Системы контроля версий

Java Collections
5 уровень, 1 лекция

ОТКРЫТА

— Привет, Амиго!

— Привет!

— Сегодня я расскажу тебе о системах контроля версий.

Как ты уже, наверное, знаешь, программы часто бывают очень большими и пишутся очень долго. Иногда десятки человек могут писать программу годами.

Проекты с миллионом строк кода – это реальность.

— Ничего себе.

— Это все очень сложно. Люди часто мешают друг другу, часто меняют один и тот же код и т.д. и т.п.

Чтобы внести порядок в эту неразбериху, программисты начали пользоваться **системами контроля версий** для кода своей программы.

Система контроля версий – это программа, представленная в виде клиента и сервера.

На сервере программа хранит данные (код, который пишут программисты), а программисты, с помощью программ-клиентов, добавляют или меняют его.

Основное ее отличие от просто программ, которые позволяют распределённую работу с документами, это то, что она хранит все предыдущие версии всех документов (файлов с кодом).

— А можно больше подробностей. Как это все работает?

— Вот представь, что ты – программист и хочешь внести небольшие изменения в исходный код программы, который хранится в репозитории (хранилище) на сервере.

Для этого тебе надо:

1) Залогиниться на сервер (Login).

2) Скопировать последнюю версию всех файлов к себе на компьютер – команда Checkout.

3) Внести изменения в нужные файлы.

4) Запустить программу локально и проверить, что она компилируется и работает.

5) Отправить свои «изменения» на сервер – команда Commit.

— В общем ясно.

— Но и это еще не все. Вот представь, что ты пришел утром на работу, а в Индии уже обед и твои индийские коллеги уже что-то поменяли, и закоммитили свои изменения в ваш репозиторий (хранилище) на сервере.

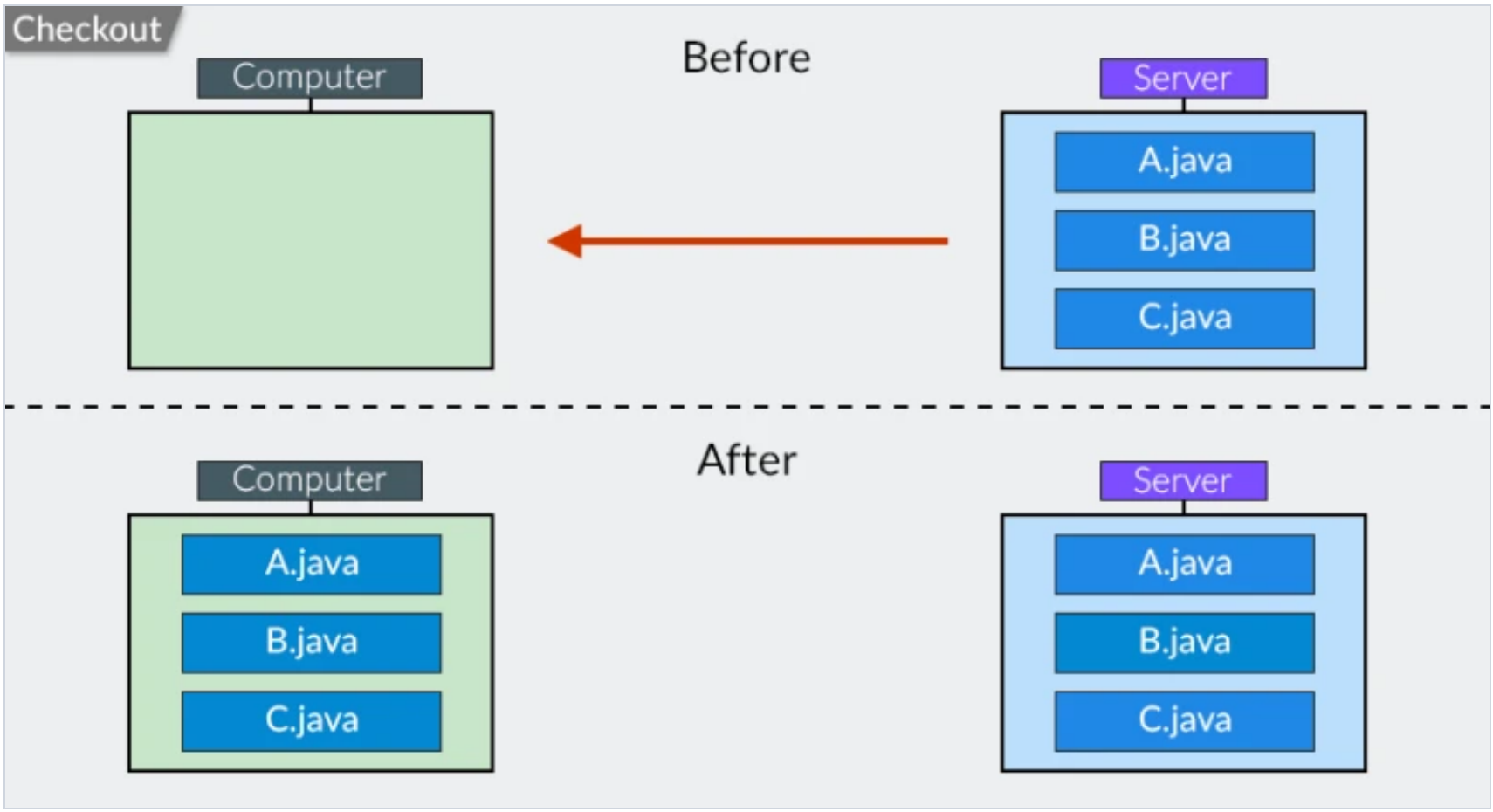
Тебе же надо работать с последней актуальной версией данных. Тогда ты выполняешь команду **Update**.

— А в чем ее отличие от **Checkout**?

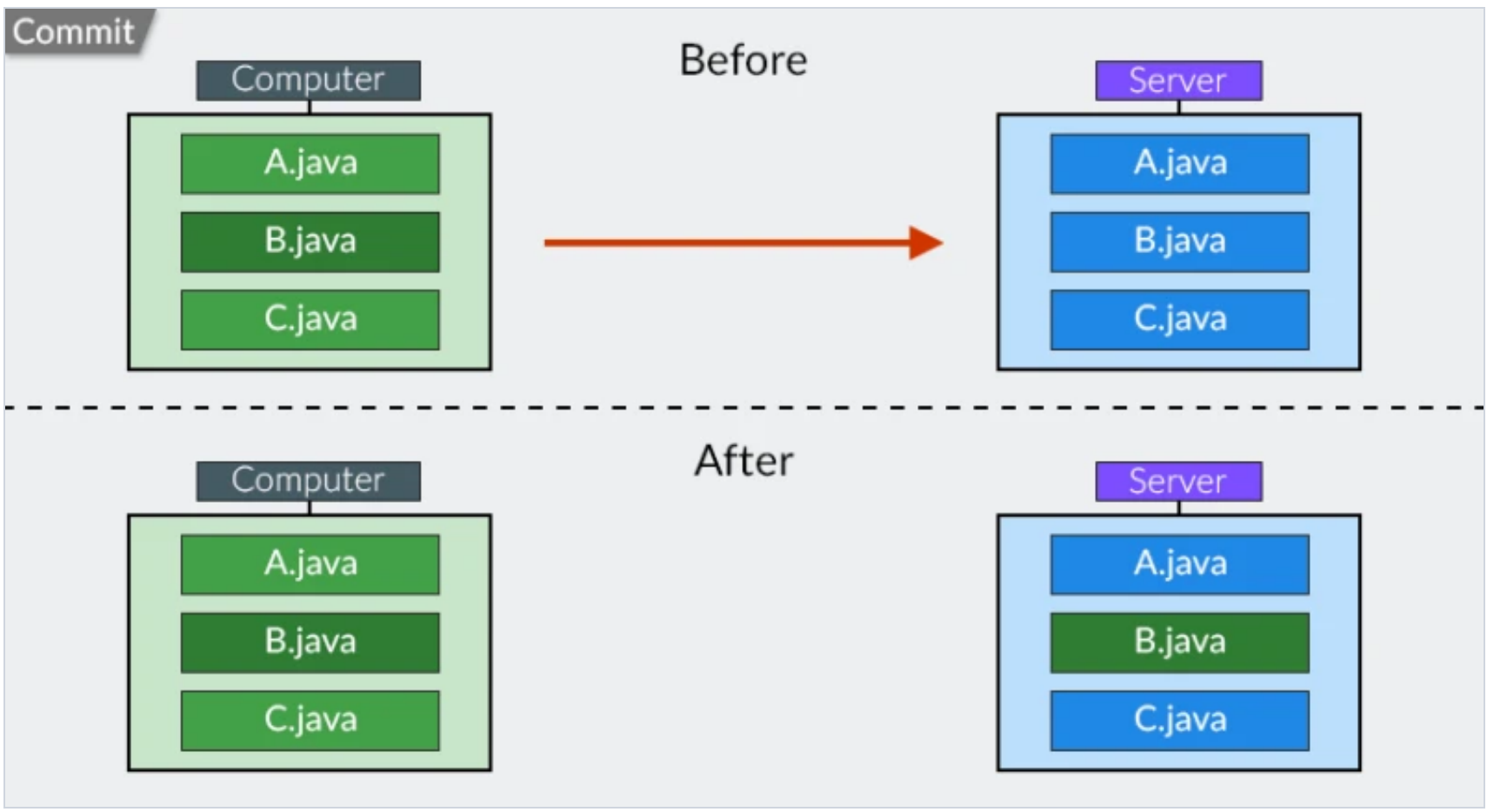
— **Checkout** предназначена для копирования всех файлов репозитория, а команда Update – только для тех файлов, которые

Вот как примерно это работает:

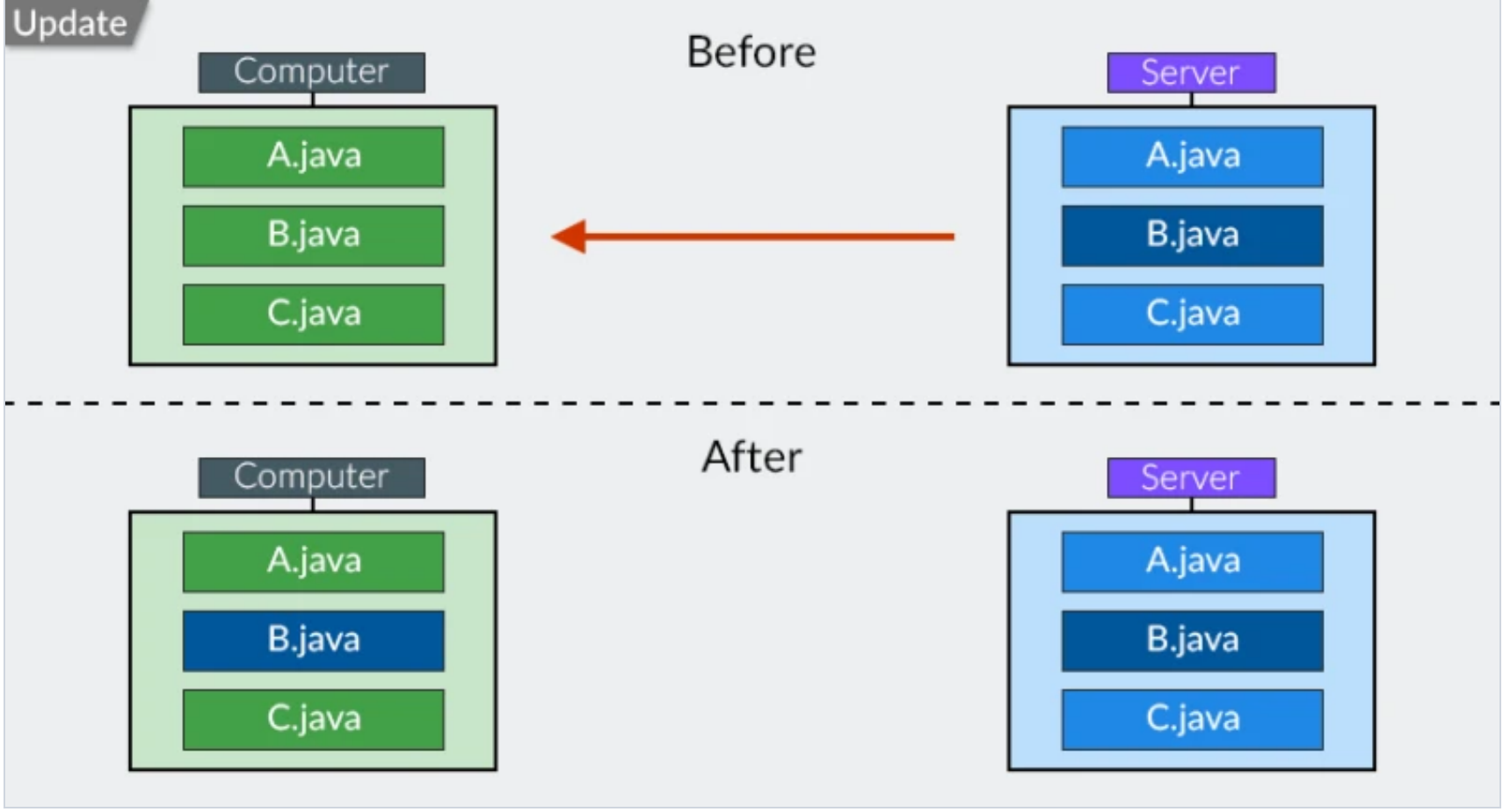
Checkout:



Теперь допустим, мы изменили файл B и хотим залить его на сервер. Для этого надо использовать команду **Commit:**



А вот как работает команда **Update**:



— Как интересно. А есть еще команды?

— Да, и довольно много. Но они могут быть разные в зависимости от того, какую именно программу контроля версий ты выберешь. Поэтому я стараюсь рассказывать только общие принципы.

Есть еще такая операция, как мэрджинг – объединение двух документов. Допустим, два программиста одновременно меняли один и тот же файл. Тогда программа на сервер не даст обоим его закоммитить. Кто первый – тот и прав.

— А что делать второму?

Второму предложат сделать **Update**, чтобы забрать к себе последние изменения с сервера. Это кстати хороший стиль – перед коммитом делать **Update**

Затем, во время исполнения операции Update, программа-клиент попробует объединить локальные изменения с изменениями, полученными с сервера.

Если программисты вносили изменения в разные части файла, то программа контроля версий, скорее всего, успешно объединит их. Если в одно и то же место, то программа контроля версий сообщит о конфликте во время мерджа и предложит пользователю самому вручную объединить эти изменения.

Такое часто случается, когда например, оба программиста добавили что-то в конец файла.

— Ясно. Разумно, в общем.

— И еще одна вещь – ветки.

Представь, что двум программистам из команды дали задание переписать один модуль. Или, ещё лучше – написать его заново. Пока этот модуль закончен не будет, программа не сможет работать, а может быть даже и компилироваться.

— Что же делать?

— Для этого в репозиторий добавили ветки (branches). Т.е. грубо говоря – разделили репозиторий на две части. Но не по файлам или директориям, а по версиям.

Представь, что электричество так и не было открыто, и роботы не были бы изобретены. Тогда не было бы трех освободительных войн, и вся история людей прошла бы по совсем иному пути.

Такой путь – это и есть альтернативная ветка истории.

Или можешь просто попробовать представить, что ветка – это просто копия репозитория. Т.е. в какой-то момент, мы клонировали репозиторий на сервере и у нас кроме главного (который часто называют trunk – ствол) появился еще один – branch (ветка).

НАЧАТЬ ОБУЧЕНИЕ

А почему нельзя было просто сказать, что мы скопировали репозиторий?

— Это не есть копирование в чистом виде.

Такие ветки можно не только отделять от ствола (trunk), но и присоединять к нему.

— Т.е. можно какую-то работу выполнить в ветке, а когда она будет закончена – добавить репозиторий-ветку к репозиторию-стволу?

— Ага.

— И что при этом станет с файлами?

‹ Предыдущая лекция

Следующая лекция ›

+71

Комментарии (26) популярные новые старые

JavaCoder

Введите текст комментария

Петр Селищев Уровень 26, Санкт-Петербург, Россия 16 июня, 09:55

Бесплатный курс по Гиту:

https://ru.hexlet.io/courses/intro_to_git

Как загрузить существующий проект на GitHub

Отправка проекта на GitHub из IntellijIDEA

Цикл Git разработки

Ответить +2

Дмитрий Рыбин Уровень 40, Краснодар, Россия 2 февраля, 00:22

Обучалка

Ответить +5

IwanIV Уровень 41, Россия 28 декабря 2021, 16:25

За Git и контроль версий надо было рассказывать в начале курса, когда рассказывали про Идею.

Ответить +5

Ars Уровень 41 21 ноября 2021, 17:20

Перевод trunk как ствол - очень странный перевод...

Ответить 0

SchlechtGut Уровень 51, Москва 30 ноября 2021, 18:32

ствол дерева это же trunk, логично получается

Ответить +1

aleksdenni Уровень 37, Полтава, Украина 9 сентября 2021, 22:52

Раз все оставляют ссылки то и я оставлю введение в гит

Ответить +2

Андрей Уровень 41, Иркутск, Россия 21 декабря 2020, 18:43

Могу посоветовать неплохой курс по GIT на GeekBtains "Git. Базовый курс" - он абсолютно бесплатный, состоит из 13 уроков, объясняют все с самого нуля (включая установку и настройку GIT) - вполне полезно, я считаю)

Отличный курс, тоже его прошел

Ответить

0

Raphael Уровень 41, Москва, Россия

10 декабря 2020, 10:04

Интерактивная **Git-обучалка** на русском

Ответить

+24

Stepan A. Уровень 37, Сочи, Россия

16 июня 2020, 12:06

Основы Git

Ответить

+7

Vorlock Уровень 31, Днепр, Украина

22 января 2020, 05:55

5) Отправить свои «изменения» на сервер – команда Commit.
если говорить в рамкам git - то это не верное утверждение. на сервер отправляет команда push. а commit добавляет внесенные изменения в local area.

Ответить

+9

alex Уровень 41, Россия

25 апреля 2020, 10:59

тут говориться не про конкретную систему а в общем, в следующей лекции будет уже про гит

Ответить

+2

VN Уровень 40, Россия

18 декабря 2019, 17:59

Ох, уж эти Индусы

Ответить

+1

Показать еще комментарии