

Системы контроля версий, git

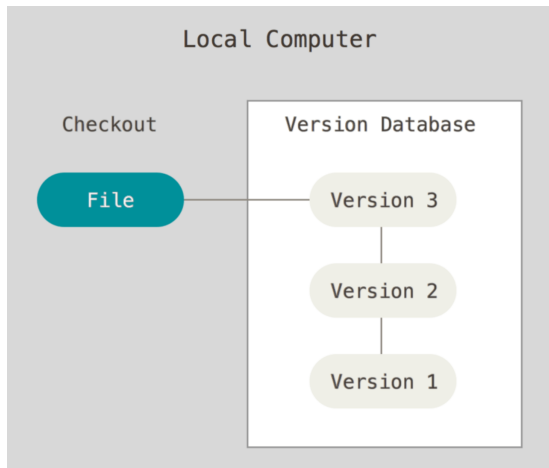
Юрий Литвинов
yurii.litvinov@gmail.com

28.09.2018

Мотивация

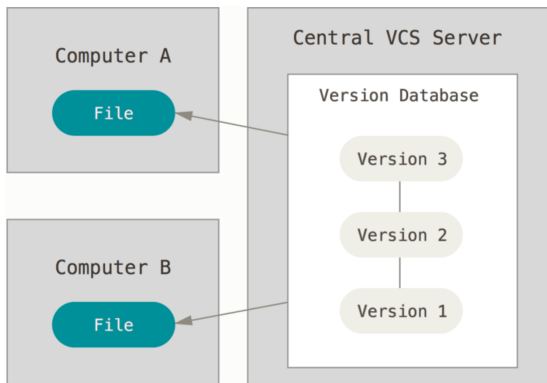
- ▶ Откат изменений
- ▶ Управление версиями
- ▶ Централизованное хранение кода
- ▶ Командная разработка

Локальные копии



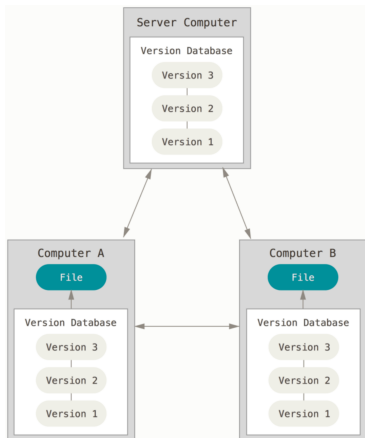
© <https://git-scm.com/book/ru>

Централизованные VCS



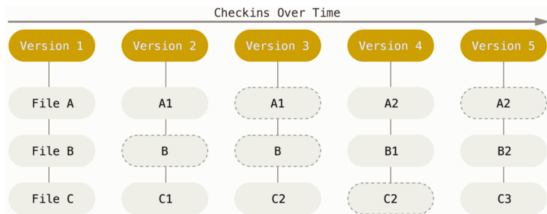
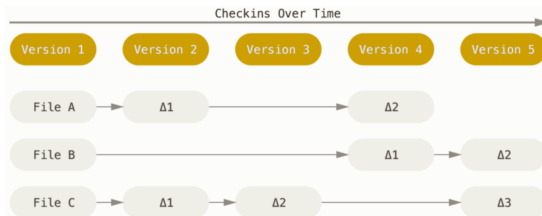
© <https://git-scm.com/book/ru>

Распределенные VCS



© <https://git-scm.com/book/ru>

Управление версиями

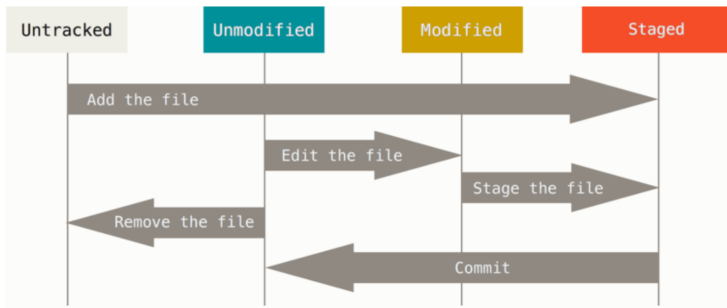


© <https://git-scm.com/book/ru>

Дельта

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

Жизненный цикл файла



© <https://git-scm.com/book/ru>

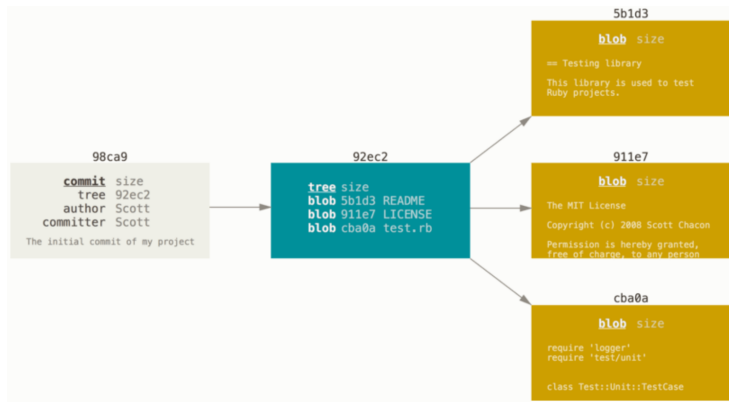
Основные команды

- ▶ `git add` — добавить новый файл под управление `git` или добавить изменение к коммиту
- ▶ `git status` — показать список изменённых/добавленных/удалённых файлов
- ▶ `git diff` — показать изменения по каждому файлу
- ▶ `git commit` — зафиксировать изменения, создав новый коммит
- ▶ `git rm` — удалить файл и удалить его из репозиторий
- ▶ `git log` — просмотреть список коммитов
- ▶ `git checkout` — откатить изменения в файле или перейти на другую ветку

Как всё устроено

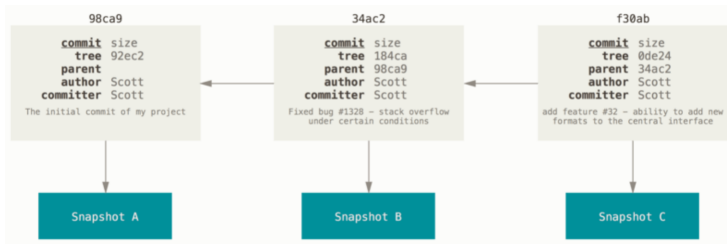
```
$ git add README test.rb LICENSE
```

```
$ git commit -m 'initial commit of my project'
```



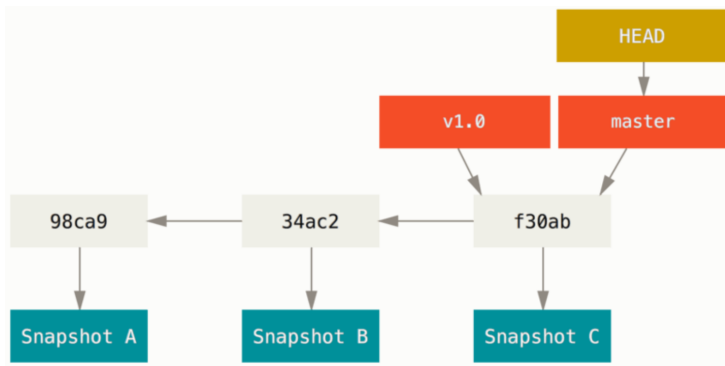
© <https://git-scm.com/book/ru>

Коммит и его родители



© <https://git-scm.com/book/ru>

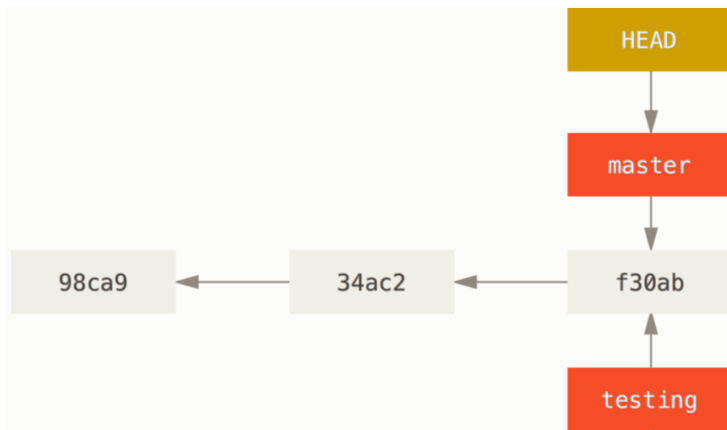
Ветки



© <https://git-scm.com/book/ru>

Создание ветки

\$ git branch testing



© <https://git-scm.com/book/ru>

Переключение ветки

\$ git checkout testing



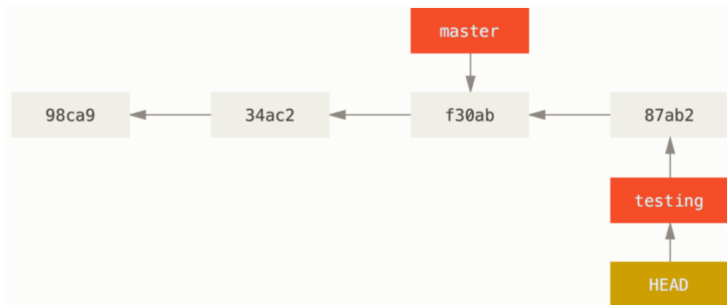
© <https://git-scm.com/book/ru>

Новый коммит

<Что-то поделали с файлами в рабочей копии>

\$ git add <изменения, которые хотим коммитить>

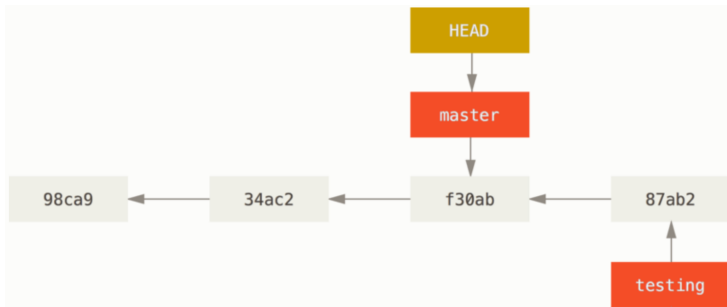
\$ git commit -m 'made a change'



© <https://git-scm.com/book/ru>

Переключимся на master

\$ git checkout master



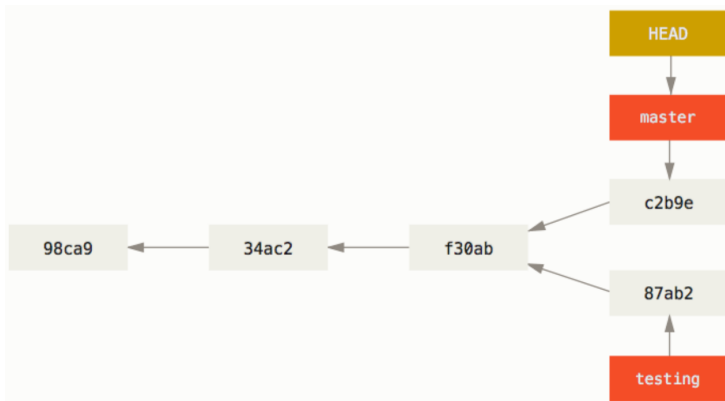
© <https://git-scm.com/book/ru>

Сделаем новый КОММИТ там

<Что-то поделали с файлами в рабочей копии>

\$ git add <изменения, которые хотим коммитить>

\$ git commit -m 'made other changes'



© <https://git-scm.com/book/ru>

Слияние веток

\$ git checkout master

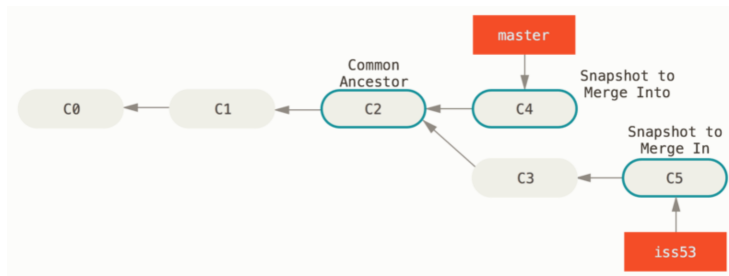
Switched to branch 'master'

\$ git merge testing

Merge made by the 'recursive' strategy.

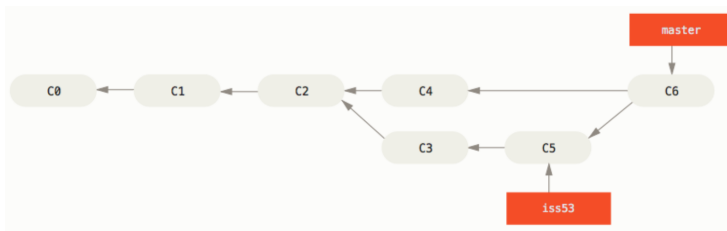
index.html | 1 +

1 file changed, 1 insertion(+)



© <https://git-scm.com/book/ru>

Результат



© <https://git-scm.com/book/ru>

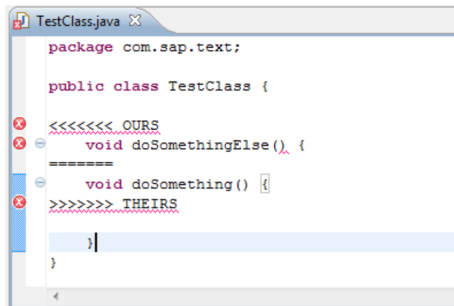
Конфликты

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

Конфликты в коде



```
TestClass.java X
package com.sap.text;

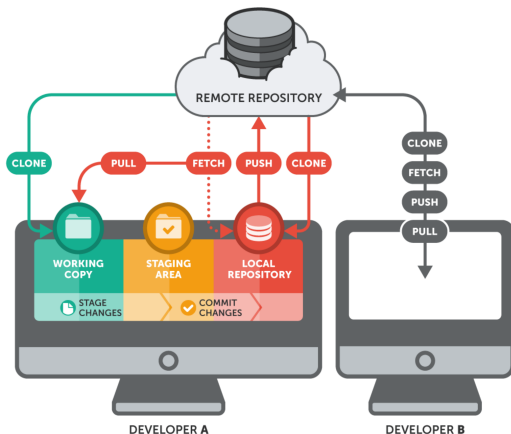
public class TestClass {

<<<<<<< OURS
    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>

Процесс работы

- ▶ Программист хочет сделать новую фичу
- ▶ Отводит себе ветку от мастера
- ▶ 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 мастера)
- ▶ 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

То же, но с домашкой

- ▶ Хотите сделать новую задачу
- ▶ Отводите себе ветку от мастера
 - ▶ `git checkout master`
 - ▶ `git branch my-cool-hometask-3.1`
- ▶ Создаёте **прямо там** проект, делаете задачу
- ▶ Коммитите, сколько хотите
 - ▶ После каждого значимого продвижения
 - ▶ `git add` на каждый новый файл, который надо коммитить
 - ▶ `git commit -a -m "Внятное описание изменений"`

То же, но с домашкой (2)

- ▶ Когда считаете, что задача готова, пушите её на гитхаб
 - ▶ `git push -u origin my-cool-hometask-3.1`
- ▶ Идёте на гитхаб, делаете пуллреквест
 - ▶ Выбираете ветку в “Branch:”
 - ▶ Жмёте на Pull request
 - ▶ Вводите внятное описание пуллреквеста
 - ▶ Жмёте на Create pull request
- ▶ Ссылку на то, что получилось, выкладываете на HwProj
 - ▶ Список всех пуллреквестов можно посмотреть во вкладке Pull requests на гитхабе
- ▶ Ждёте, пока я прокомментирую решение

То же, но с домашкой (3)

- ▶ В это время можно заняться следующей задачей
 - ▶ Не забыть `git checkout master`
 - ▶ `git branch my-cool-hometask-3.2`
- ▶ Получаете ревью на гитхабе, исправляете замечания
 - ▶ Переключаетесь на исходную ветку
 - ▶ Коммитите текущие изменения в рабочей копии
 - ▶ `git checkout my-cool-hometask-3.1`
 - ▶ Исправляете, коммитя сколько хотите
 - ▶ Коммитите исправления
 - ▶ Пушите на гитхаб
 - ▶ `git push`, без всяких `-u`
- ▶ Проверяете, что всё ок, в пуллреквестах на гитхабе
- ▶ Когда задача принята, мерджите пуллреквест на гитхабе и удаляете ветку

Что надо выкладывать

- ▶ .cpp, .h-файлы
- ▶ Проектные файлы:
 - ▶ Visual Studio: .vcxproj, .sln
 - ▶ Qt Creator: .pro
 - ▶ CLion: всё содержимое папки .idea, кроме workspace.xml и tasks.xml
- ▶ Текстовые файлы и прочие ресурсы, которые используются в тестах или во время работы программы

Что не надо выкладывать

- ▶ Бинарные файлы: .exe, .dll, бинарники под линуксом
 - ▶ Включите себе отображение расширений файлов
- ▶ Промежуточные результаты компиляции: .o, .obj, ...
- ▶ Скрытую папку .vs в Visual Studio
 - ▶ Включите себе отображение скрытых файлов
- ▶ Makefile-ы и подобные вещи в Qt Creator

.gitignore:

- ▶ <https://git-scm.com/docs/gitignore>
- ▶ <https://github.com/github/gitignore>

Хорошие практики

- ▶ Аккуратно заполняем Имя и Email
 - ▶ Желательно, чтобы они совпадали с именем аккаунта и почтой, с которой регались на гитхабе
- ▶ Коммитим только то, что нужно, чтобы получить в чистую папку и собрать проект
- ▶ Всегда пишем адекватные комментарии к коммитам
- ▶ Коммитим как можно чаще
- ▶ Один коммит — одна функциональность
 - ▶ Сделали что-то, хоть немного напоминающее осмысленное -> КОММИТ

Хорошие практики (2)

- ▶ Коммит не должен содержать в себе файлы, не относящиеся к изменениям
 - ▶ .gitignore
- ▶ 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

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 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

Полезные ссылки

- ▶ Консольный клиент: <https://git-scm.com/downloads>
- ▶ TortoiseGit: <https://tortoisegit.org/>
- ▶ Удобная консоль под винду: <http://cmder.net/>
- ▶ Удобный консольный файловый менеджер под винду:
<https://www.farmanager.com/>
- ▶ Книжка с картинками: <http://git-scm.com/book/ru/> (must read)
- ▶ Рекомендации по процессу от гитхаба:
<https://guides.github.com/introduction/flow/index.html> (для общего развития)
- ▶ Инструкция по пользованию гитом от Тимофея Брыксина:
https://docs.google.com/document/d/1URPcqZDMwIHDW9KoMbjoPwLJGswIMimq0-FD1_68rOY