# РАСПРЕДЕЛЕННЫЕ VCS. КАЧЕСТВЕННЫЕ КОММИТЫ

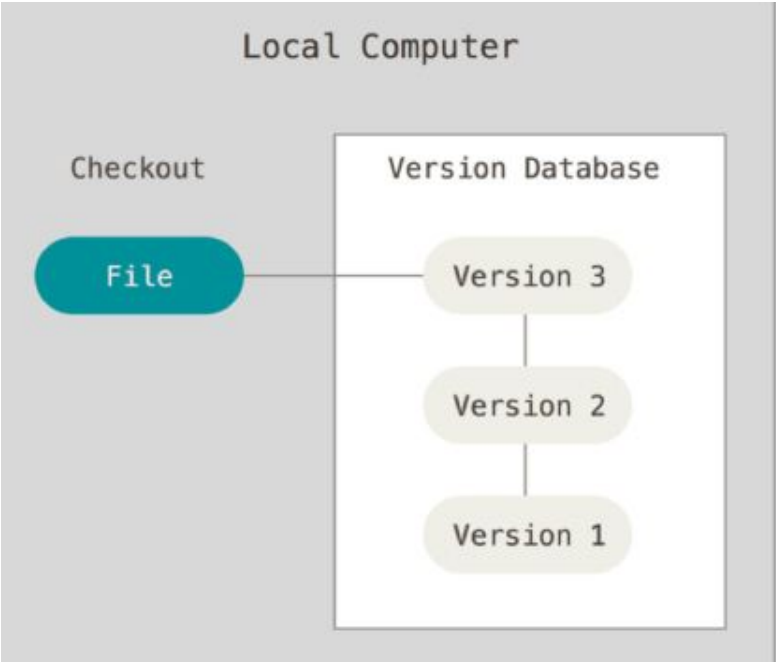
Сессия 1

МАРТ 14, 2016

# В ДАЛЕКОМ ПРОШЛОМ

Первые системы управления версиями (VCS) отслеживали изменения в файлах непосредственно на компьютере пользователя.

Обычно, они работали с несколькими копиями одного и того же файла чтобы имитировать ветви.

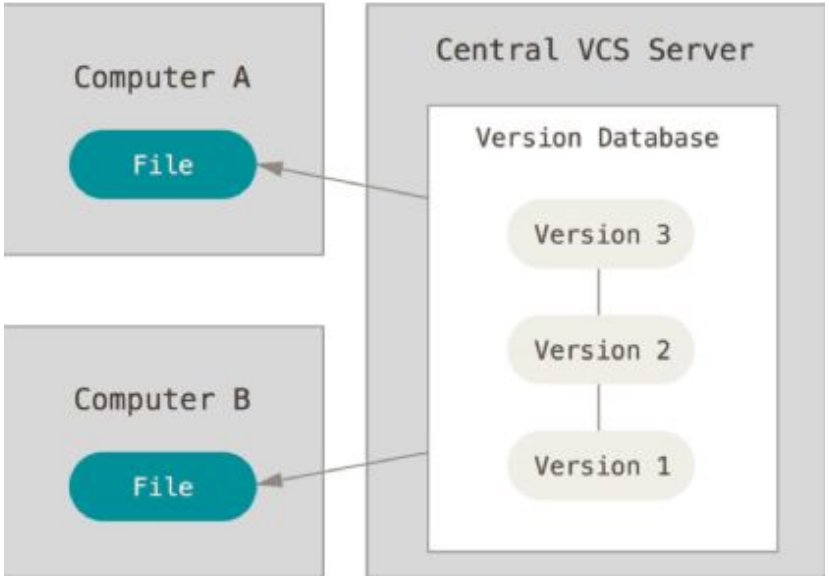


# ЕДИНЫЙ СЕРВЕР

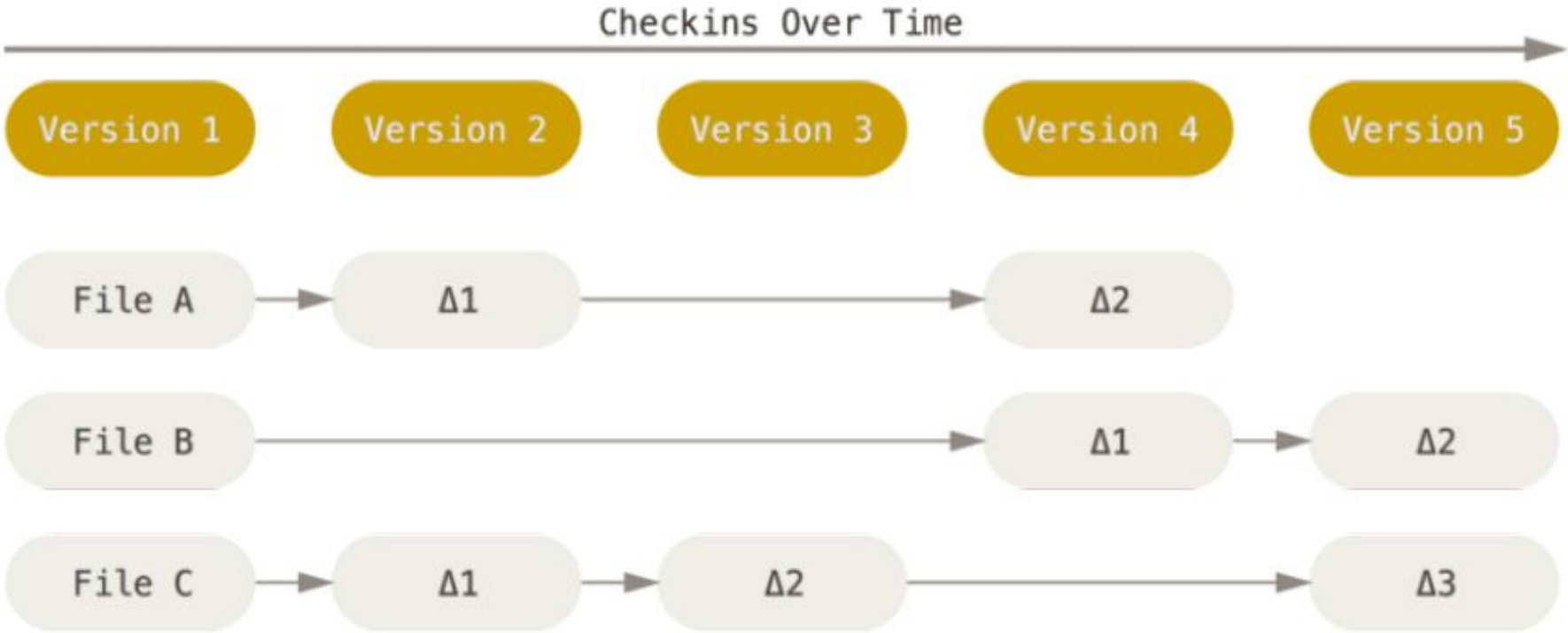
Схожий подход для сохранения изменений,
но уже в центральном хранилище,
был применен такими VCS,
как Subversion, CVS и Perforce.

Создание новой ветви в подобных системах
означает создание новой копии файловой
структуры.

Сбои в работе центрального сервера
могли блокировать работу на длительное время.



# ТАКИЕ СИСТЕМЫ ХРАНИЛИ ДЕЛЬТЫ ФАЙЛОВ



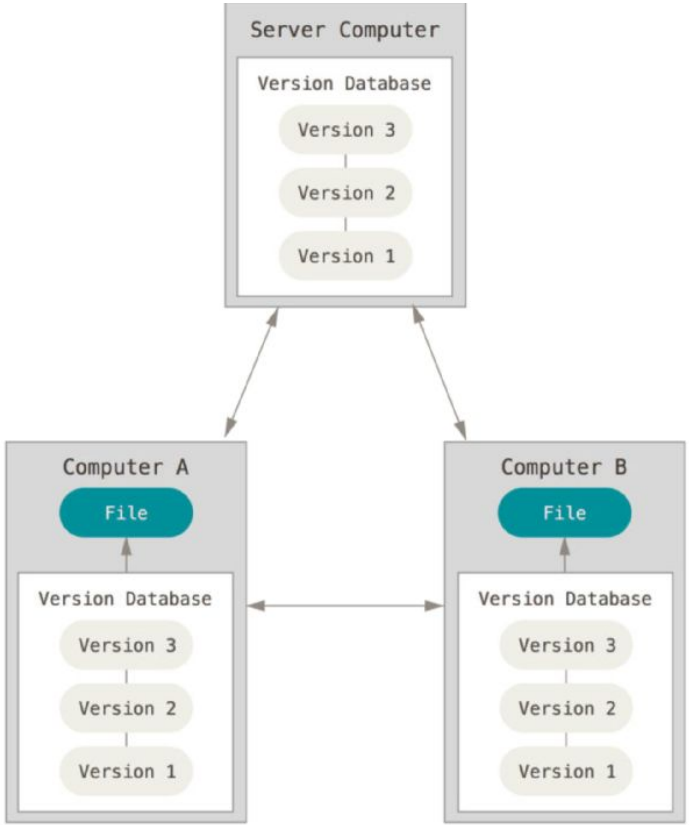
# РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ

В распределенных DVCS (Git, Mercurial, Bazaar или Darcs), клиенты не просто получают последнюю версию файла: они полностью копируют репозиторий.

Если один из серверов с DVCS выходит из строя, любой из клиентских репозиториев может быть скопирован на сервер для его восстановления.

Каждая синхронизация с сервером — создание полной резервной копии на локальном компьютере.

**В распределенных DVCS большинство операций выполняются локально!**

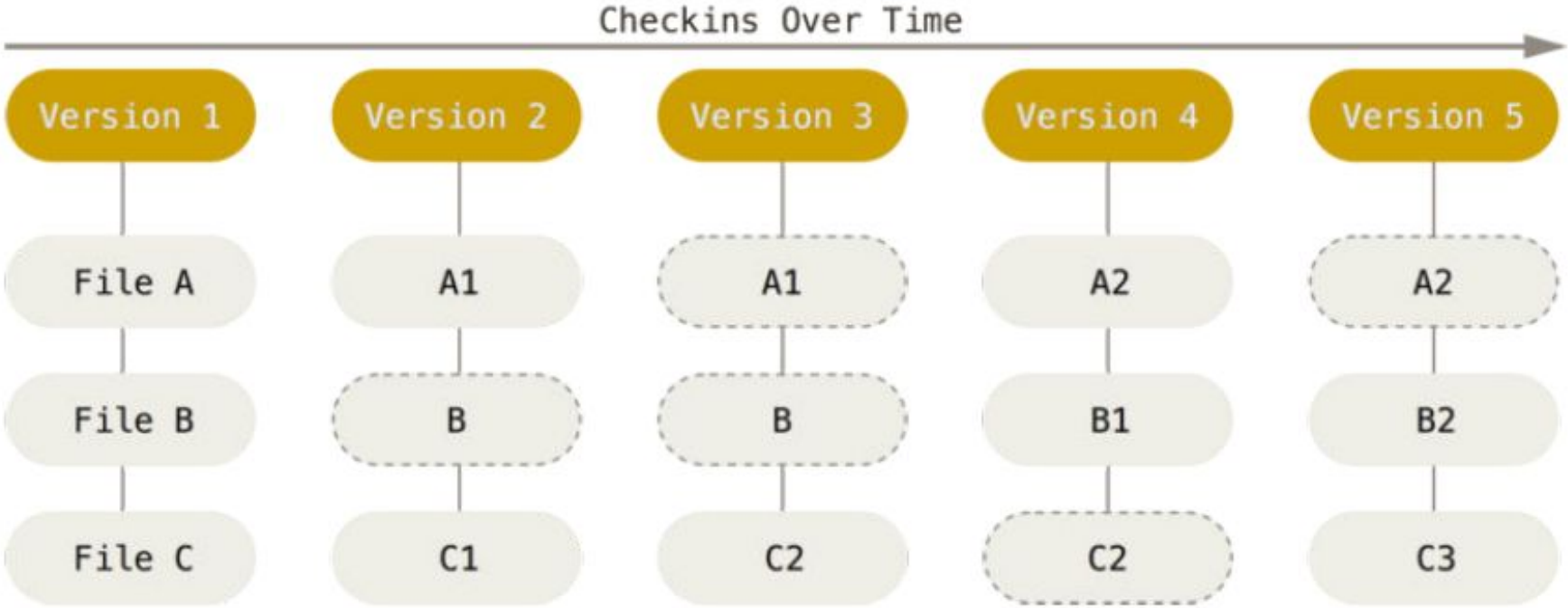


# ПОЧЕМУ ГИТ?

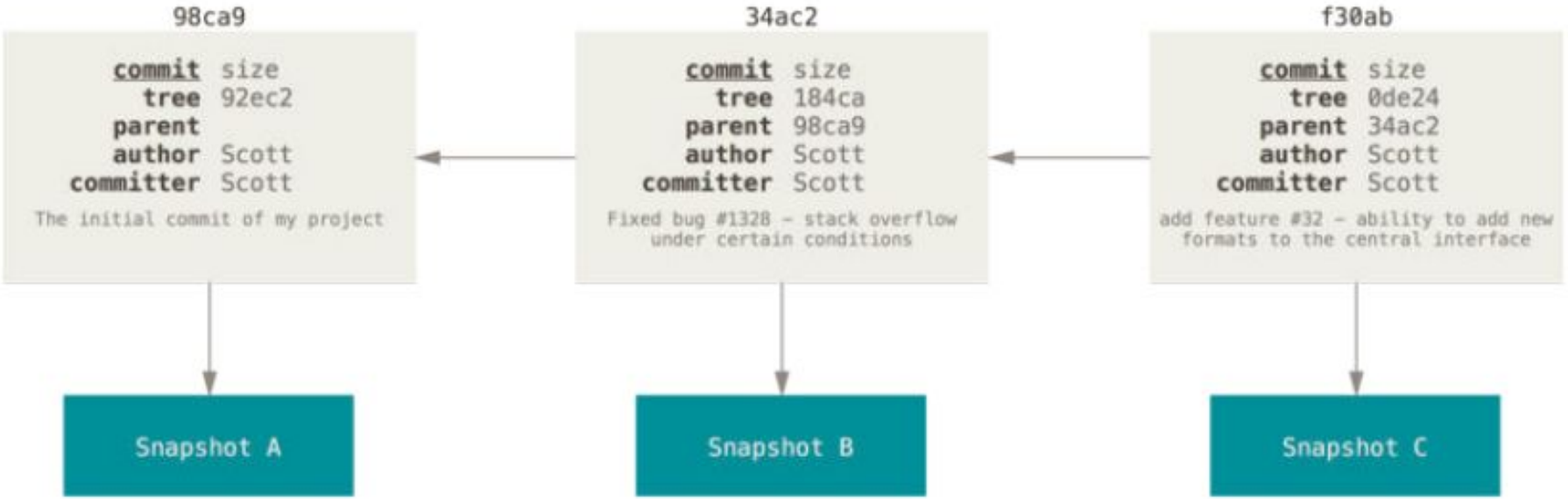
- Скорость
- Простота дизайна
- Полная распределенность
- Возможность нелинейной разработки (тысячи параллельных веток)
- Эффективно справляется с большими проектами, как разработка ядра Linux (скорость и объемы данных)


С момента своего рождения в 2005, Гит развивался в сторону простоты использования, но все также верен первоначальным идеалам. Он невероятно быстр, надежен и крайне эффективен на больших проектах благодаря легкой поддержке ветвления и нелинейной разработки.

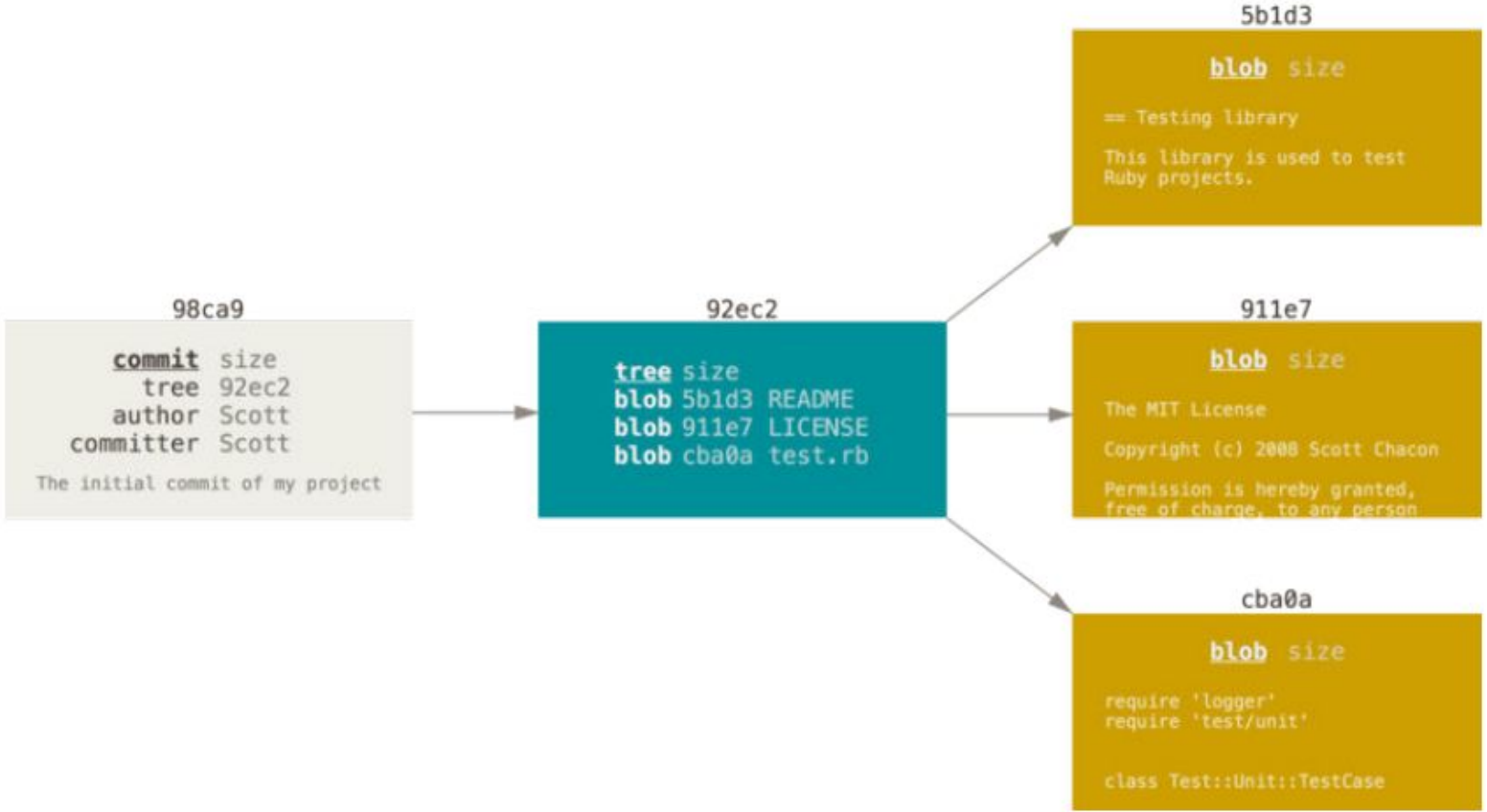
# ГИТ ХРАНИТ СЛЕПКИ ДАННЫХ (SNAPSHOT)



# ЦЕПЬ КОММИТОВ



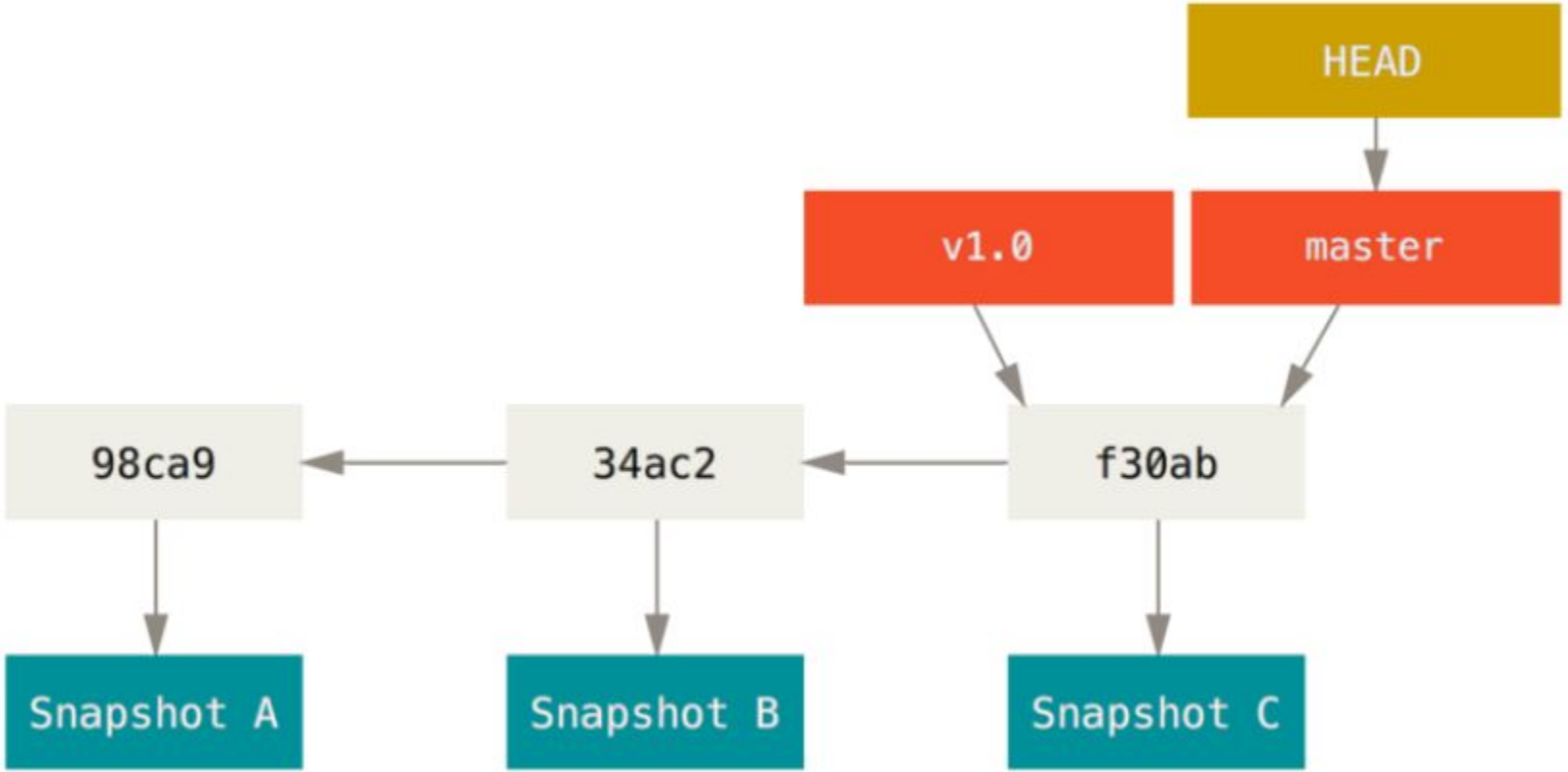
# СЛЕПОК = КОММИТ



# КЛЮЧЕВЫЕ УТВЕРЖДЕНИЯ

1. Когда вы клонируете репозиторий, вы получаете его полную копию
2. Репозиторий Гита представляет собой ориентированный граф коммитов
3. Каждый коммит хранит ссылку на своего предка
4. Ветвь или Тег всего лишь указатель на определенный коммит
5. Конкретный коммит существует до тех пор, пока существует хотя бы один указатель на него
5. У Гита есть сборщик мусора, который удаляет “забытые” коммиты

# УКАЗАТЕЛИ


HEAD
v1.0
master
98ca9
34ac2
f30ab
Snapshot A
Snapshot B
Snapshot C

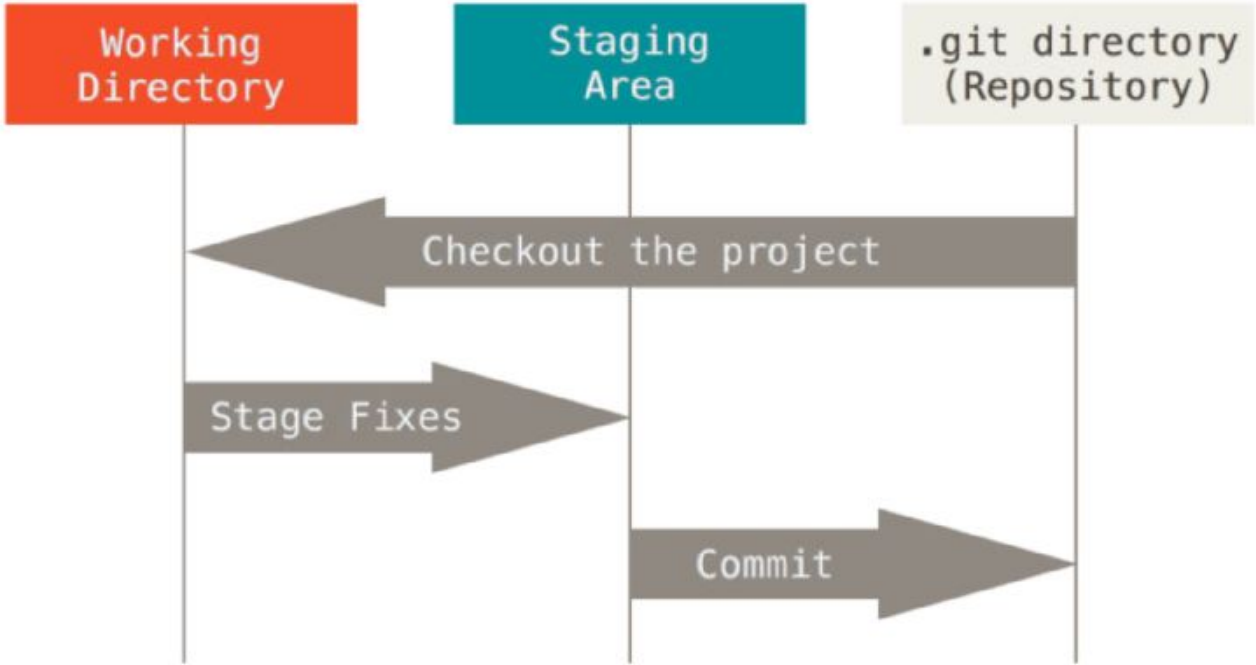
# КОММИТ

Обычно, чтобы сделать коммит вам
необходимо выполнить следующие команды:

```
git add readme.txt
git commit
```

Сначала вы добавляете некоторые файлы
в ваш будущий коммит (Гит называет это Staging Area),
а затем выполняете сам коммит.

# ТРИ СОСТОЯНИЯ



# ПРОМЕЖУТОЧНАЯ ОБЛАСТЬ (STAGING AREA)

# ИЗМЕНЕНИЕ СДЕЛАННОГО КОММИТА

Вы всегда можете изменить последний сделанный коммит
при помощи простой команды в консоли:

```
git commit --amend
```

Используя ее можно изменить комментарий к вашему коммиту,
а также добавить новые или убрать старые изменения.

Очень плохая идея менять коммиты,
которые уже были опубликованы в репозитории!

# РЕКОМЕНДАЦИИ И УДАЧНЫЕ ПРАКТИКИ

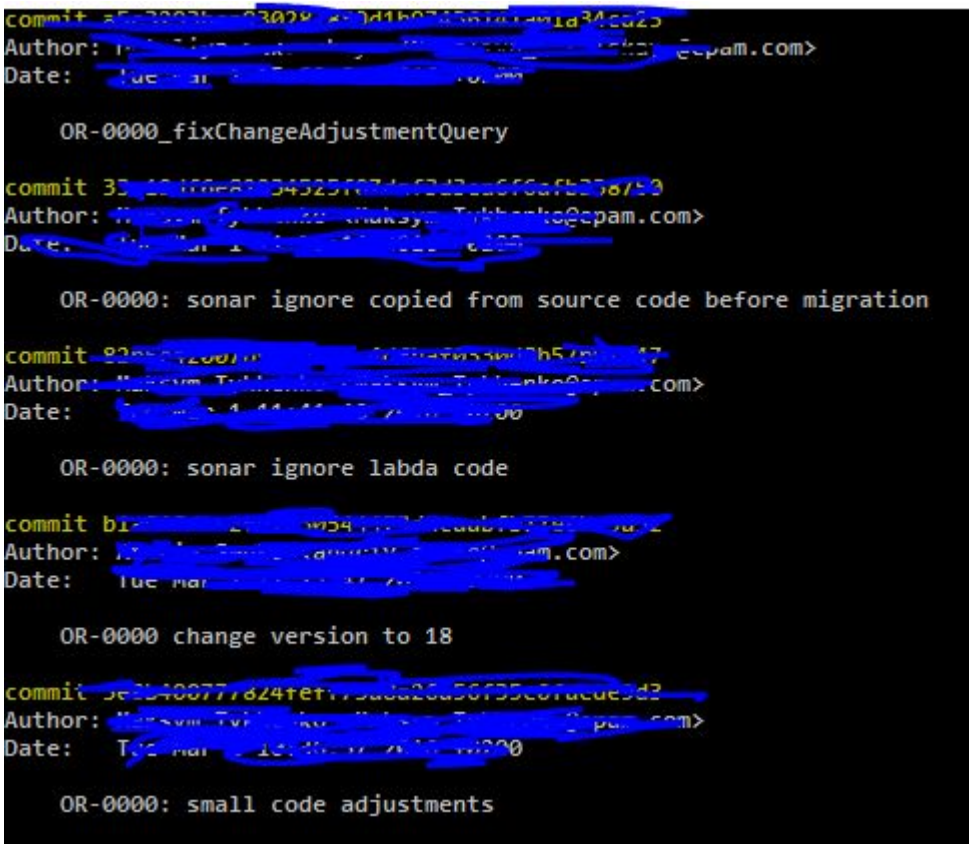
1. Коммиты должны быть атомарны. Добавляйте в ваш коммит изменения близкие по смыслу.
2. Коммиты должны содержать только работающий код.
3. Не коммитьте в нерабочий код (если только ваш код не содержит исправления).
4. Пишите подробные комментарии к коммитам. Они должны описывать проделанные изменения.
5. Не храните в репозитории то, что можно получить из исходных файлов (скомпилированные классы, сгенерированные отчеты и т.п.).

# РЕКОМЕНДАЦИИ И УДАЧНЫЕ ПРАКТИКИ

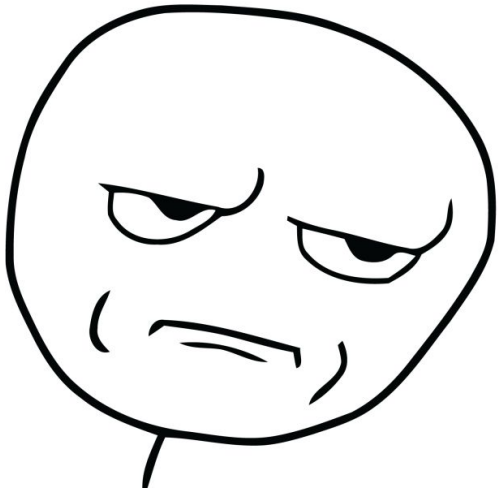
7. Не храните в репозитории конфигурационные файлы, которые зависят от локального окружения или то, что не является неотъемлемой частью проекта (конфигурации IDE, и т.п.).
8. Не добавляйте в репозиторий большие бинарные файлы.
9. Добавляйте, удаляйте, перемещайте или переименовывайте файлы в отдельном коммите.

# КОММЕНТАРИИ К КОММИТАМ

# КАК ДЕЛАЕМ ЧАСТО МЫ



# РАЗВЕ НЕ ВСЕ ТАК ДЕЛАЮТ?



# ПРИМЕР ИЗ РЕПОЗИТОРИЯ ГИТА

```
commit fac908389d4571ab5004872a54e50349272b63fc
Author: Jeff King <peff@peff.net>
Date:   Wed Dec 10 10:42:10 2014 -0500

        commit: loosen ident checks when generating template

        When we generate the commit-message template, we try to
        report an author or committer ident that will be of interest
        to the user: an author that does not match the committer, or
        a committer that was auto-configured.

        <a lot of text here>

        We know that the author and committer strings we are parsing
        have been generated by us earlier in the program, and
        therefore they must be parseable. We could just call
        split_ident_line without even checking its return value,
        knowing that it will put _something_ in the name/mail
        fields. Of course, to protect ourselves against future
        changes to the code, it makes sense to turn this into an
        assert, so we are not surprised if our assumption fails.

        Signed-off-by: Jeff King <peff@peff.net>
        Signed-off-by: Junio C Hamano <gitster@pobox.com>
```

# ЕЩЕ НЕСКОЛЬКО ПРИМЕРОВ

```
commit c0e0ed6efe497902a2e2ce5fb1586577a27eea1b
Author: Christian Hesse <mail@eworm.de>
Date:   Fri Dec 12 09:50:13 2014 +0100

        tests: skip RFC1991 tests for gnupg 2.1

        GnuPG >= 2.1.0 no longer supports RFC1991, so skip these tests.

        Signed-off-by: Christian Hesse <mail@eworm.de>
        Signed-off-by: Junio C Hamano <gitster@pobox.com>
```

```
commit b41a36e635803f1dc011007e836ae244f9ae04c1
Author: Christian Hesse <mail@eworm.de>
Date:   Fri Dec 12 09:50:12 2014 +0100

        tests: create gpg homedir on the fly

        GnuPG 2.1 homedir looks different, so just create it on the fly by
        importing needed private and public keys and ownertrust.

        This solves an issue with gnupg 2.1 running interactive pinentry
        when old secret key is present.

        Signed-off-by: Christian Hesse <mail@eworm.de>
        Signed-off-by: Junio C Hamano <gitster@pobox.com>
```

# И ГДЕ ЖЕ ЗОЛОТАЯ СЕРЕДИНА?

# ЗОЛОТАЯ СЕРЕДИНА

```
STORY-2015: Create Users page. Refactor UserService.

  - simplify hardToUnderstandFoo()
  - rename badNameFoo() -> goodNameFoo()
  - extract someName class field
  - remove duplicated stateField
  - clean code from foo1()private method call.
    It was triggered twice from foo1() and its parent method.
  - remove oldMethod() because of 0 usages.


STORY-2015: Create Users page. Add getUsers() in UserService.

  - add GET_ALL_USERS query to User entity

  - add getUsers() method in UserDao

  - add getUsers() method in UserService
```

1. Краткий заголовок. Можно указать номер задачи в трекере.
2. После заголовка оставьте одну пустую строку.
3. Перечислите изменения в коде и как они влияют на поведение используя императив (“add test” вместо “I’ve added test” или “adding test”).
4. Дополнительно, можете указать вашу мотивацию. Гораздо важнее знать не что было изменено, а почему те или иные изменения были сделаны.

# ПОЧЕМУ ИМЕННО ТАК?

*Branches: OR-8066 → develop-valencia-next*

Assignee: [redacted] → [redacted]

- changed facility-location-select-range that it can return empty location code on F12 key pressed
- added check for empty location code in isLocationCodeNew()
- rewritten checkForFacilituLocation() function: added recursion calls for differnt cases of "?" position; recursion was added because it needs that facilityLocation pop-ups show consequantly and chosen code corespond field in which it was called
- added errTemplate variable which used when F12 pressed on Facility location pop-up for generating error on retail location pop-up

Данный формат предназначен для автоматизированной рассылки электронных писем на основании комментариев к коммитам. Например в “GitLab”.

# Q&A

# ПОЛЕЗНЫЕ ССЫЛКИ И РЕСУРСЫ

Основные ресурсы:

1. Книга "Pro Git"
2. Официальная документация
3. Git magic

Самоучители:

1. Неплохой пошаговый самоучитель
2. Become a Git guru

Дополнительно:

1. Git concepts simplified

Ссылки по теме:

1. Git best practices
2. Atomic commit convention
3. Эксперименты над blob и tree
4. Note about commit messages