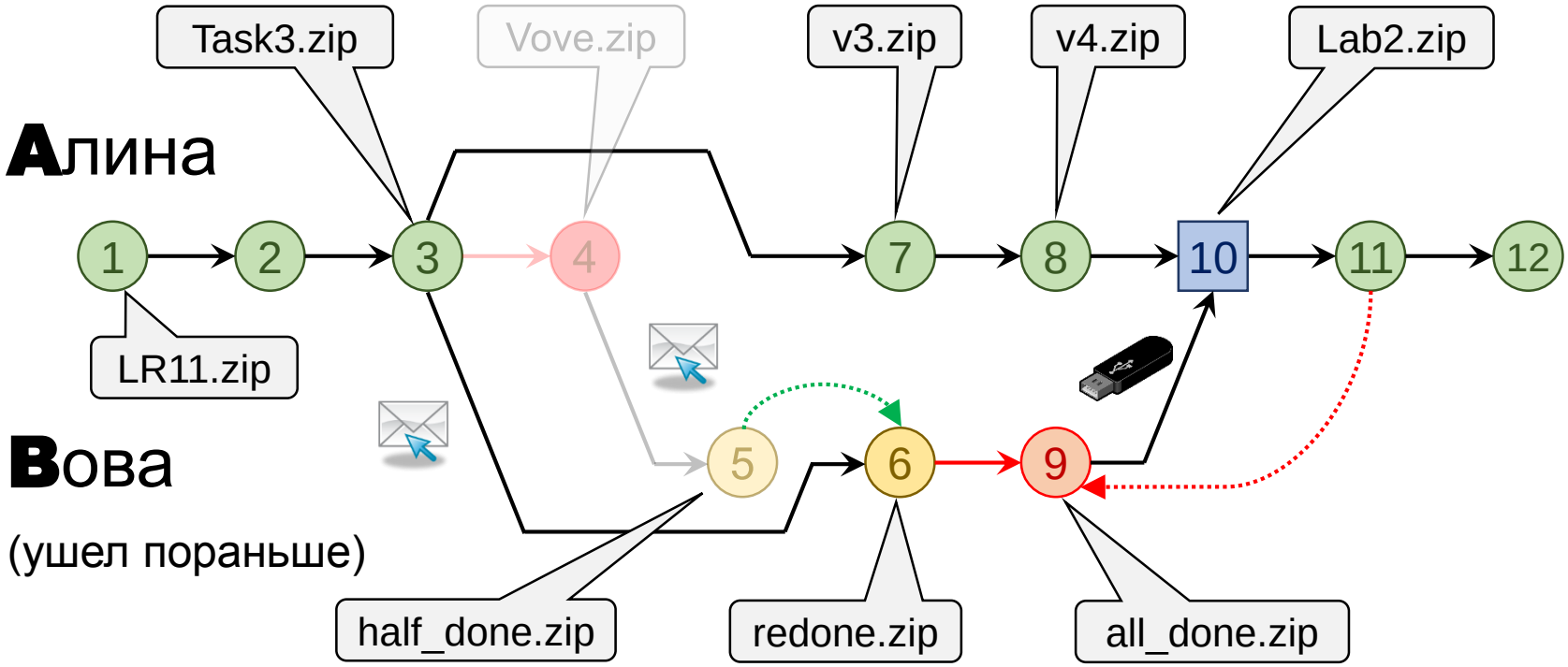
Системы контроля версий (VCS)

Курс «Технология программирования»

Кафедра управления и информатики НИУ «МЭИ»

Осень 2015 г.



1) Ведение истории:
   - сохранение состояния на заданный момент;
   - параллельная разработка нескольких версий.

2) Управление изменениями:
   - сравнение состояний;
   - перенос изменений;
   - слияние двух версий;
   - отмена изменений.

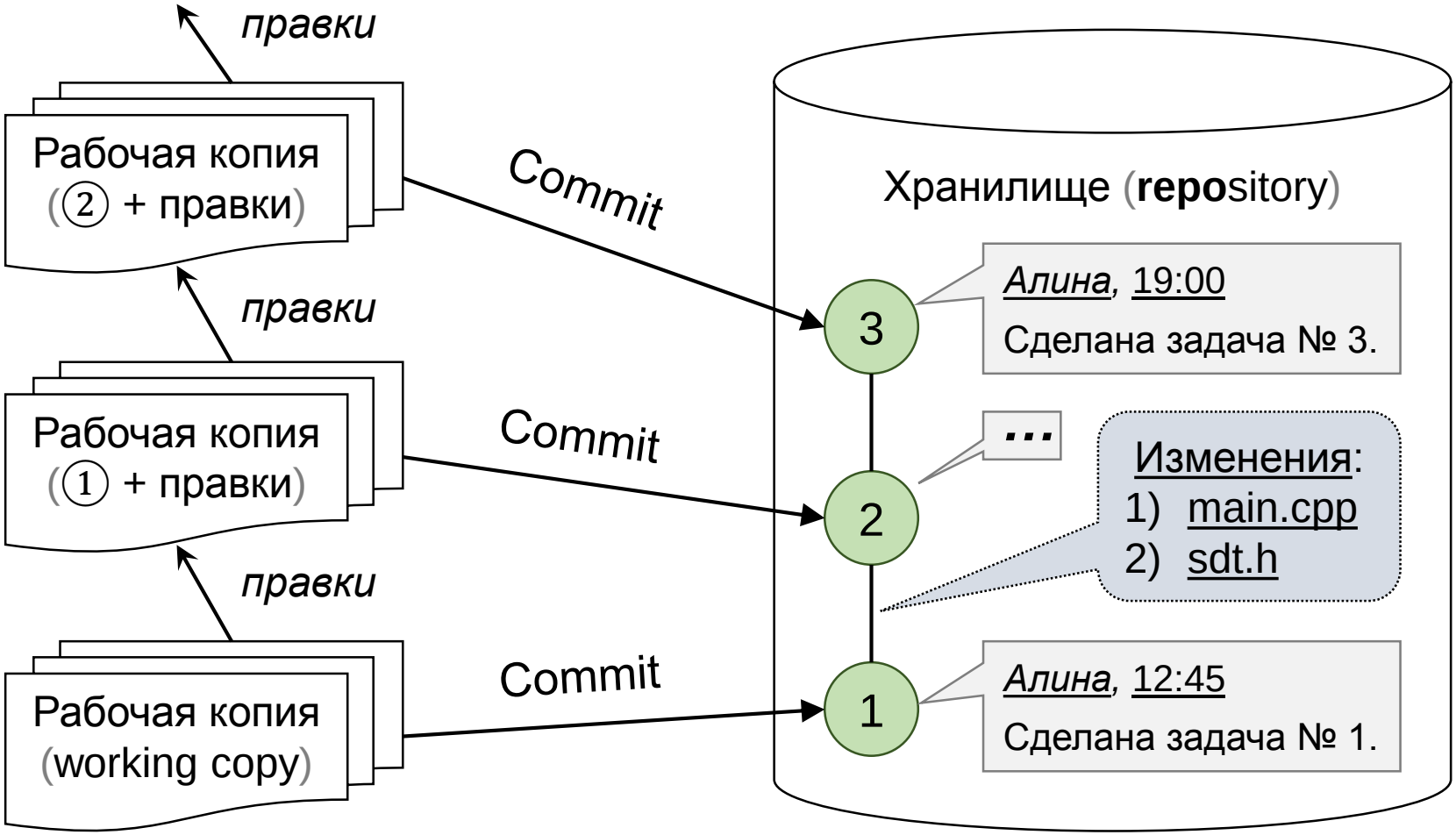
Ведение истории совместной разработки

- 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 файлов и их версий, служебной информации.

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

  состояние всего хранилища или отдельных файлов
  в момент времени («пункт истории»).

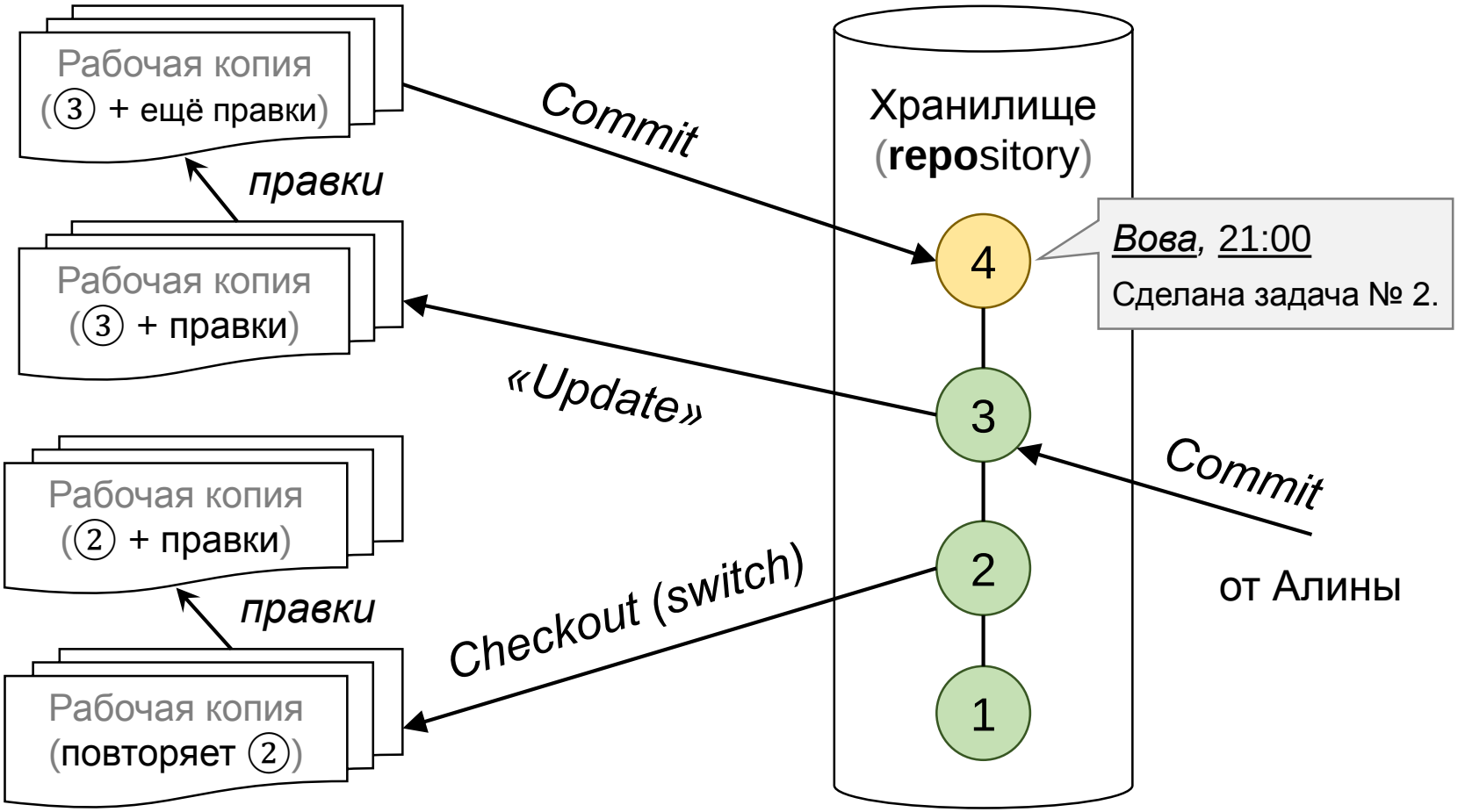
- **Commit** («[трудовой] вклад», не переводится) —

  процесс создания новой версии; иногда синоним версии.

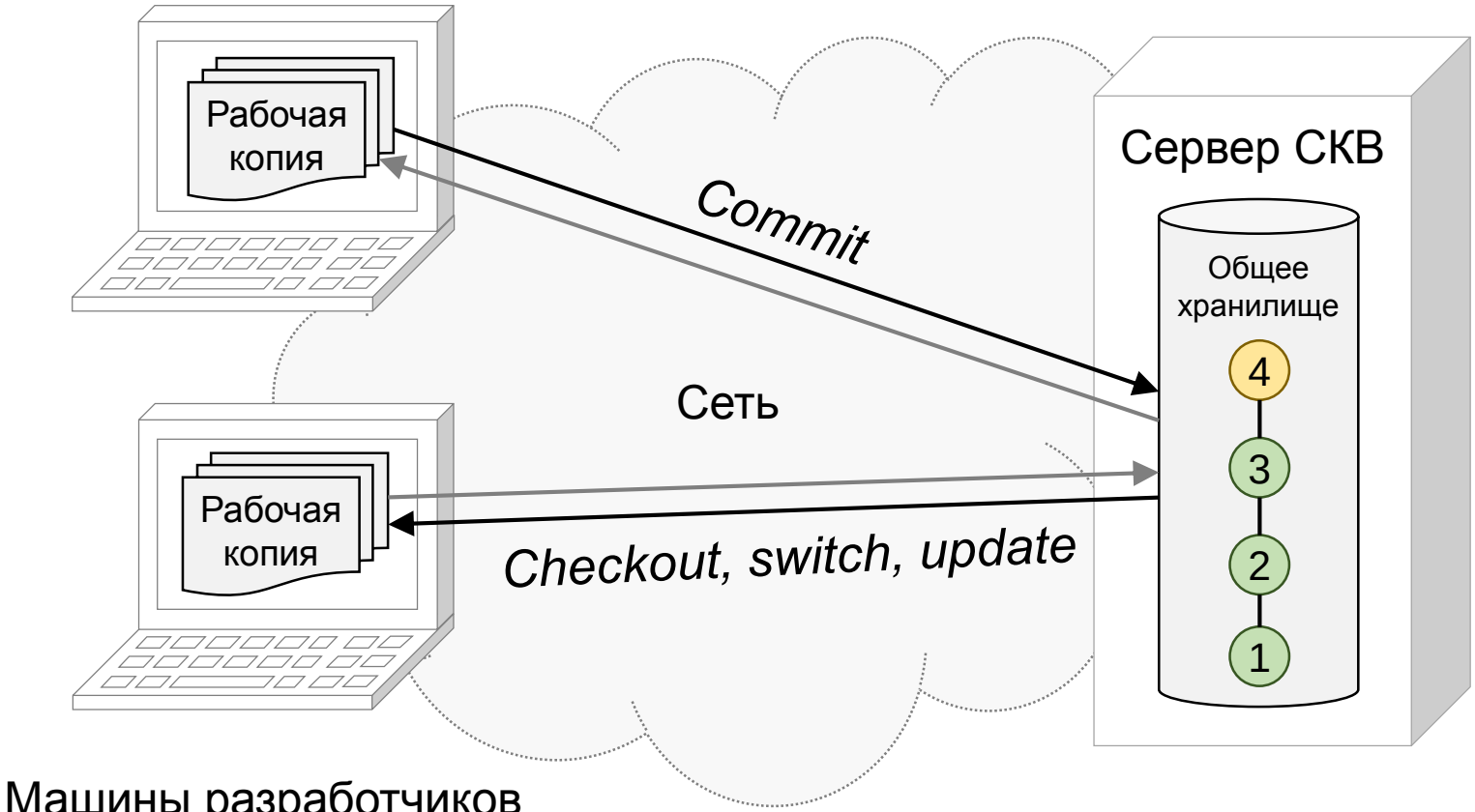
- **Рабочая копия (working copy) —**

  текущее состояние файлов проекта (любой версии),
  полученных из хранилища и, возможно, измененных.

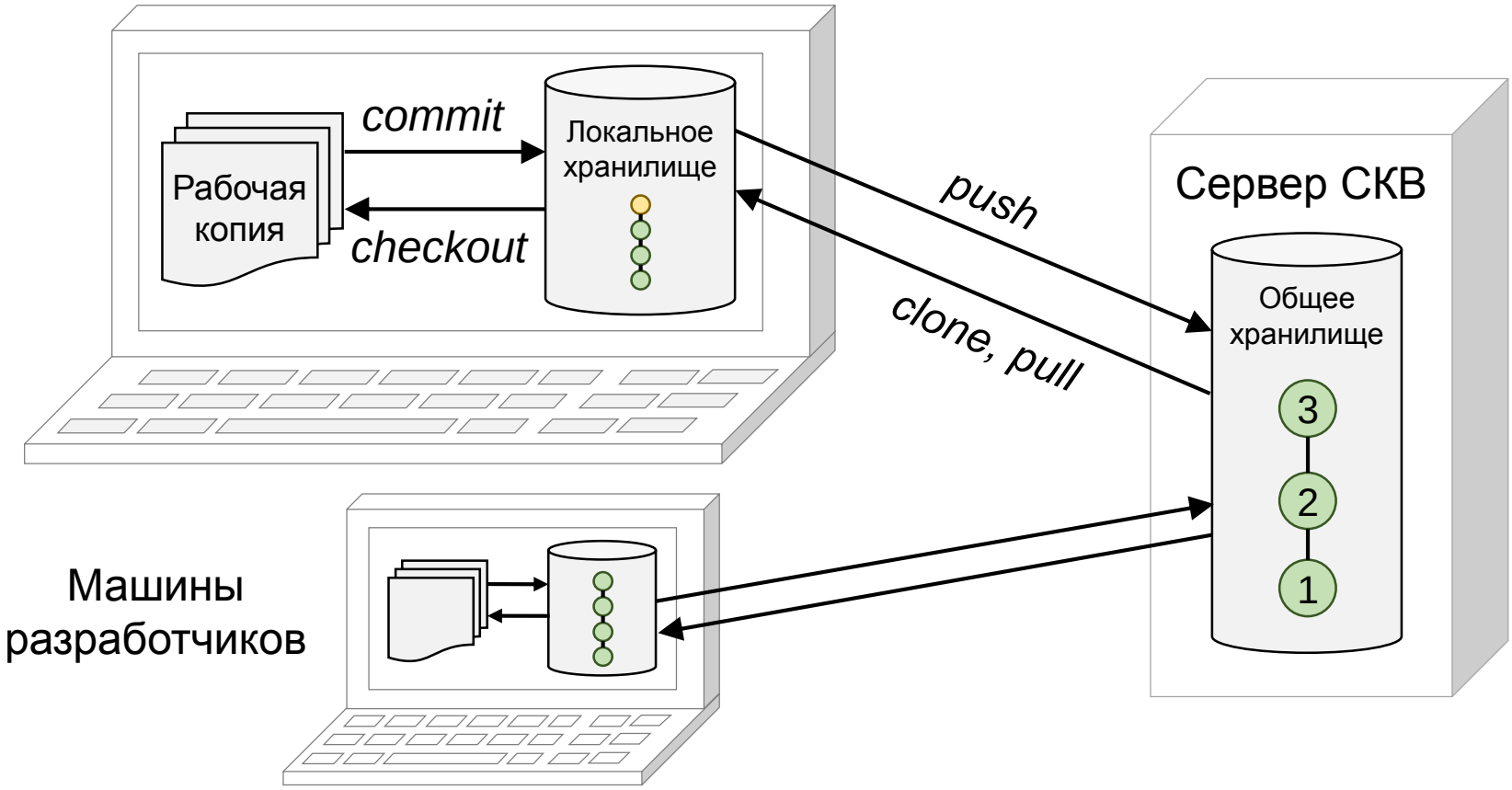
Работа с общим хранилищем



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.**

Добавление и игнорирование

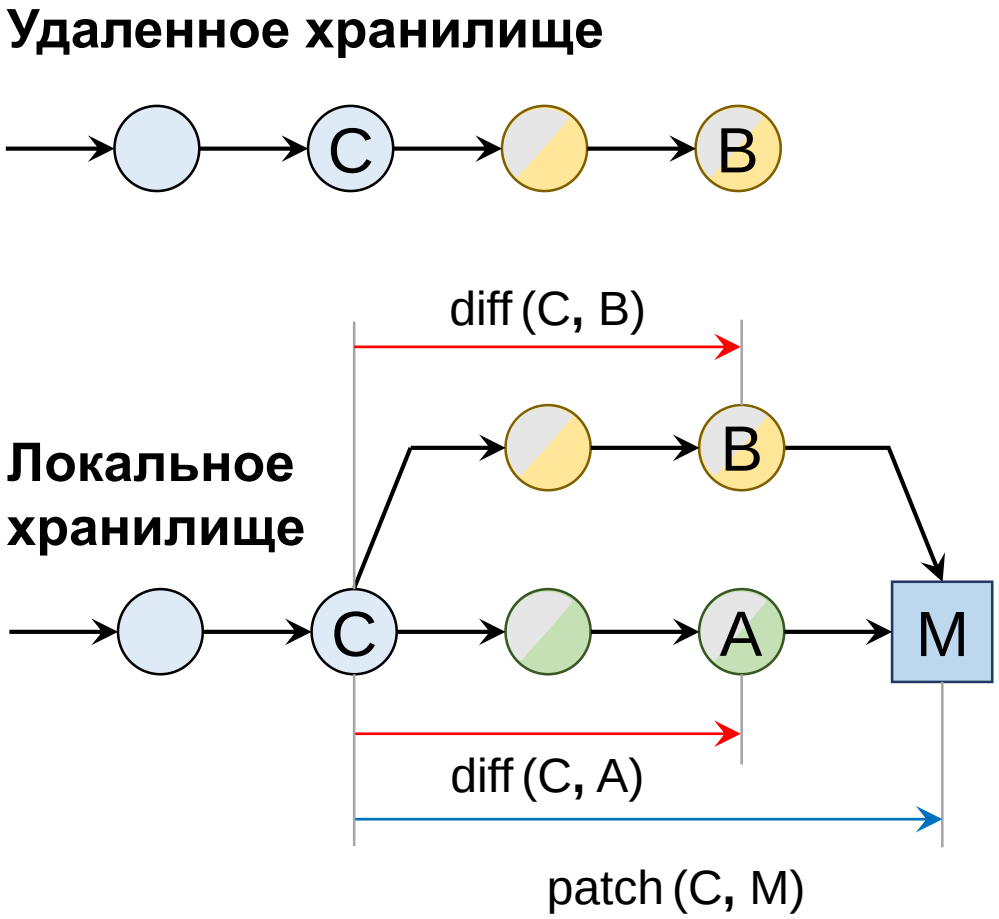
- Чтобы СКВ учла изменения в файлах и новые файлы, их нужно добавить (add) в набор изменений (index).
  1. Перед commit-ом указывается, какие изменения учесть.
  2. В commit входят только указанные изменения.
- Игнориуемые файлы СКВ не учитывает никогда.
  - «Не учитывает»:
    - не сообщает об изменениях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

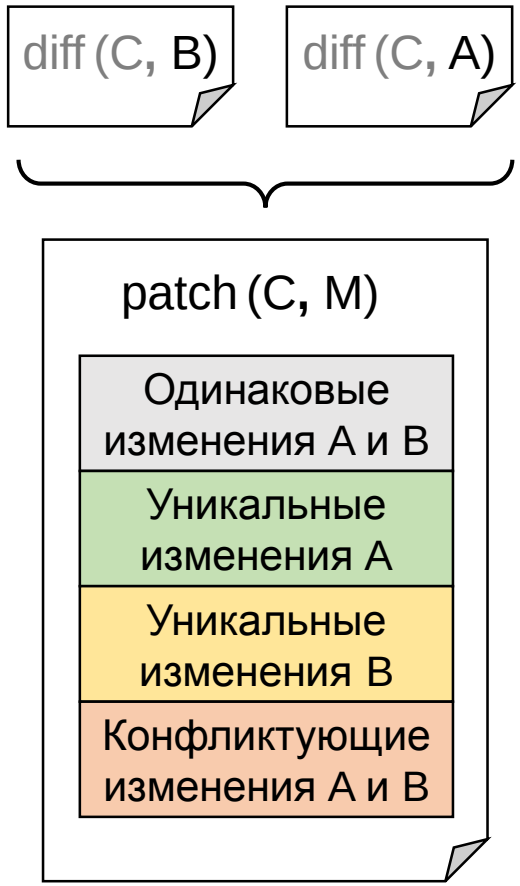
Обновление

Последний локальный commit — A, последний в удаленном **(remote)** хранилище — B.

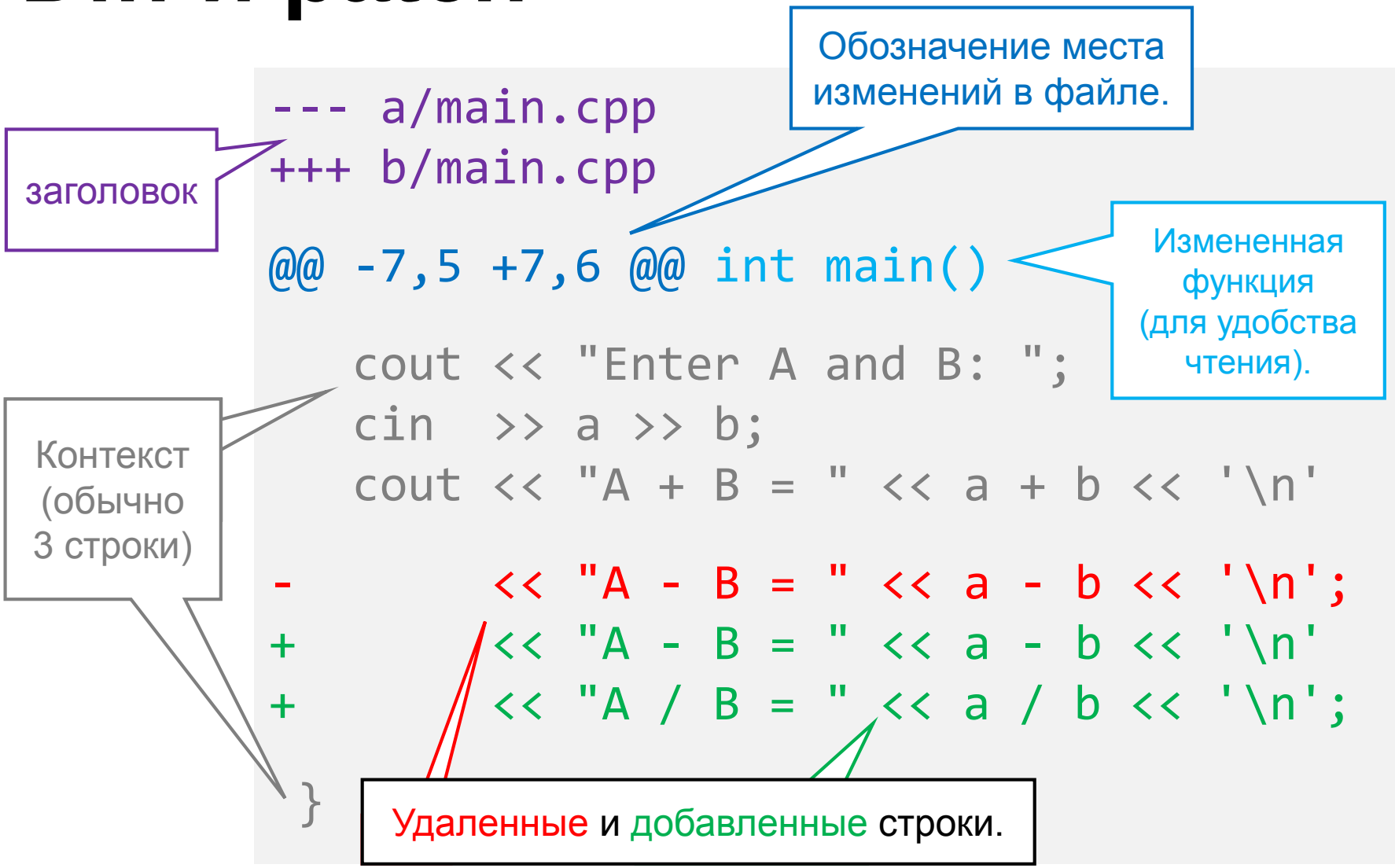
1. Найти commit C — первого общего предка A и B.
2. Загрузить из удаленного хранилища цепочку commit-ов от C до B.
3. Совместить изменения, сделанные в локальном от C до A, с загруженными изменениями от C до B.
4. Результат записать как новый commit M.

Слияние версий (merge)





Diff и patch


заголовок
Обозначение места изменений в файле.
Измененная функция (для удобства чтения).
Контекст (обычно 3 строки)
--- a/main.cpp
+++ b/main.cpp
@@ -7,5 +7,6 @@ int main()
cout << "Enter A and B: ";
cin >> a >> b;
cout << "A + B = " << a + b << '\n'
- << "A - B = " << a - b << '\n';
+ << "A - B = " << a - b << '\n'
+ << "A / B = " << a / b << '\n';
}
Удаленные и добавленные строки.

Основные понятия VCS

- **Слияние (merge) —**

  объединение двух версий (наборов изменений) в единую версию.

- **Конфликт (conflict) —**

  ситуация, когда VCS не может автоматически слить внесённые изменения (т. е. когда параллельно 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**).

- **Откат (revert*) —**

  удаление из истории изменений, внесенных commit-ом (не обязательно последним); «patch в обратную сторону».

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);
  - для переключения ветвей.

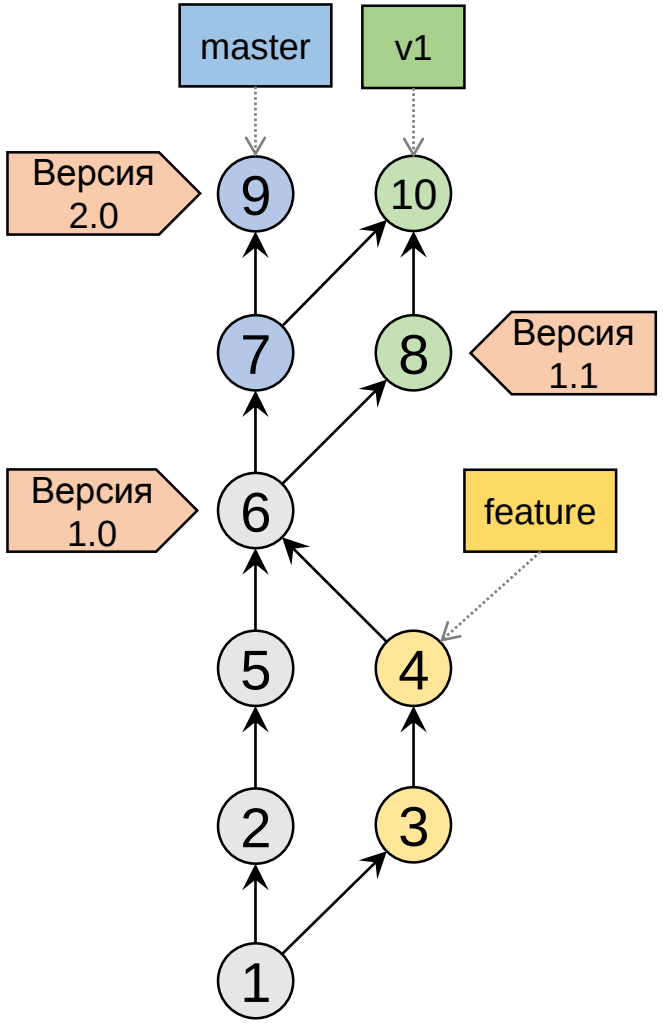
Ветвления и метки

- **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.
  - Для DVCS обновление — слияние одноименных веток локального и удаленного хранилищ.

- **Метка (tag) —** отмеченная версия.



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

- **Centralized:**

  участники просто синхронизируют хранилища через общее.

- **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.

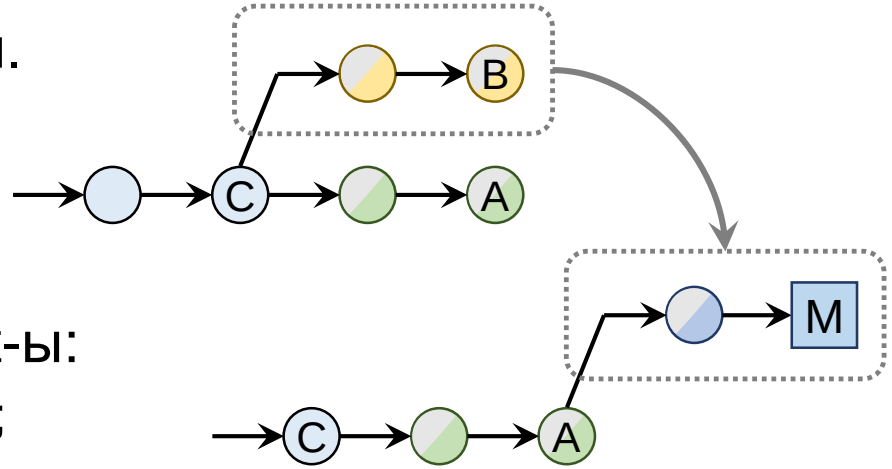
- В любой модели разумно периодически сливать обновления из «главной» ветви в дочернюю (feature, локальную и т. п.).

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

- **Rebase** — перенос ветви.
  - вариация merge;
  - удобно как update;
  - искажает историю.



- **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.

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.