

# Системы контроля версий

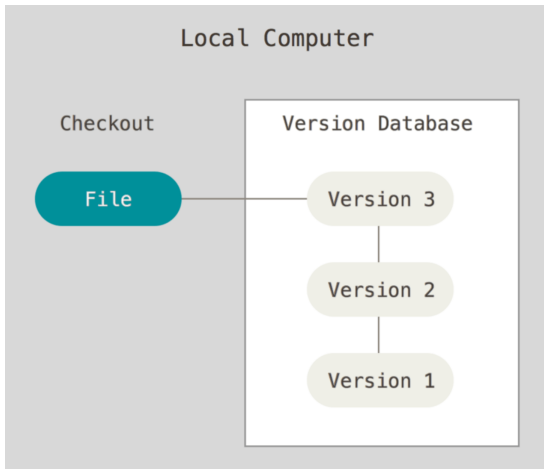
Юрий Литвинов  
y.litvinov@spbu.ru

21.09.2022

# Мотивация

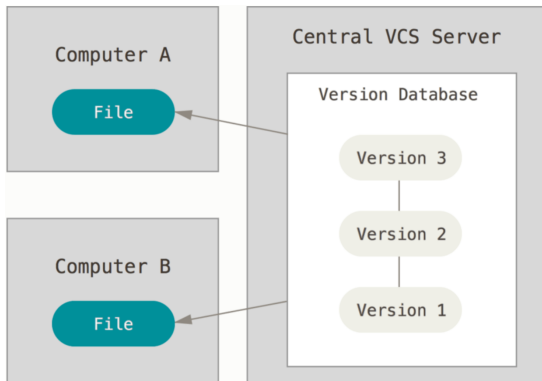
- ▶ Откат изменений
  - ▶ Система контроля версий — это машина времени
- ▶ Управление версиями
  - ▶ Машина времени с альтернативными временными ветками
- ▶ Централизованное хранение кода
- ▶ Командная разработка

# Локальные копии



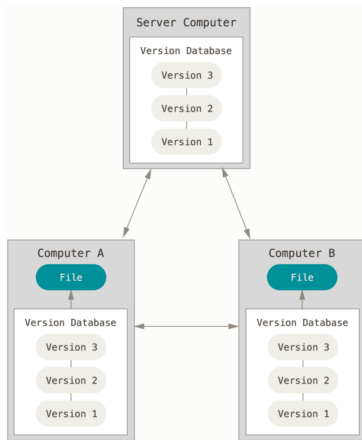
© <https://git-scm.com/book/ru>

# Централизованные VCS



© <https://git-scm.com/book/ru>

# Распределённые VCS



© <https://git-scm.com/book/ru>

# Дельты

- ▶ Системы контроля версий управляют изменениями
- ▶ Единица работы — коммит, набор изменений по всем файлам репозитория
- ▶ Файлы системы контроля версий не очень интересуют!
- ▶ Рабочая копия — папка с исходниками, где ведётся разработка

	21	<code>+#include &lt;QtScript/QtScriptEngine&gt;</code>
	22	<code>+#include &lt;QtScript/QtScriptValue&gt;</code>
18	23	
19		<code>-#include &lt;trikControl/brickInterface.h&gt;</code>
20	24	<code>#include &lt;trikControl/brickFactory.h&gt;</code>
21	25	
	26	<code>+#include &lt;trikKernel/fileUtils.h&gt;</code>
	27	<code>+</code>
22	28	<code>using namespace tests;</code>
23	29	

# Конкретно Git

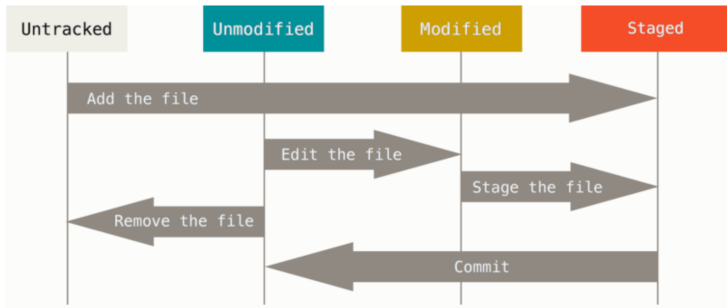
- ▶ Не GitHub!
- ▶ Не файлообменник
- ▶ Распределённая система контроля версий
- ▶ Удобная поддержка веток
- ▶ 2005 год, Линус Торвальдс (тот самый), для версионирования ядра Linux
  - ▶ Был написан за несколько дней после ссоры с BitKeeper-ом как набор скриптов на Bash-е
  - ▶ Речь про сотни тысяч коммитов и тысячи коммитеров
- ▶ С тех пор реализован под все нормальные ОС в виде библиотеки
  - ▶ И консольной утилиты, её использующей

# Что поставить, чтобы всё работало

- ▶ Git for Windows, <https://git-scm.com/download/win>
- ▶ Linux/MacOS: `apt install git`, `brew install git` или что-то подобное
  - ▶ Кстати, для Windows есть Chocolatey: `choco install git`
- ▶ На первое время какой-нибудь графический клиент:
  - ▶ Github Desktop
  - ▶ TortoiseGit
  - ▶ SmartGit
  - ▶ ...
  - ▶ Плагин к вашей любимой IDE



# Git, жизненный цикл файла



© <https://git-scm.com/book/ru>

## Основные команды

- ▶ `git add` — добавить новый файл под управление `git` или добавить изменение к коммиту
  - ▶ Add... в TortoiseGit
- ▶ `git status` — показать список изменённых/добавленных/удалённых файлов
  - ▶ Diff в TortoiseGit
- ▶ `git diff` — показать изменения по каждому файлу
  - ▶ В окне Diff двойным кликом по файлу в TortoiseGit
- ▶ `git commit` — зафиксировать изменения, создав новый коммит
  - ▶ Git Commit в TortoiseGit
- ▶ `git log` — просмотреть список коммитов
  - ▶ Show log в TortoiseGit
- ▶ `git restore` — откатить изменения в файле или перейти на другую ветку
  - ▶ Revert... или Switch/Checkout
  - ▶ Есть ещё `git reset --hard`, плюс `git clean -dfx`

# Демо

- ▶ Надо скачать и поставить консольный клиент Git и TortoiseGit
- ▶ Создадим локально репозиторий, научимся коммитить файлы, смотреть историю и откатывать изменения

# Ветки

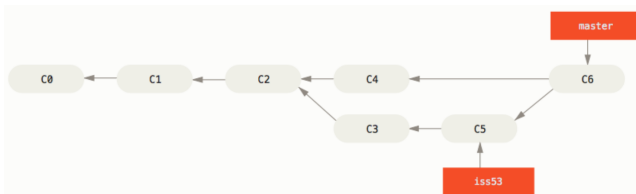
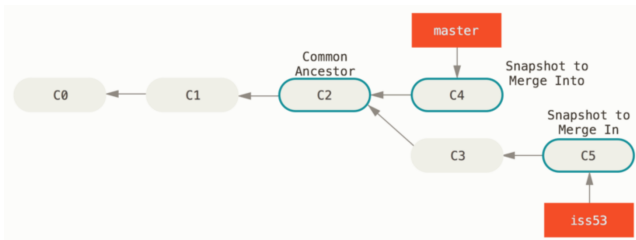
- ▶ Ветка — цепочка коммитов
- ▶ История коммитов может ветвиться, как дерево
  - ▶ Хотим проверить вариант решения
  - ▶ Хотим не выкладывать в основную ветку недоделанную работу
  - ▶ Хотим при командной разработке не мешать друг другу
- ▶ `git checkout -b <имя ветки>` — отвести новую ветку от текущей
- ▶ `git checkout <имя ветки>` — переключиться на указанную ветку
  - ▶ Switch/Checkout в TortoiseGit, там же флажок Create New Branch
- ▶ Очень важно следить, от какой ветки отводим новую

## Слияние веток

`git merge <имя ветки>` — притянуть изменения из указанной ветки в текущую

`$ git checkout master`

`$ git merge iss53`



# Конфликты

```

Theirs - REMOTE
128 // Enables or disables all editor actions.
129 void setActionsEnabled(bool enabled) {
130
131
132
133 public slots:
134 qReal::Id::createElement(const QString &type) {
135
136 void cut() {
137 void copy() {
138 void paste(bool logicalCopy) {
139
Mine - LOCAL
138 // Enables or disables all editor actions.
139 void setActionsEnabled(bool enabled) {
140
141
142 // Handles deletion of the element from scene.
143 void onElementDeleted(Element *element) {
144
145 // Enable or Disable mousegestures
146 void enableHouseGestures(bool enabled) {
147 public slots:
148 qReal::Id::createElement(const QString &type) {
149
150 void cut() {
151 void copy() {
152 void paste(bool logicalCopy) {
153
Merged - editorViewScene.h
138 // Enables or disables all editor actions.
139 void setActionsEnabled(bool enabled) {
140
141
142
143
144
145
146
147 public slots:
148 qReal::Id::createElement(const QString &type) {
149
150 void cut() {
151 void copy() {
152 void paste(bool logicalCopy) {
153

```

```

TestClass.java
package com.sap.text;

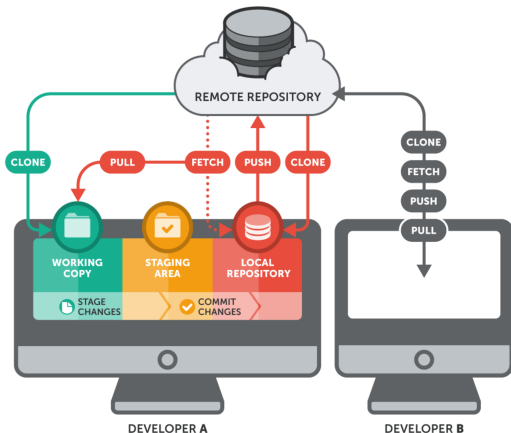
public class TestClass {

    <<<<<< CURS
    void doSomethingElse() {
        =====
    void doSomething() {
        >>>>>> THEIRS
    }
}

```

# Удалённые репозитории

- ▶ git clone
- ▶ git remote
- ▶ git push
- ▶ git fetch
- ▶ git pull



© <https://www.git-tower.com/learn/git/ebook/en>

# GitHub

- ▶ GitHub — один из облачных хостингов git-репозиторий
  - ▶ Никто не мешает использовать Git локально или поднять сервер самому
- ▶ Ещё из известных GitLab (включая self-hosted), BitBucket, Azure DevOps Server, ...
- ▶ Не только хостинг репозитория, но и:
  - ▶ UI для просмотра исходников и поиска по ним
  - ▶ Пуллреквесты
  - ▶ Форки
  - ▶ Социальное взаимодействие
  - ▶ CI/CD
  - ▶ Средства для управления проектом



# Демо

- ▶ Регистрируемся на GitHub
- ▶ Создаём репозиторий прямо на GitHub
- ▶ Клоним его себе
- ▶ Отводим ветку под домашку
- ▶ Делаем её там
- ▶ Коммитим/пушим
- ▶ Делаем пуллреквест в main

# Процесс работы

- ▶ Программист хочет сделать новую фичу
- ▶ Отводит себе ветку от main-a
- ▶ Реализует там фичу
- ▶ Тестирует и рефакторит её, когда считает, что она готова, делает пуллреквест
- ▶ Пока пуллреквест ревьюят, программист делает новую фичу (опять-таки, отведя новую ветку от main-a)
- ▶ По пуллреквесту появляются замечания, программист переключается на ветку пуллреквеста и правит там замечания
- ▶ Когда поправил, коммитит и пушит исправления, они автоматически добавляются в пуллреквест
- ▶ Просит ревьюеров, чтобы они посмотрели фиксы
- ▶ Переключается обратно на свою рабочую ветку и продолжает писать код, возможно, делая ещё пуллреквесты
- ▶ Цикл повторяется до тех пор, пока пуллреквест не принимают
- ▶ Программист удаляет ветку с фичей, когда она замержена

# Хорошие практики

- ▶ Коммитим только исходные тексты, конфиги, картинки и т.п.
- ▶ Если что-то может быть автоматически сгенерировано по тому, что уже есть в репозитории, это НЕ коммитим
- ▶ Всегда пишем адекватные комментарии к коммитам
  - ▶ Одно-два осмысленных предложения, в повелительной форме
  - ▶ Комментарии к коммиту обязательны, за fix или "." сразу увольняют
- ▶ Коммитим как можно чаще
  - ▶ Сделали что-то осмысленное — коммит

## Хорошие практики (2)

- ▶ Никогда не коммитим исполнимые файлы (включая .dll/.so), объектные файлы, локальные настройки
  - ▶ Из каждого правила есть исключения, но всё же
- ▶ Обязательно выкладываем файлы, необходимые для сборки
  - ▶ Проектные файлы, мейкфайлы, скрипты и т.п.
  - ▶ Visual Studio: .vcxproj, .sln
  - ▶ IDEA: всё содержимое папки .idea, кроме workspace.xml и tasks.xml
- ▶ Имеет смысл проверить, клонив репозиторий в пустую папку
  - ▶ Кстати, никто не запрещает иметь локально несколько копий репозитория, например, с разными ветками

## Хорошие практики (3)

- ▶ Коммит не должен содержать в себе файлы, не относящиеся к изменениям
  - ▶ .gitignore
  - ▶ <https://github.com/github/gitignore>
  - ▶ Обязательно надо проверять, что коммитим (git diff)
- ▶ Коммит не должен добавлять/убирать пустые строки, менять пробелы на табы и т.д., если это не суть коммита
- ▶ Стилль исходного кода и отступов должен совпадать с текстом вокруг
- ▶ Делаете пуллреквест — посмотрите diff

**In case of fire**



1. git commit



2. git push



3. leave building

## Ещё полезные команды

- ▶ `git add -p` — интерактивное добавление изменений к коммиту, позволяет коммитить только часть файла
- ▶ `git commit --amend` — исправить последний коммит
  - ▶ `git commit --amend -m "an updated commit message"`
  - ▶ Применять **только до** `git push`
- ▶ `git rebase <имя ветки>` — положить изменения из данной ветки поверх текущей, альтернатива `merge`
- ▶ `git reset --hard` — откатить все изменения в рабочей копии до последнего коммита
  - ▶ Обязательно проверить `git status`, что не откатите лишнего
- ▶ `git reset --hard <хеш коммита>` — откатить все изменения в текущей ветке до указанного коммита, забыть все коммиты, что были после
  - ▶ И случайно грохнуть всю домашку перед зачётом