Системы контроля версий (VCS)

Курс «Технология программирования»

Кафедра управления и информатики НИУ «МЭИ»

Осень 2016 г.

Мотивирующие примеры

- Бригада выполняет задание на семестр совместно.
  - Нужно: обмениваться кодом и совмещать наработки.
  - Вручную: передавать файлы, копировать фрагменты.
  - СКВ: общее хранилище, автоматическое совмещение.
- Программа работает, но нужно её изменить.
  - Нужно: вернуться к работающей версии.
  - Вручную: резервные копии (каждый день много раз?).
  - СКВ: есть история версий, можно перейти к любой.
- Файл изменен, но непонятно, как именно.
  - Нужно: найти все изменения.
  - Вручную: внимательно читать и сравнивать тексты.
  - СКВ: вычисление разницы между файлами и версиями.

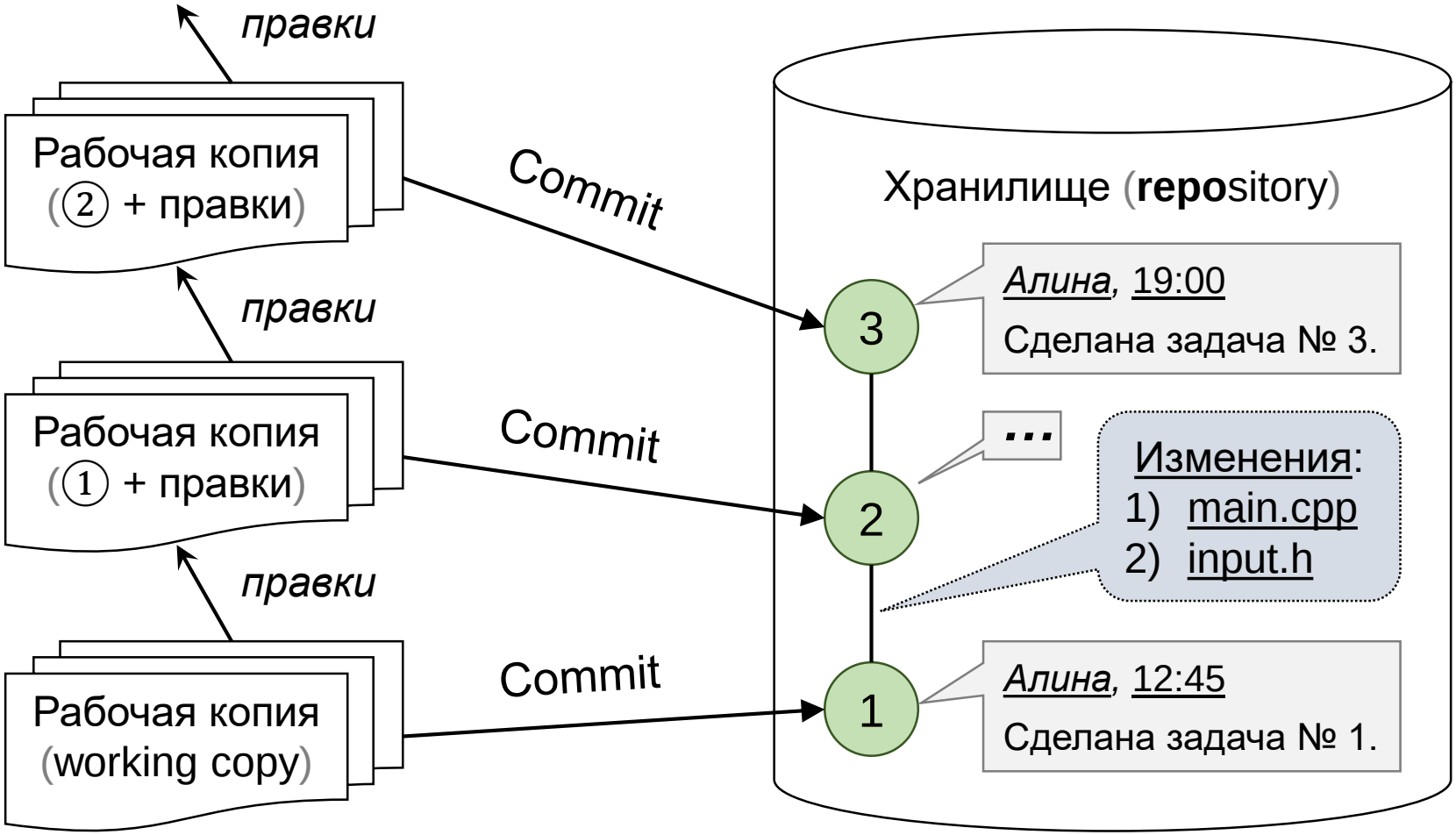
Особенности совместной разработки программ

- Автоматическое ведение истории любых изменений.
- Автоматическая синхронизация.
- Одновременное редактирование в реальном времени.

Разработка программ

- Раздельное редактирование.
- Явное создание пунктов истории:
  - составление набора сохраняемых изменений;
  - комментарии.
- Явное совмещение наборов изменений.

Единоличная работа с VCS



Основные понятия VCS

- **Хранилище (repository),** или **репозитарий,** —

  место хранения всех версий и служебной информации.

- **Версия (revision),** или **ревизия,** —

  состояние всех файлов на определенный момент времени, сохраненное в репозитарии, с дополнительной информацией

- **Commit** («[трудовой] вклад», не переводится) —

  синоним версии; процесс создания новой версии.

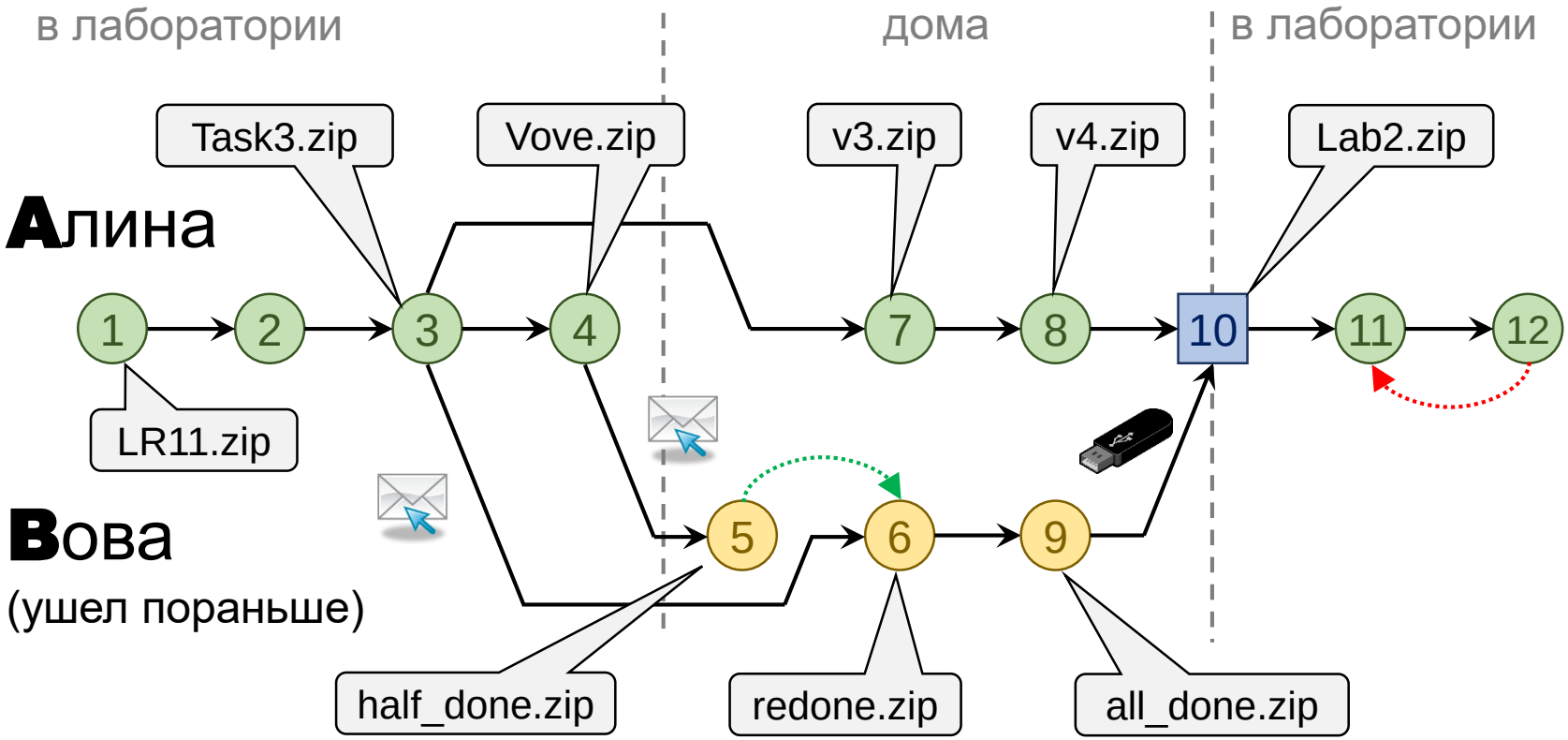
- **Рабочая копия (working copy) —**

  текущее состояние файлов проекта, основанное на версии, загруженной из хранилища (обычно на последней).

Добавление и игнорирование

- Чтобы СКВ учла изменения в файлах и новые файлы, их нужно добавить (add) в набор изменений (index).
  1. Перед commit-ом указывается, какие изменения учесть.
  2. В commit входят только указанные изменения.
- Игнориуемые файлы СКВ не учитывает никогда.
  - «Не учитывает»:
    - не сообщает об изменениях (удобство);
    - не позволяет добавить (защита от ошибок);
  - Список хранится в файле .gitignore в корне хранилища.
  - Пример: исполняемые файлы *.exe.

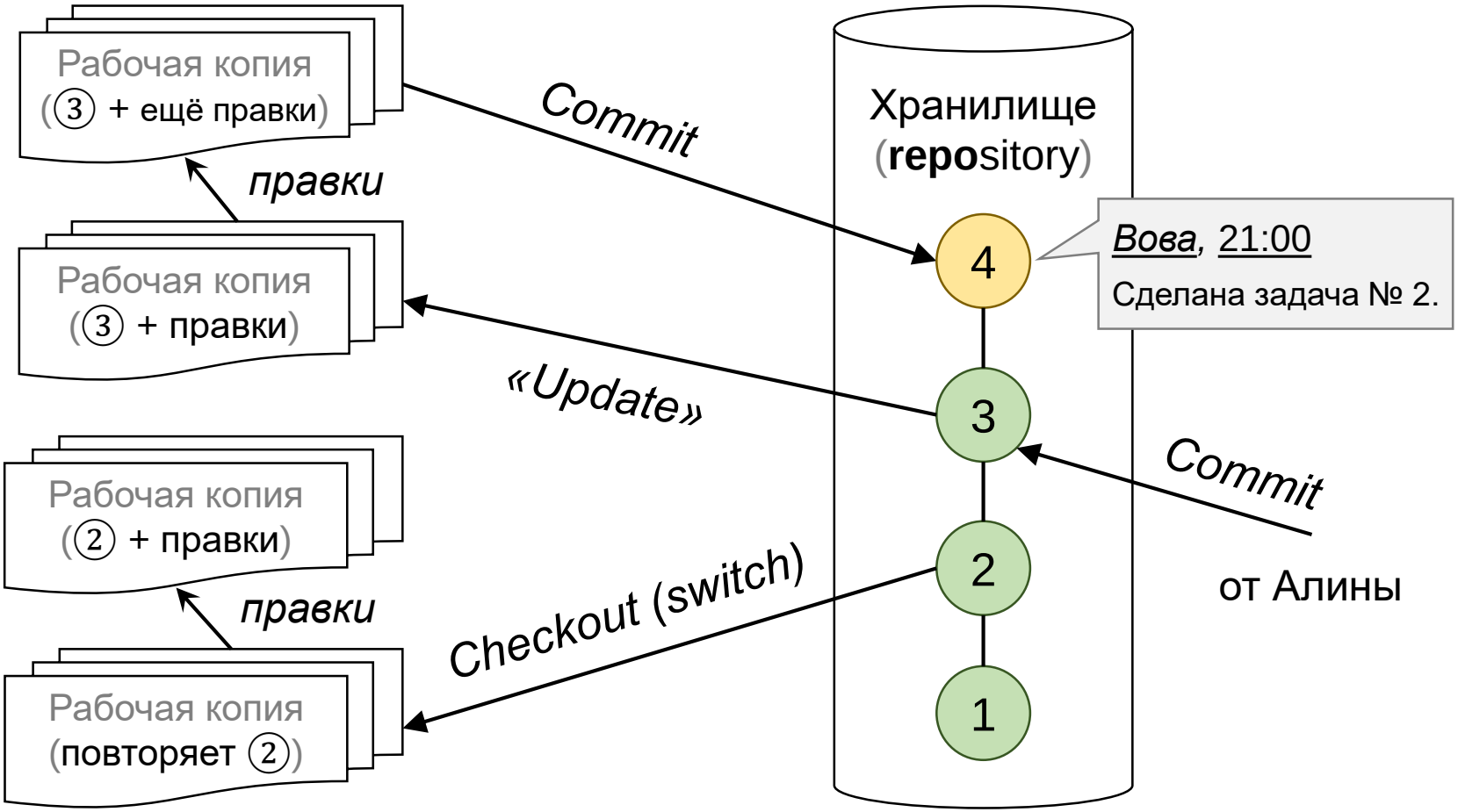
Проблемы совместной разработки



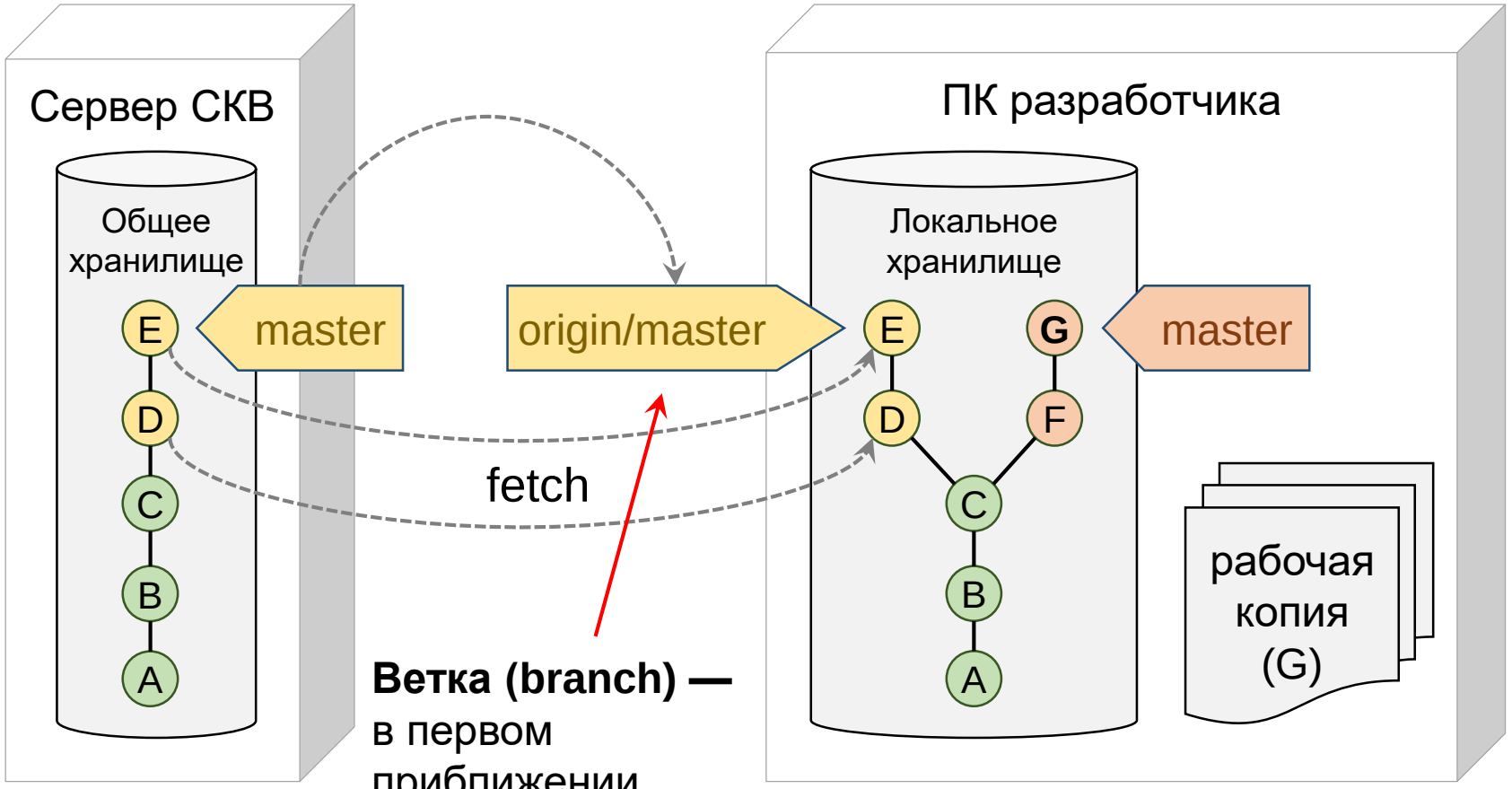
- Версии должны быть доступны всем.
- Нужно совмещать свои наработки с чужими.
- История должна оставаться понятной.

В западном криптоанализе стороны называют Alice (A) и Bob (B), но мы же в России и не криптоаналитики.

Работа с общим хранилищем

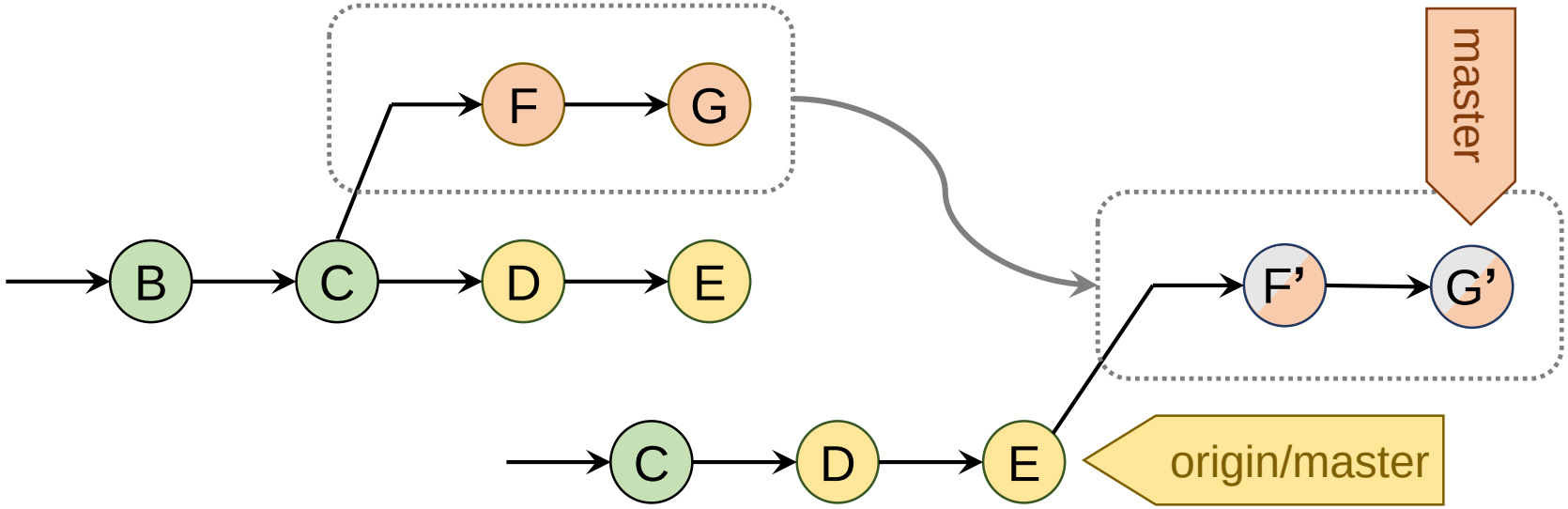


Обновление: проблема



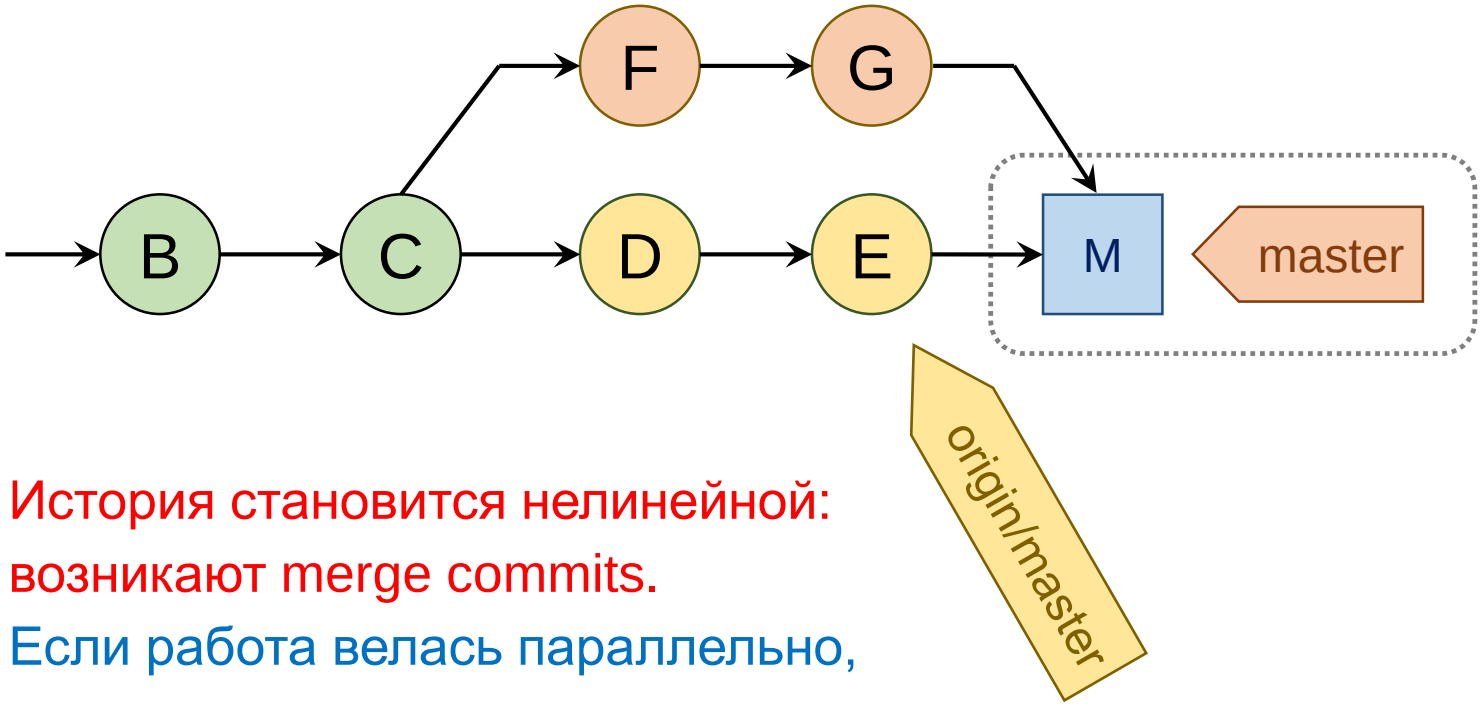
Ветка (branch) — в первом приближении есть начало цепочки коммитов, указатель на самой новый коммит в ней.

Обновление: решение (I)



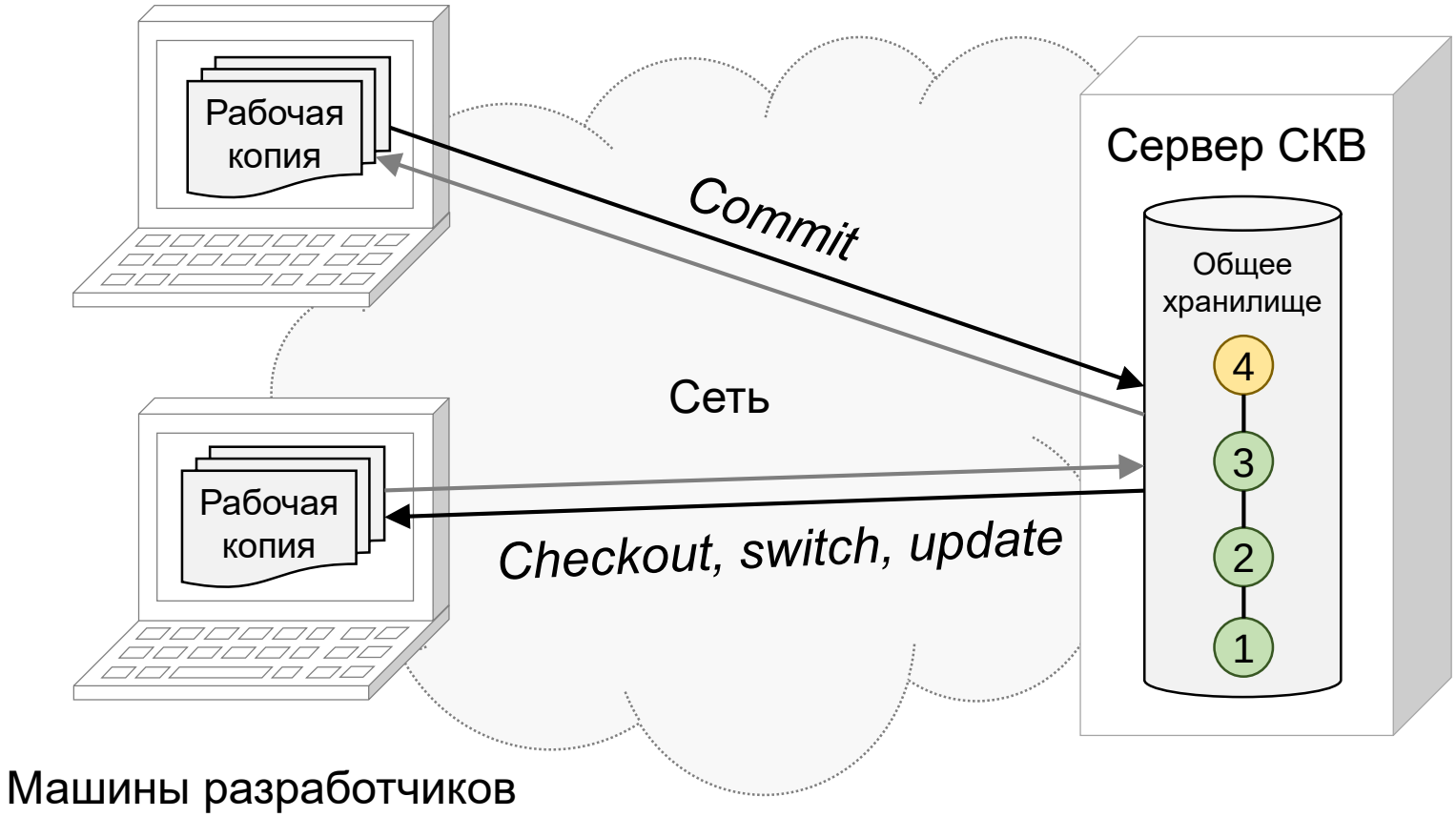
- История остается линейной, ветвление исчезает.
- История искажается: изменяются коммиты и их порядок.
- Возможны ошибки программиста при переносе коммитов.

Обновление: решение (II)

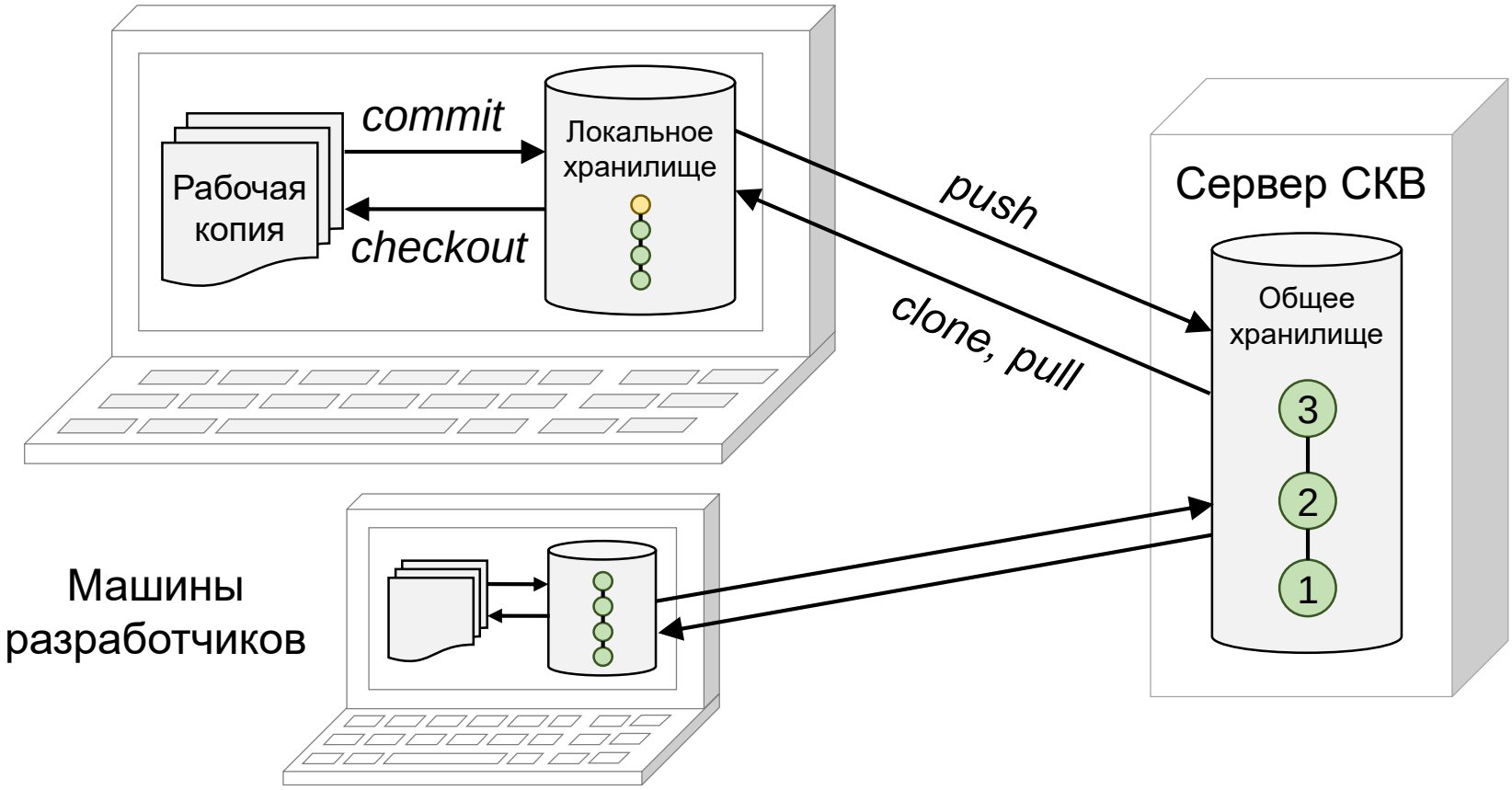


- История становится нелинейной: возникают merge commits.
- Если работа велась параллельно, это остается видно.
- Совмещаются только последние версии — меньше риск ошибиться программисту.
- Все версии неприкосновенны.

Общее хранилище: централизованная VCS



Отдельные хранилища: распределенная VCS (DVCS)



Виды систем контроля версий

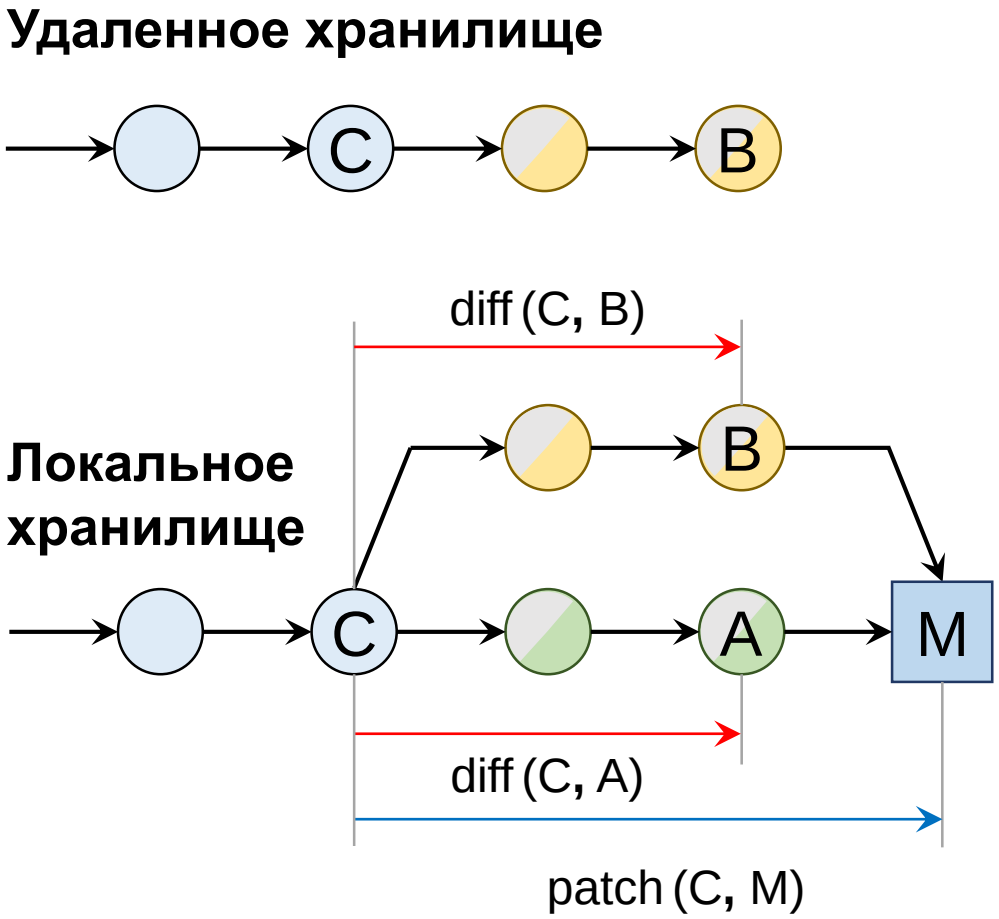
Централизованные

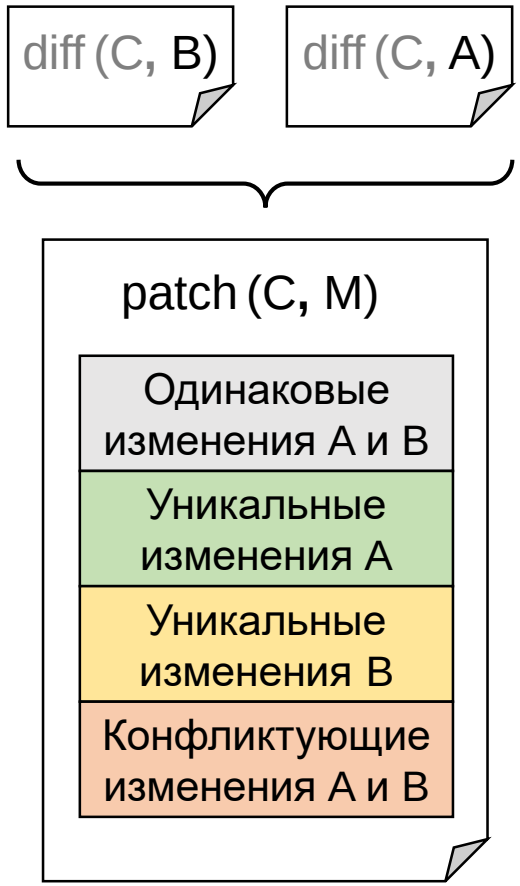
- Простота использования.
- Вся история — всегда в едином общем хранилище.
- Нужно подключение к сети.
- Резервное копирование нужно только одному хранилищу.
- Удобство разделения прав доступа к хранилищу.
- Почти все изменения навсегда попадают в общее хранилище.

Распределенные

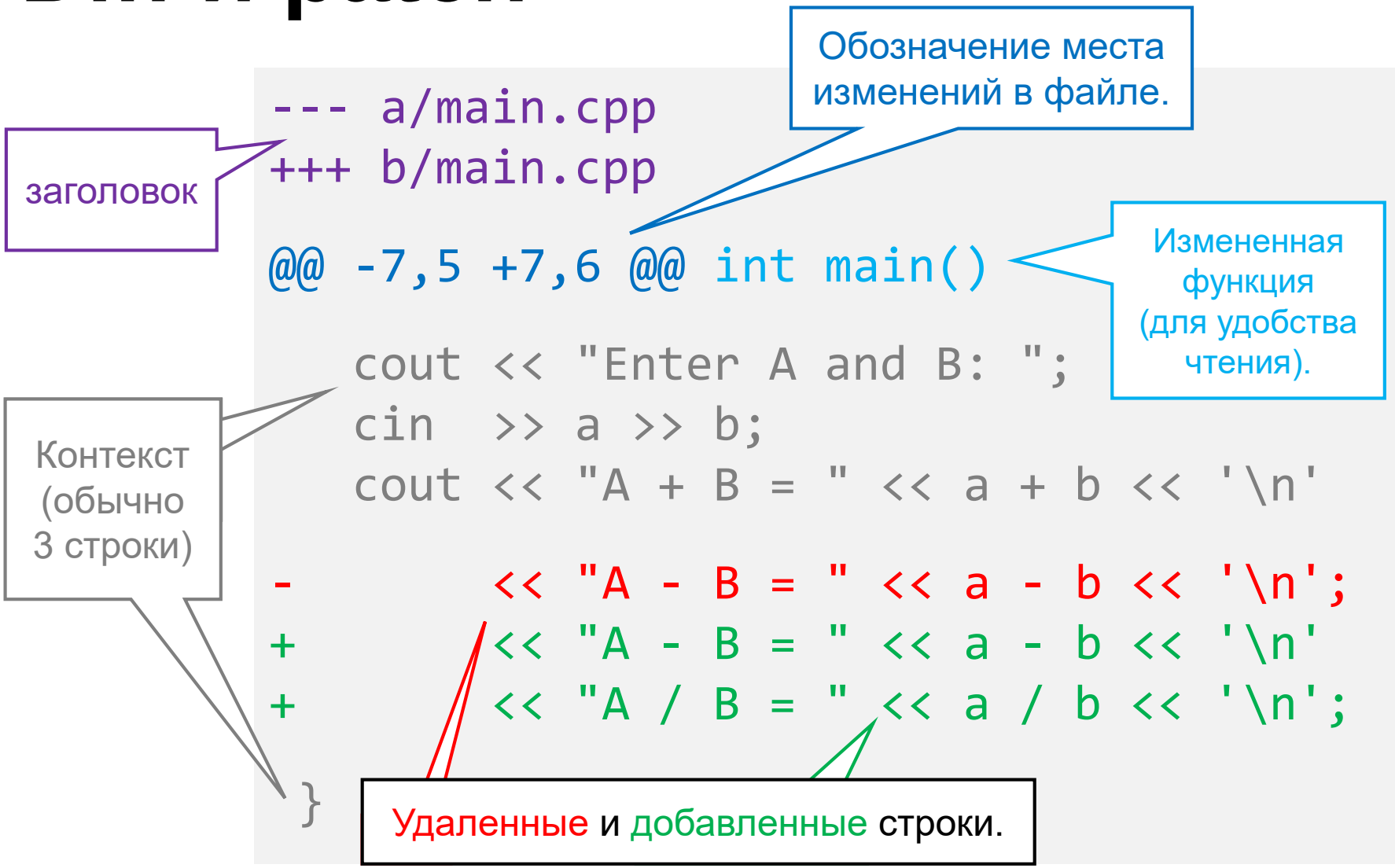
- Двухфазный commit:
  1) запись в локальную историю;
  2) пересылка изменений другим.
- Подключение к сети не нужно.
- Локальные хранилища могут служить резервными копиями.
- Локальное хранилище контролирует его владелец,
  - но общее — администратор.
- **Возможна правка локальной истории перед отправкой на сервер.**

Слияние версий (merge)





Diff и patch



Основные понятия VCS

- **Слияние (merge) —**

  объединение двух версий в единую;
  слияние ветвей — объединение их последних версий.

- **Конфликт (conflict) —**

  ситуация, когда VCS не может автоматически слить внесённые изменения (т. е. когда были исправлены одни и те же места в файлах).

- **Разность (difference, diff) —**

  построчные различия между файлами (разных версий).

- **Заплатка (patch), патч —**

  файл-инструкция, какие правки нужно внести (по сути, это **diff**).

Основные понятия DVCS

- **Загрузка изменений (fetch) —**

  загрузка наборов изменений (commit-ов) из удаленного хранилища.

- **Обновление, «подтягивание» (pull) —**

  загрузка изменений и немедленное слияние с локальным хранилищем (pull = fetch + merge).

- **Отправка изменений (push) —**

  передача наборов изменений в удаленное хранилище с немедленным слиянием.

  - Если при слиянии возникает конфликт, происходит ошибка.
  - Возможна, но не рекомендуется, **force push** — принудительная перезапись удаленной истории.

Чистая рабочая копия

- **Clean** working copy, или **pristine** state.
- Без измененных файлов, занесенных в СКВ.
  - Возможно, с новыми файлами.
- Необходима:
  - для любого слияния:
    - merge, rebase;
    - pull = fetch + merge;
    - cherry-pick (перенос одиночных commit-ов);
    - в централизованных VCS — также для обновления.
  - для переключения ветвей.

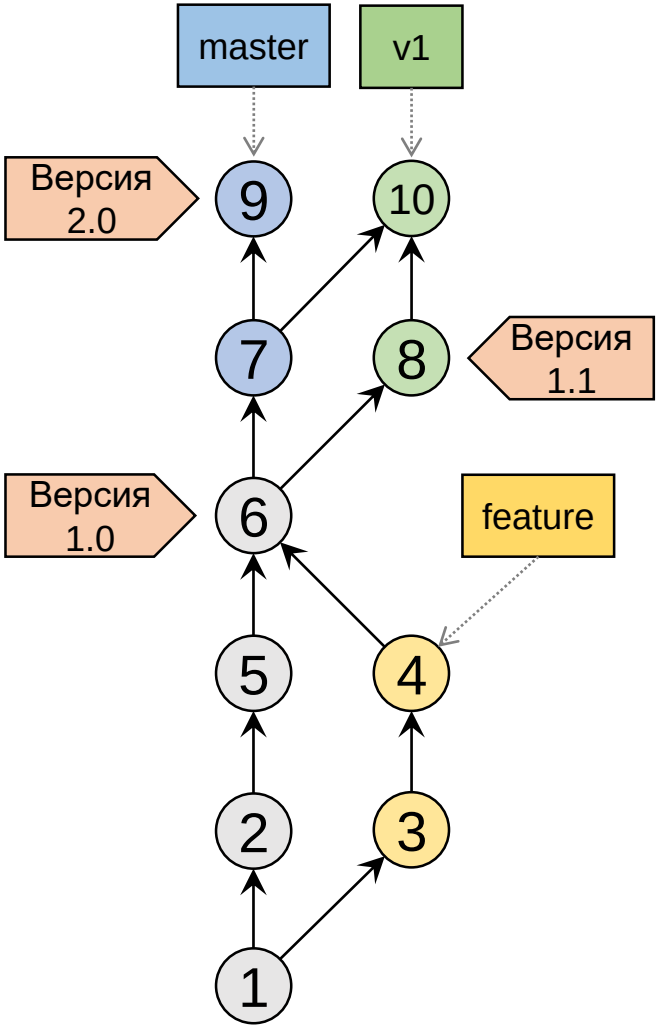
Ветвления и метки

- **Ветвление («ветка», branch) —**

  один из параллельных участков истории в одном хранилище, исходящих из одной версии (точки ветвления).

  - Обычно есть главная ветка (master), или ствол (trunk).
  - Между ветками, то есть их концами, возможно слияние.

- **Метка (tag) —** отмеченная версия. Отличие от ветки в том, что тэг не перемещается при добавлении коммитов.



Модели ветвлений (branching models), или рабочие процессы (workflows)

Договоренность о ветках внутри команды.

- Кем, когда и зачем создаются?
  a) по одной каждым разработчиком для своих задач
  b) по ветке на задачу,
  c) по ветке на группу связанных задач…
- Что содержат?
  - ветка со стабильным кодом,
  - ветка с нововведениями…
- Кем, когда, куда и по каким критериям сливаются?
  - когда код в ветке протестирован, она сливается в master,
  - раз в неделю master сливается во все ветки…
- Как называются?

Модели ветвлений (branching models), или рабочие процессы (workflows)

- **Centralized:**

  участники просто синхронизируют хранилища через общее (работает в централизованных СКВ или с одной веткой).

- **Feature branches:**

  новые компоненты, задачи, исправления реализуются в отдельных ветках, а по окончании сливаются в главную.

- **Gitflow** = feature + release + maintenance branches
  - release branches как на предыдущем слайде;
  - maintenance branches для исправлений;
  - есть ветвь «для тестирования», «нестабильная» и др.

- **Custom**

Модели ветвлений открытых проектов

- **Forking**
  - Участники клонируют центральное хранилище и ведут в нем разработку по своему усмотрению.
  - Руководитель может слить часть изменений из хранилища участника в общее.
    - **Pull request (PR) —** просьба участника забрать у него изменения.
- **Developer branches**
  - Программисты работают в собственных ветвях, руководитель сливает ветви в общую.
  - От developer branch могут отходить feature branches.

Особые возможности Git

- **Stash** — скрытые commit-ы:
  - полезно при обновлении;
  - «буфер обмена» между ветвями.
- **Staging:**
  - детальный контроль содержимого очередного commit;
  - можно включить в commit часть файла (hunk).
- **Cherry-pick** —

  копирование commit-ов между ветвями.

Цели разработчика

- ✓ Восстанавливать состояние проекта на любой момент.
- ✓ Разрабатывать версии параллельно.
- ✓ Сравнивать и совмещать результаты разработчиков.
- ✓ Поддерживать общую версию хранилища.

Задачи и средства VCS

1. Ведение истории версий проекта:
   - журнал (log);
   - метки (tags);
   - ветвления (branches).
2. Работа с изменениями:
   - выявление (diff);
   - слияние (patch, merge).
3. Обеспечение совместной работы:
   - получение версии с сервера;
   - загрузка обновлений на сервер.

Некоторые VCS и их особенности

- **Subversion (SVN):**
  - одна из старейших, и потому все еще популярна;
  - централизованная.
- **Git:**
  - распределенная, популяризовала этот тип;
  - самая распространенная на сегодня (GitHub, BitBucket).
- **Mercurial (Hg):**
  - распределенная, похожа на Git, но отличается рядом аспектов;
  - широкие возможности по управлению хранилищами.
- **Perforce:**
  - Основана на Git, но требует центрального хранилища;
  - улучшенная работа с файлами не-кода (документы и т. п.);
  - ядро комплексной системы ведения проекта;
  - коммерческая лицензия.
- **CVS:**
  - морально устарела;
  - имела механизм locks: файл мог редактировать только один человек в момент времени.

Ресурсы к лекции

- Scott Chacon, Ben Straub. *Pro Git.*
- Joel Spolsky. *Hg Init: a Mercurial Tutorial.*
- Ben Collins-Sussman et al. *Version Control with Subversion.*
- Хостинги хранилищ:
  - GitHub:
    - самый популярный;
    - больше функций, но только Git;
    - бесплатно — только открытые хранилища.
  - BitBucket:
    - Git, Mercurial;
    - есть бесплатные закрытые хранилища;
    - я им пользуюсь и смогу подсказать :-).
- Все ссылки есть на странице курса.