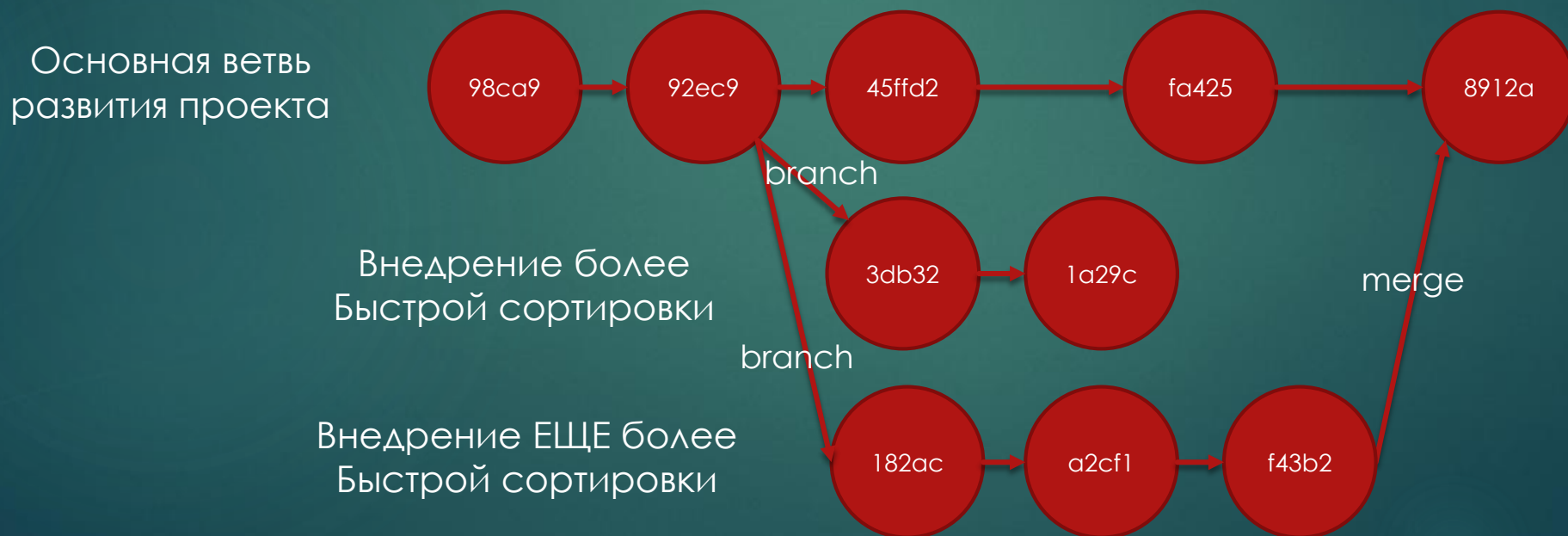
Ветвление и метки

Допустим, Вы хотите внести в проект новый большой и сложный функционал.

Если вносить все изменения огромным «коммитом», изменения будет тяжело отслеживать.

Если разбить это изменения на много маленьких «коммитов», то наверняка проект будет неработоспособен в своих промежуточных версиях, что может помешать другим разработчикам.

Компромиссным решением является создание ветви.



Конфликты

Клара у карла украла кораллы
а Карл у Клары украл **пианино**

Клара у карла украла кораллы
а Карл у Клары украл кларнет



Если сливаемые изменения
взаимоисключающие –
создается конфликт, который
придется разрешить вручную

master ("ours", ca09b440)	Working Tree	branch ("theirs", ed76d636)
1 Клара у Карла украла кораллы	1 Клара у Карла украла кораллы	1 Клара у Карла украла кораллы
2 а Карл у Клары украл пианино	2 <<<<<<< HEAD	2 а Карл у Клары украл тромбон
3	3 а Карл у Клары украл пианино	3
	4 =====	
	5 а Карл у Клары украл тромбон	
	6 >>>>>>> branch	
	7	

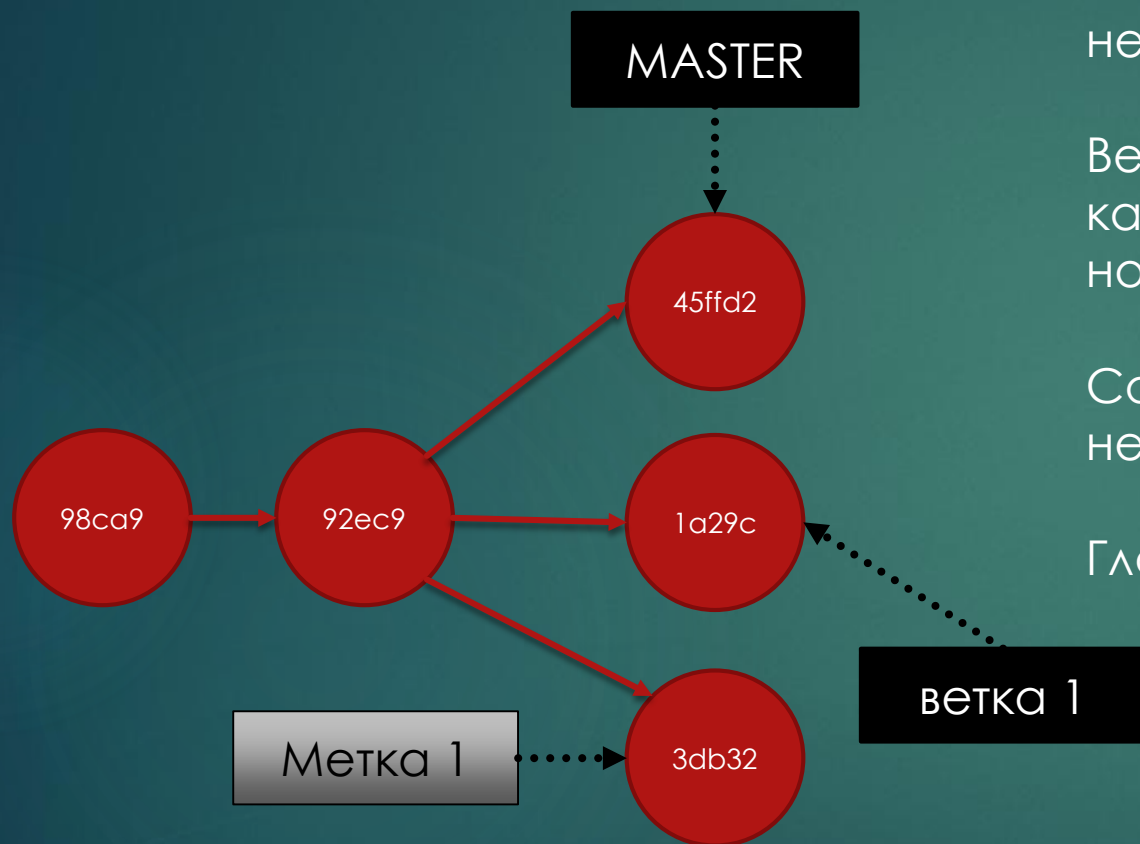
Метки

Когда проект наконец готов к релизу, очень полезно запомнить при каком состоянии репозитория это произошло. При дальнейшей разработке проекта репозиторий уйдет «вперед» и при возникновении каких-то ошибок будет тяжело соотнести собранный проект и его код.

Так же, нередко ситуация «в прошлой версии работало, а в этой нет». Если версии отмечены в репозитории их можно легко сравнить и проанализировать все изменения.



Ветвление и метки в GIT



Каждая ревизия (коммит) в GIT может иметь несколько потомков закрепленных в репозитории.

Ветка – это простой указатель, который указывает на какую либо ревизию и автоматически создает для нее нового потомка при операции commit.

Совершенно нормальной является ситуация, когда несколько веток указывают на один и тот же коммит

Главная называется **master**

Метки отличаются от веток лишь тем, что «в них» нельзя коммитить и как-либо их перемещать (но можно удалять)

git checkout, commit и reset

Предположим в репозитории есть некоторое начальное состояние и ветка **master** указывает на него.



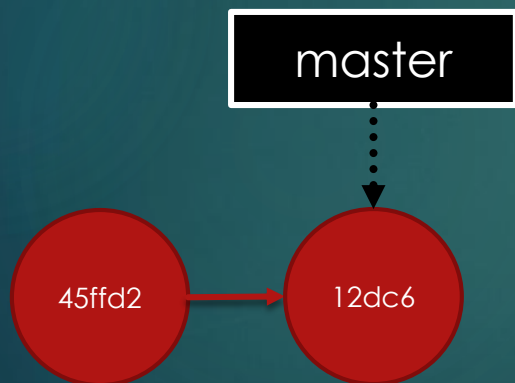
git checkout, commit и reset

Мы можем выгрузить это состояние в рабочую копию при помощи операции checkout. Затем внести изменения и создать новую ревизию в репозитории, которая станет потомком начальной. Указатель ветки master перенесется на новое состояние. Мы внесли коммит в эту ветку

Следует отметить, что операция checkout определена как для отдельных коммитов, так и для веток с метками.

Но checkout чего-либо кроме ветки. Автоматически создает новую «анонимную» ветку, которая будет утеряна, если не дать ей имя при помощи команды branch.

Проще говоря - коммитить без активной ветки - нельзя.



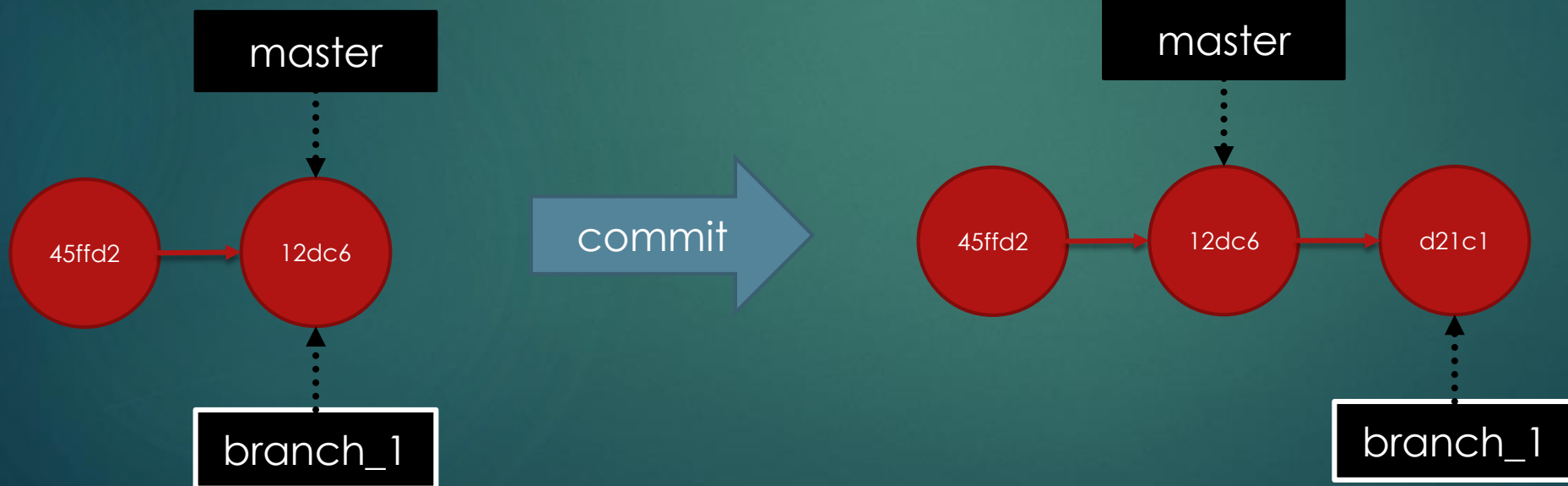
git checkout, commit и reset

Создадим новую ветку, по имени **branch_1** при помощи команды branch.

Сразу же перейдем на нее при помощи команды checkout

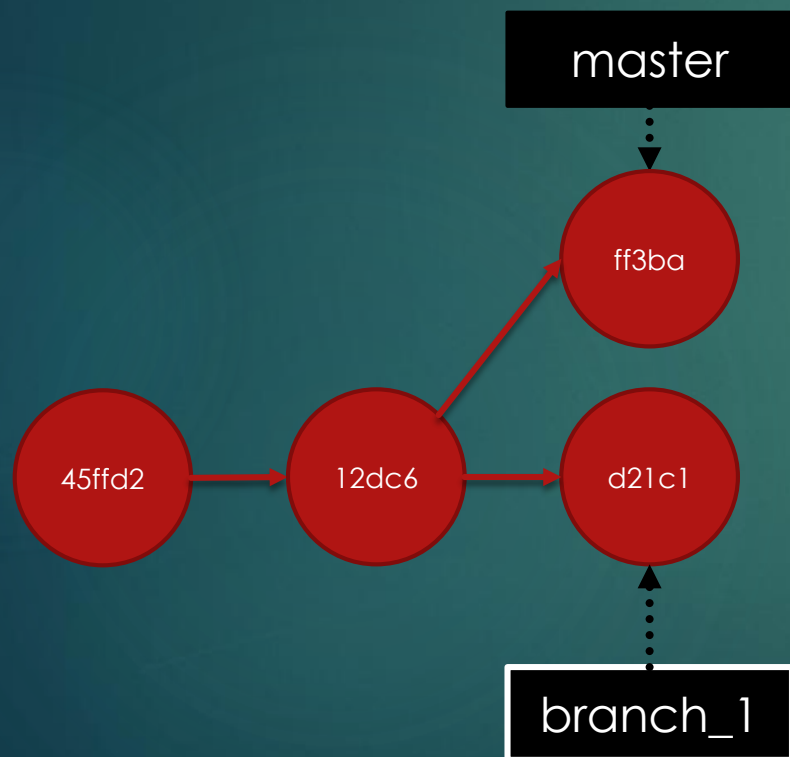
Добавим новый коммит в активную ветку **branch_1**

Это сдвинет её указатель на новый коммит. Ветка **master** останется на месте



git checkout, commit и reset

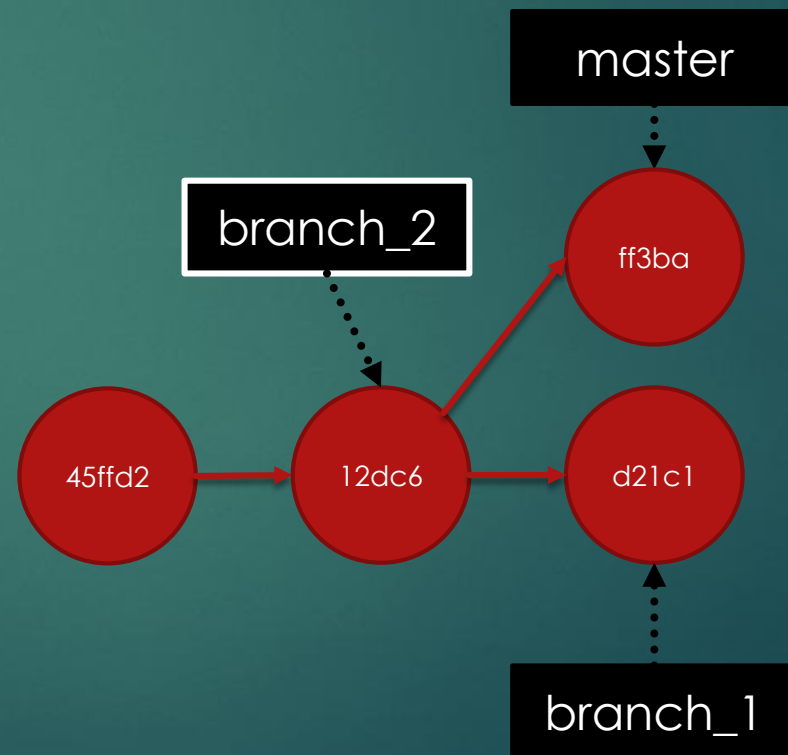
Переходим обратно на ветку **master** при помощи операции checkout и тоже добавляем новый коммит.



checkout
& branch

Мы можем перейти на любой коммит и начать новую ветку от него при помощи команды branch

Аналогично можно создавать метки при помощи команды tag

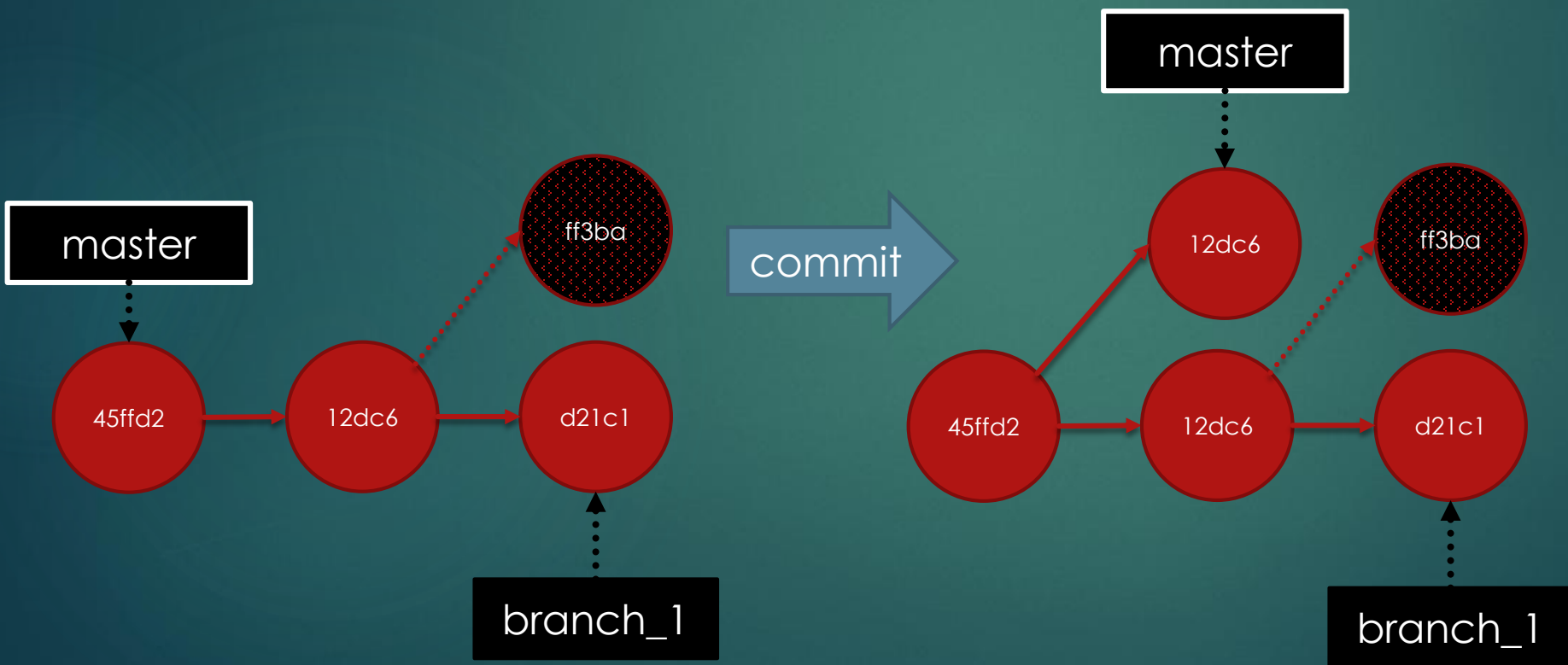


git checkout, commit и reset

Указатель ветки можно перемещать в произвольном направлении по графу коммитов при помощи операции reset.

При коммитах ветка может продолжить движение, но уже по другой цепочке коммитов

Коммиты, на которые не опирается ни одна ветка при этом оказываются в некотором подвешенном состоянии и могут быть удалены (в том числе и произвольно)

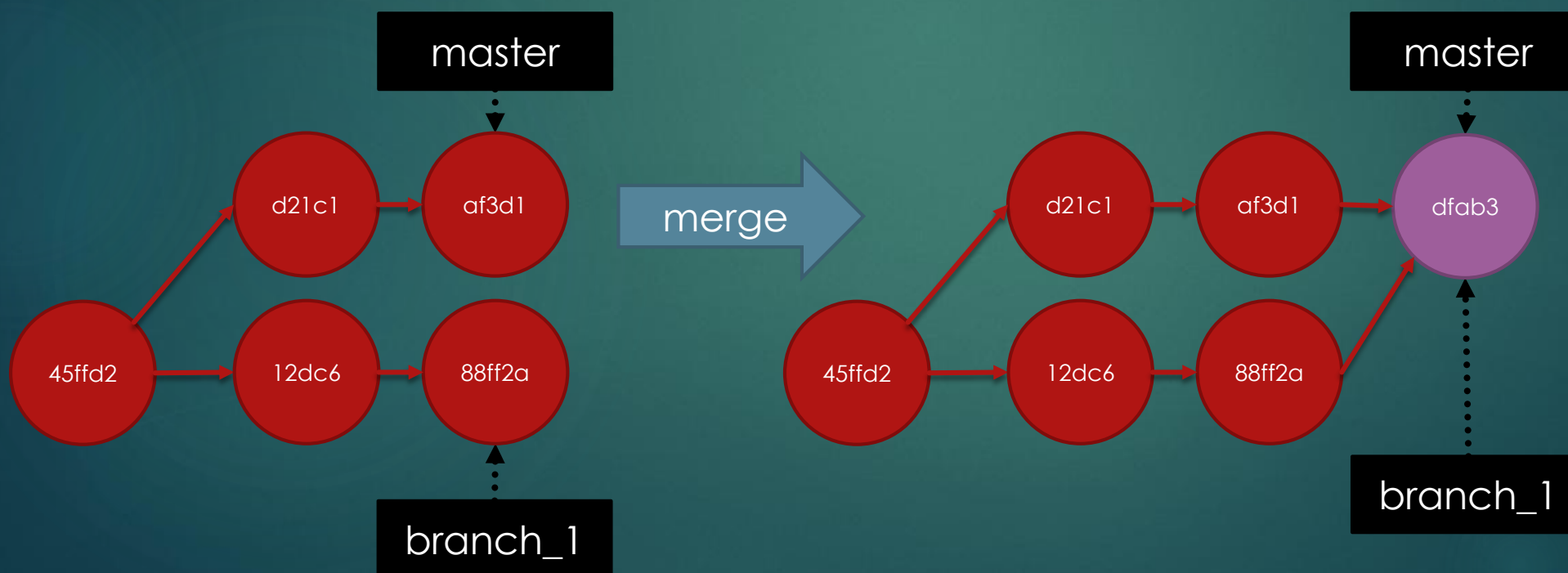


git merge

Слияние в git проводится тремя способами.

- 1) Merge (простое слияние)
- 2) Fast-forward (перемотка вперед)
- 3) Cherry-pick (слияние отдельных коммитов)

Простое слияние
ВЕТВИ **branch_1** В ВЕТЬ **master**

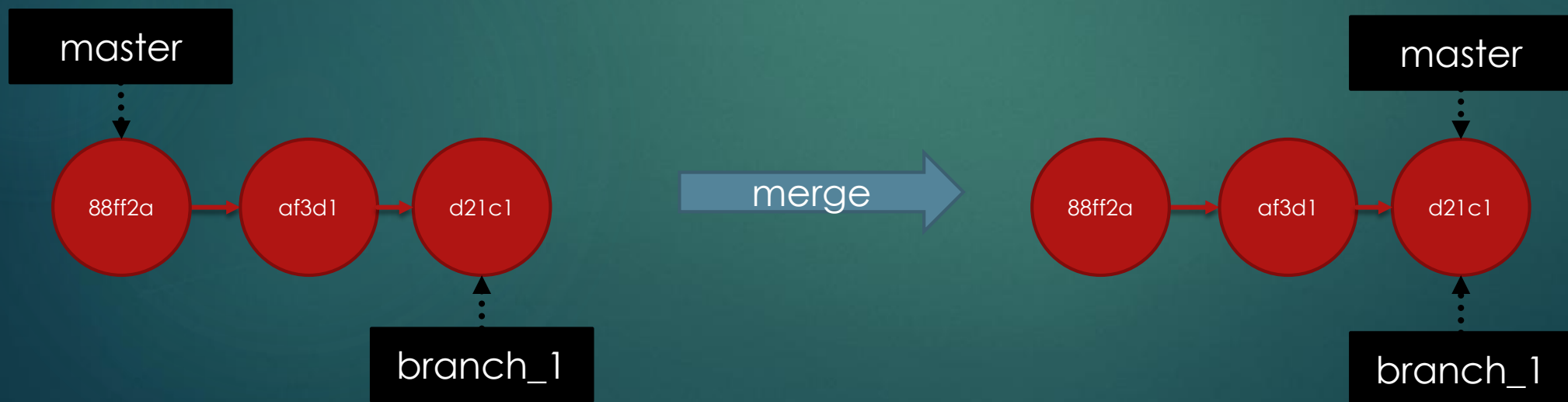


При простом
слиянии создается
специальный
merge-commit

git merge fast-forward

Слияние перемоткой вперед возможно, только при наличии прямого пути «вперед» по графу коммитов для одной из сливаемых веток. Порядок слияния **master->branch_1** или **branch_1->master** – имеет значение.

Слияние перемоткой вперед
ветви **branch_1** в ветвь **master**



При слиянии
перемоткой
вперед
коммитов
не создается

git cherry-pick

Операция cherry-pick не полноценное слияние. Это операция позволяет создать в одной из веток копию коммита из другой.

При этом под коммитом тут понимается не «слепок» состояния репозитория, а набор изменений, приведший к такому состоянию

Копирование коммита между из ветви **branch_1** в ветвь **master**



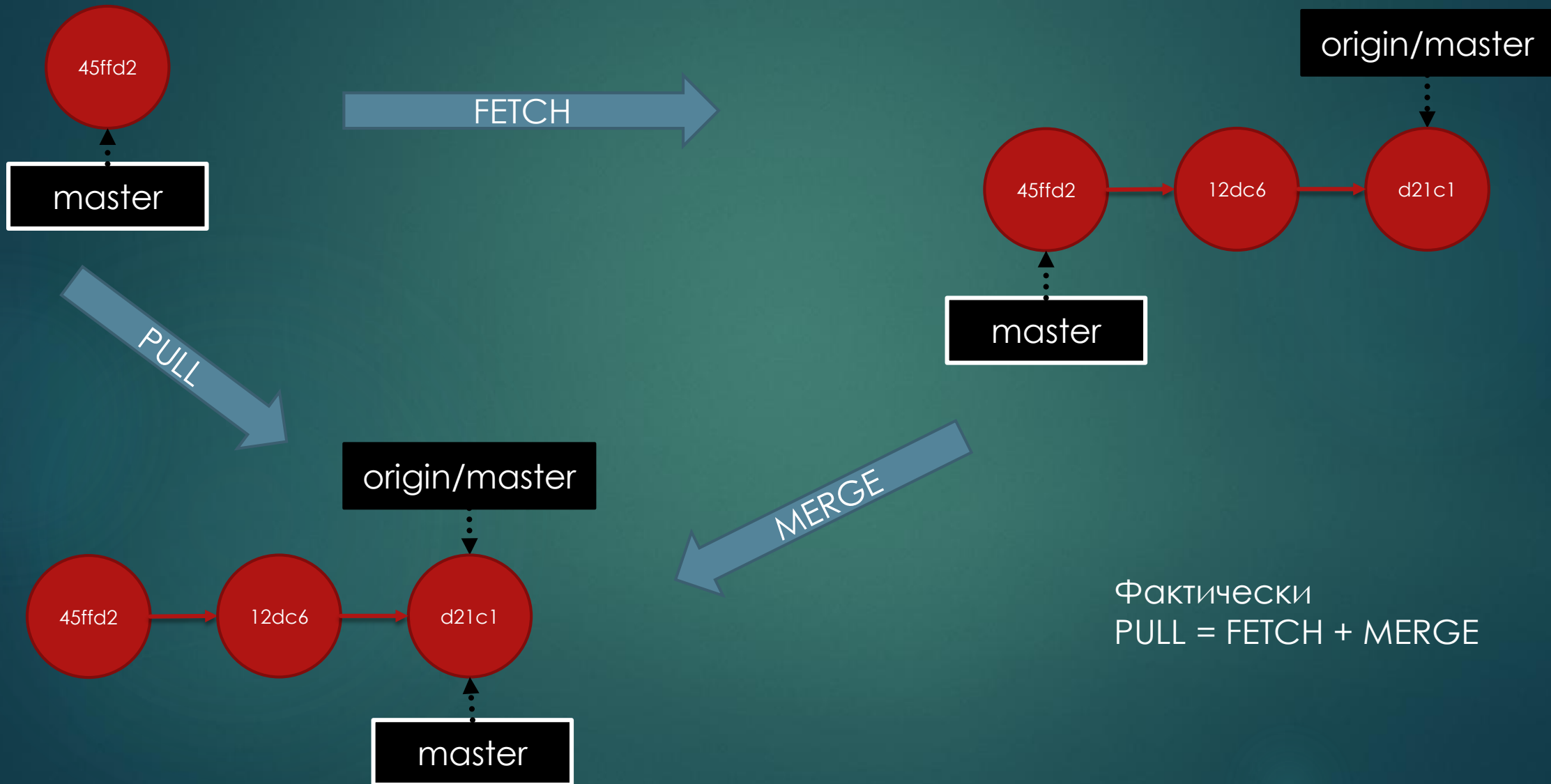
Обмен коммитами между репозиториями

Коммиты между репозиториями передаются путем слияния соответствующих удаленной и локальных веток методом **fast_forward**.

Для обмена данными между репозиториями определены три операции

- ▶ CLONE – Создание полного клона удаленного репозитория локально
- ▶ PUSH – слияние локальной ветки на удаленную (передача локально созданных коммитов)
- ▶ FETCH – получение удаленных коммитов без слияния (слияние потом можно осуществить в ручную)
- ▶ PULL – слияние удаленной ветки на локальную (полученные удаленных коммитов)

Обмен коммитами между репозиториями: FETCH и PULL



Обмен коммитами между репозиториями: PUSH

origin/master



PUSH

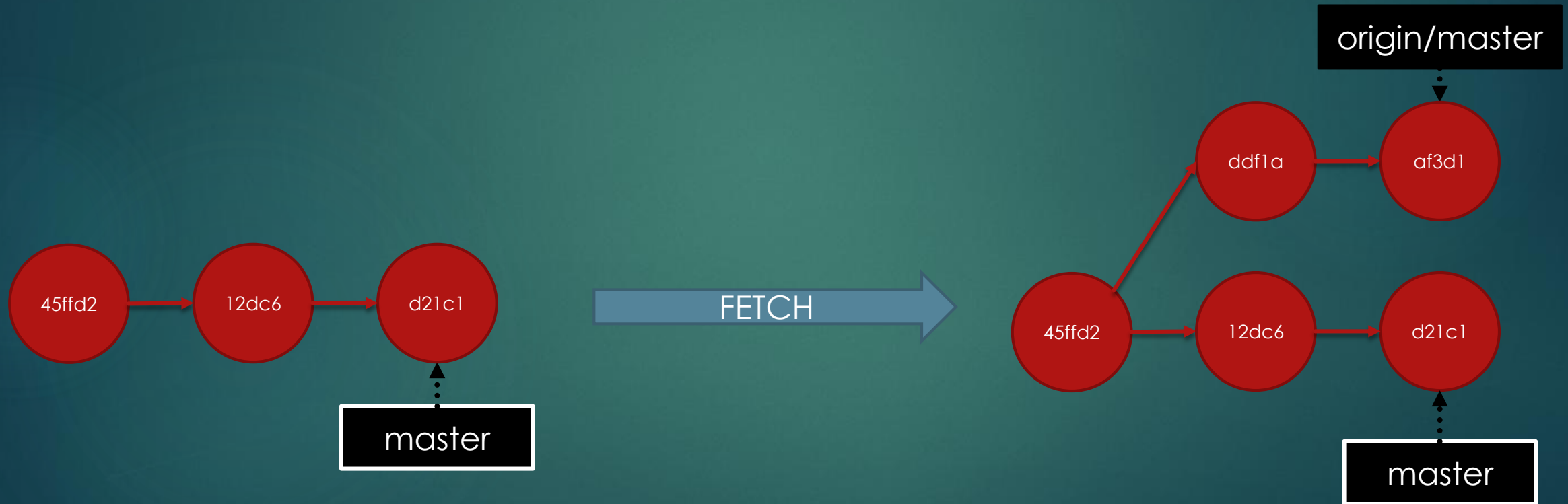
origin/master



git rebase - перебазирование ветвей


Редко удастся сделать fast-forward слияние при получении коммитов из удаленного репозитория

Гораздо чаще получается что-то наподобие такого



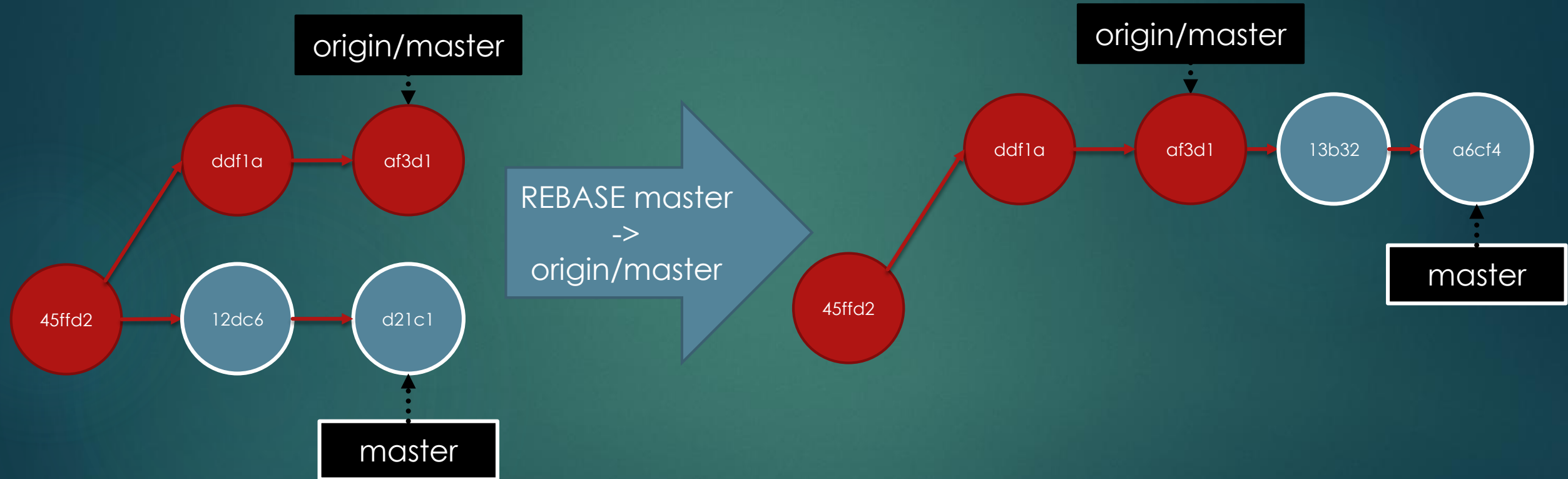
git rebase - перебазирование ветвей

Когда в разработке участвует много людей, история коммитов разбавленная merge-коммитами начинает выглядеть плохо

Message	Date	Author
 ОЧЕНЬ ДОСАДНАЯ ОШИБКА	20.05.2016 16:20	vasilly.prokopyev
Merge branches 'develop-rsce' and 'develop-rsce' of https://bitbucket.org/sotrsiss/sot_core i...	20.05.2016 15:39	unknown
Меньше сущностей	20.05.2016 15:34	unknown
Доделал до какого-то вида управление сборкой	20.05.2016 15:31	vasilly.prokopyev
Таск менеджео ориентируется на таблицы теперь	20.05.2016 15:16	unknown
Поправил созданные таблицы - заменил data_id на take_id и take_type. Ну и приделал л...	20.05.2016 12:46	vasilly.prokopyev
Улучшенная пагинация	19.05.2016 16:54	unknown
Заготовка под систему управления таск менеджером	19.05.2016 14:10	unknown
Merge branch 'develop-rsce'	19.05.2016 11:19	vasilly.prokopyev
Добавил таблицы для запросов на сборку data_id и для параметров TaskManager-a	19.05.2016 10:50	vasilly.prokopyev
Merge branch 'develop-rsce'	18.05.2016 15:03	vasilly.prokopyev
Фикс фабрики и косяка с legacy_schema_aliasing. оказывается наша бд - legacy :\ (cher...	18.05.2016 15:01	vasilly.prokopyev
мелкие правки	18.05.2016 13:01	Vasily.Prokopyev
Добавил черновичек тестов (удалил общие конфигураци, т.к. они судя по всему не ра...	18.05.2016 11:17	vasilly.prokopyev
Теперь на титульной странице можно переключать отображаемую базу	18.05.2016 09:28	unknown
Переделал механизм выбора конфигурации на переменные среды	17.05.2016 13:01	vasilly.prokopyev
Добавил фабрики для создания тестовых данных	17.05.2016 11:27	vasilly.prokopyev
Исправил мелкие огрехи и (кажется окончательно) доделал миграции	17.05.2016 11:10	vasilly.prokopyev
Merge branch 'develop-rsce' of https://bitbucket.org/sotrsiss/sot_core into develop-rsce	13.05.2016 13:34	unknown
ДОбавление заглушек на битые задания + празные правки	13.05.2016 13:31	unknown
Миграции опять идут с самого начала :(12.05.2016 13:48	vasilly.prokopyev
Забыл новую версию БД	12.05.2016 13:34	vasilly.prokopyev
Подправил каскады и описания именуемых сущностей БД, для корректной работы ...	12.05.2016 13:13	vasilly.prokopyev
Новые зависимости! :(Добавил затравочку под JSON-RPC взаимодействие с шефом.	12.05.2016 13:13	vasilly.prokopyev

git rebase - перебазирование ветвей

Решение – перебазирование коммитов (смена предка)

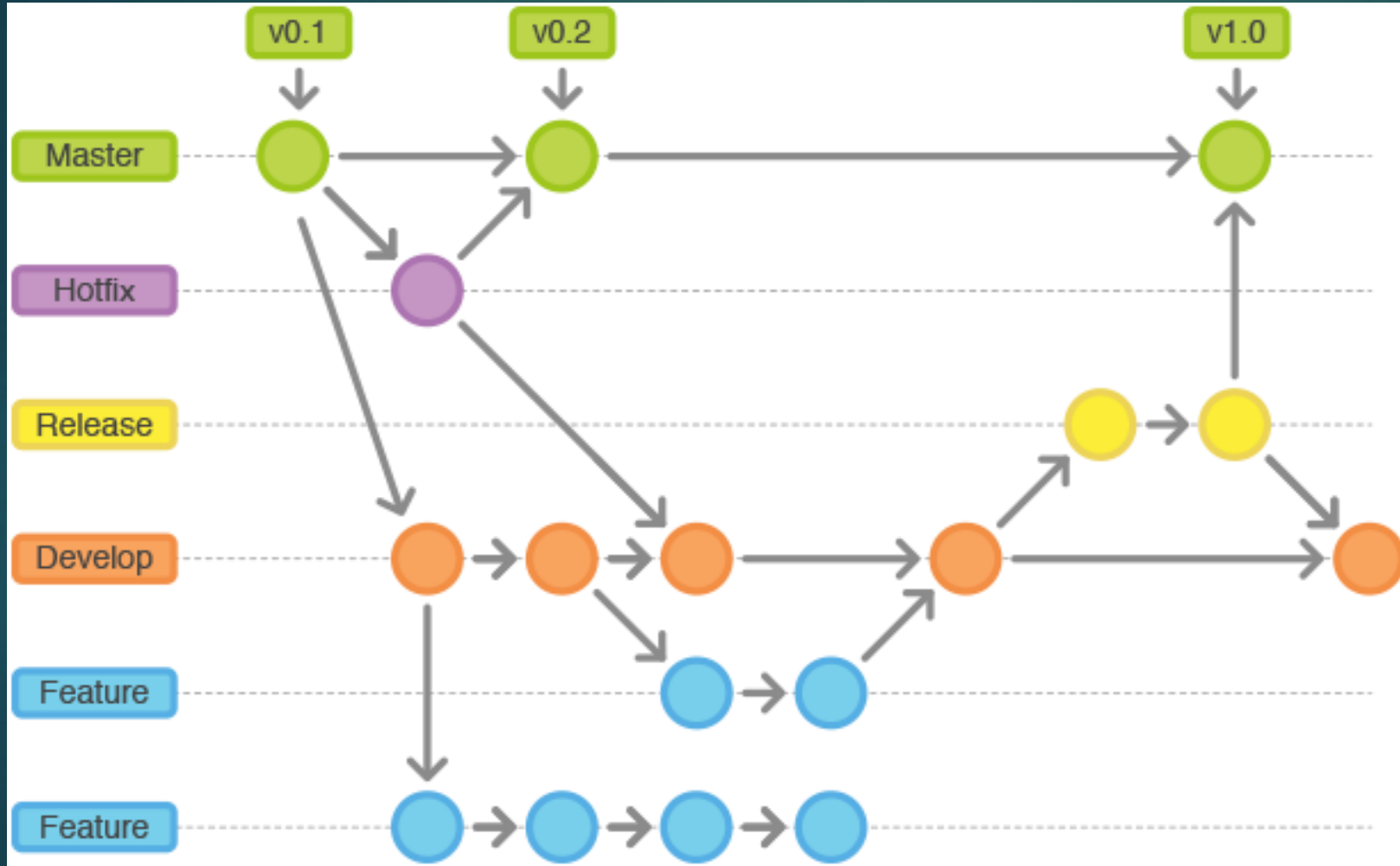


Ahh, but the bliss of rebasing isn't without its drawbacks, which can be summed up in a single line:

Do not rebase commits that exist outside your repository.

If you follow that guideline, you'll be fine. If you don't, people will hate you, and you'll be scorned by friends and family.

Повседневная работа с GIT



Материалы для изучения:
<https://git-scm.com/book/ru/v1>

ОСНОВНЫЕ КОМАНДЫ И УТИЛИТЫ GIT

- ▶ **git checkout** – получение ветки/метки/коммита в рабочую копию;
- ▶ **git stage / git commit** – индексирование и закрепление изменений в локальном репозитории;
- ▶ **git branch / git tag** – создание новой ветки/метки;
- ▶ **git merge** – слияние ветвей;
- ▶ **git reset** – перевод указателя ветки на указанный коммит;
- ▶ **git push / git pull** – передача данных удаленному репозиторию;
- ▶ **git rebase** – перебазирование ветки;

ОСНОВНЫЕ КОМАНДЫ И УТИЛИТЫ GIT

► **git log** – просмотр истории коммитов

Message	Date	Author
 Добавил openlayers на страницу выбора маршрута + добавлена агрегац...	09.09.2016 11:34	Sergey
Merge branch 'develop-rsce'	07.09.2016 16:27	vasilly.prokopyev
Костыльная правка пути файла	07.09.2016 16:18	vasilly.prokopyev
Забытый реф	17.06.2016 09:52	vasilly.prokopyev
Пагинация на странице выбора маршрута	20.07.2016 11:28	unknown
Merge remote-tracking branch 'origin/master' into develop-rsce	20.07.2016 10:13	unknown
Теперь точно добавил фильтр по bandam для каждого chunka.	29.06.2016 15:59	Sergey
Revert "Добавил фильтр по bandam для каждого chunka."	29.06.2016 15:52	Sergey
Добавил фильтр по bandam для каждого chunka.	29.06.2016 15:48	Sergey
Добавил фильтрацию (пока только по band)	17.06.2016 15:15	Sergey
Косяк в пагинации дтф и ансилари	17.06.2016 12:50	unknown
Небольшая ошибка в пагинации + добавление "просмотра" квиклуков на страницей с ...	10.06.2016 13:27	unknown
Ряд разных правок, втч таскменеджера	07.06.2016 15:03	unknown
Добавление пагинации в странице просмотра dtf файлов	03.06.2016 15:51	unknown
Забыто	02.06.2016 16:26	unknown
Добавление просмотра метаданных второго уровня	02.06.2016 13:39	unknown
Добавление новых заданий в ВИД	02.06.2016 12:04	unknown
Merge branch 'develop-rsce'	01.06.2016 14:21	vasilly.prokopyev
Новые таблицы для нового типа заданий на выделение метаданных	31.05.2016 12:59	vasilly.prokopyev
Merge branch 'develop-rsce'	30.05.2016 12:52	vasilly.prokopyev
опять ошибки в триггерах	30.05.2016 12:50	vasilly.prokopyev
Разные багфиксы	27.05.2016 16:00	unknown
Добавление выбора режима обработки по умолчанию и отображение флага радиомет...	27.05.2016 10:50	unknown
Пагинация добавлена в просмотрщик апс-файлов	27.05.2016 09:23	unknown

ОСНОВНЫЕ КОМАНДЫ И УТИЛИТЫ GIT

► git blame – ПОИСК АВТОРА И ДАТЫ ПРАВКИ

```
View Commit: 07.09.2016 - 94cf7f9b - vasily.prokopyev - Merge branch 'develop-rsce'
Highlight: Changes Since Commit: 17.06.2016 - 0e8897ac - vasily.prokopyev - Забытый реф

225 # Алексей: этот костыль, похоже, неизбежно порождает +1 запрос в БД на каждый вызов have_all_packets :(
226
98e97e96 vasily.prokopyev 224d 227 _have_all_packets = staticmethod(lambda data_id: func.dbo.is_data_id_ready(data_id))
228
229 @hybrid_property
230 def have_all_packets(self):
231     return db.session.execute(self._have_all_packets(self.data_id)).scalar()
232
233 @have_all_packets.expression
234 def have_all_packets(cls):
235     return func.dbo.is_data_id_ready(cls._have_all_packets(cls.data_id))
236
237 __tablename__ = 'data_ids'
238 __table_args__ = (
239     UniqueConstraint('take_id', 'take_type',
240                     {'schema': 'dbo'})
241 )
242
1c105f9c vasily.prokopyev 223d 243 ref_back_packets = relationship("Packet", back_populates="ref_data_id")
244
e3e7e1d7 M vasily.prokopyev 195d 245 ref_back_l0_file = relationship("LOFile", back_populates="ref_data_id")
246 ref_back_metadata = relationship(
247     "L1Metadata",
248     primaryjoin="and_(DataId.take_id == L1Metadata.data_id_3_bytes, DataId.take_type == L1Metadata.type)",
249     foreign_keys=["L1Metadata.data_id_3_bytes, L1Metadata.type"],
250     back_populates="ref_data_id",
251 )
252
e6743706 vasily.prokopyev 222d 253 ref_back_l0p_index = relationship("L1PIndex", back_populates="ref_data_id")
1c105f9c vasily.prokopyev 223d 254 ref_back_l1a_file = relationship("L1AFile", back_populates="ref_data_id")
e6743706 ~ vasily.prokopyev 222d 255 ref_back_chunks = relationship("Chunk", back_populates="ref_data_id")
1c105f9c vasily.prokopyev 223d 256
257
e6743706 vasily.prokopyev 222d 258 ref_back_l0_tasks = relationship("LOTask", back_populates="ref_data_id")
259 ref_back_l1as_tasks = relationship("L1ASearchTask", back_populates="ref_data_id")
260 ref_back_l1am_tasks = relationship("L1AMakeTask", back_populates="ref_data_id")
261
History of current line x
98e97e96 vasily.prokopyev 224d _have_all_packets = staticmethod(lambda data_id: func.dbo.is_data_id_ready(data_id))
? ? ? <repository creation>
```

GIT Клиенты

Аналогично с SVN – нативный клиент – клиент командной строки. Он является наиболее мощным и реализует функционал GIT полностью

Графических клиентов так же великое множество.

Самые приятные на мой взгляд:

- ▶ Плагин для PyCharm
- ▶ SmartGit
- ▶ SourceTree
- ▶ GitKraken

Облачные сервисы

Git де-факто стал отраслевым стандартом системы контроля версий. Не в последнюю очередь благодаря облачным сервисам, таким как GitHub.

GitHub предоставляет удаленные Git репозитории с удобным веб-интерфейсом управления ими и сопровождения проектов. Если проект использует OpenSource лицензию и не ограничивает доступ к своим репозиториям на чтение – GitHub предоставляет свои услуги бесплатно.

Популярность этого ресурса невероятно высока. На него постепенно переносятся очень многими организациями проекты с открытыми исходными кодами.

Среди них:

- ▶ ID software (doom 3 и прочее)
- ▶ Valve (Source engine)
- ▶ Nasa (Apollo 11)
- ▶ Microsoft
- ▶ Xiaomi
- ▶


Github используется как репозиторий пакетов для Node.js, python-pip и многих других фреймворков динамических языков программирования

Облачные сервисы

Комментарии к правкам на BitBucket

data_worker.cpp

Side-by-side diffView fileComment...

**Vasily Prokopyev** REPO OWNER

Не совсем понятно зачем нужен вообще этот класс. Нужно обсудить

Reply • Edit • Delete • 2016-10-20


11234567

```
#include "data_worker.h"

-data_worker::data_worker()
+data_worker::data_worker(const std::string &filename_param) : filename(filename_param)
{
}
```

91011

```
data_worker::~data_worker()
{
}
```

**Vasily Prokopyev** REPO OWNER

Лучше использовать колбек в стиле c++. Есть объект `std::function`, который способен завернуть в себя вызов абсолютно любой функции при помощи `std::bind` (как это делается в asio через `boost::bind`).

доки и пример <http://en.cppreference.com/w/cpp/utility/functional/function>

Reply • Edit • Delete • 2016-10-20

1415161718

```
+{
+    callback_object_p = callback_object_param;
+    callback_f = callback_f_param;
+    v_handle_data();
+}
```

Облачные сервисы

Фактически – github это социальная сеть ориентированная на разработку ПО

The screenshot shows the GitHub profile of Linus Torvalds. At the top, there's a navigation bar with the GitHub logo, a search bar, and links for Pull requests, Issues, and Gist. The profile header includes a profile picture, the name 'Linus Torvalds', the username 'torvalds', and a 'Follow' button. Below this, it says 'Block or report user'. The 'About' section lists 'Linux Foundation', 'Portland, OR', and 'Joined on 3 Sep 2011'. The 'Organizations' section shows the Linux Foundation logo. The 'Popular repositories' section lists three repositories: 'linux' (Linux kernel source tree, C, 39k stars, 15k forks), 'subsurface' (Advanced multi-platform divelog based on Qt, C++, 1.1k stars, 281 forks), and 'libdivecomputer' (Trial balloon of upstream merge - don't use, C, 49 stars, 5 forks). The '2,231 contributions in the last year' section features a heatmap showing contributions from November to October. The 'Contribution activity' section shows a timeline for November 2016, with a 'Jump to' dropdown set to 2016.

Search GitHub

Pull requests Issues Gist

Overview Repositories 3 Stars 2 Followers 45.9k Following 0

Linus Torvalds
torvalds
Follow

Block or report user

Linux Foundation
Portland, OR
Joined on 3 Sep 2011

Organizations

Popular repositories

- linux**
Linux kernel source tree
C 39k 15k
- subsurface**
Advanced multi-platform divelog based on Qt
C++ 1.1k 281
- libdivecomputer**
Trial balloon of upstream merge - don't use
C 49 5

2,231 contributions in the last year

Nov Dec Jan Feb Mar Apr May Jun Jul Aug Sep Oct

Mon
Wed
Fri

Learn how we count contributions.

Less More

Contribution activity

Jump to 2016

November 2016 2015

Created 70 commits in 1 repository

Правила хорошего тона при работе с SVN

- ▶ Проект в ветви trunk всегда должен быть работоспособен (компилироваться и хотя бы запускаться)
- ▶ Комментарии к «коммитам» должны быть развернутые и осмысленные. Не следует писать «добавил два новых файла». Следует писать, например, «добавил поддержку протокола SFTP».
- ▶ Одно функциональное изменение – один «коммит». Не пакуйте в один коммит много нововведений. Так же не следует размазывать относительно простое нововведение в несколько «коммитов».

Спасибо за
внимание