Системы контроля версий

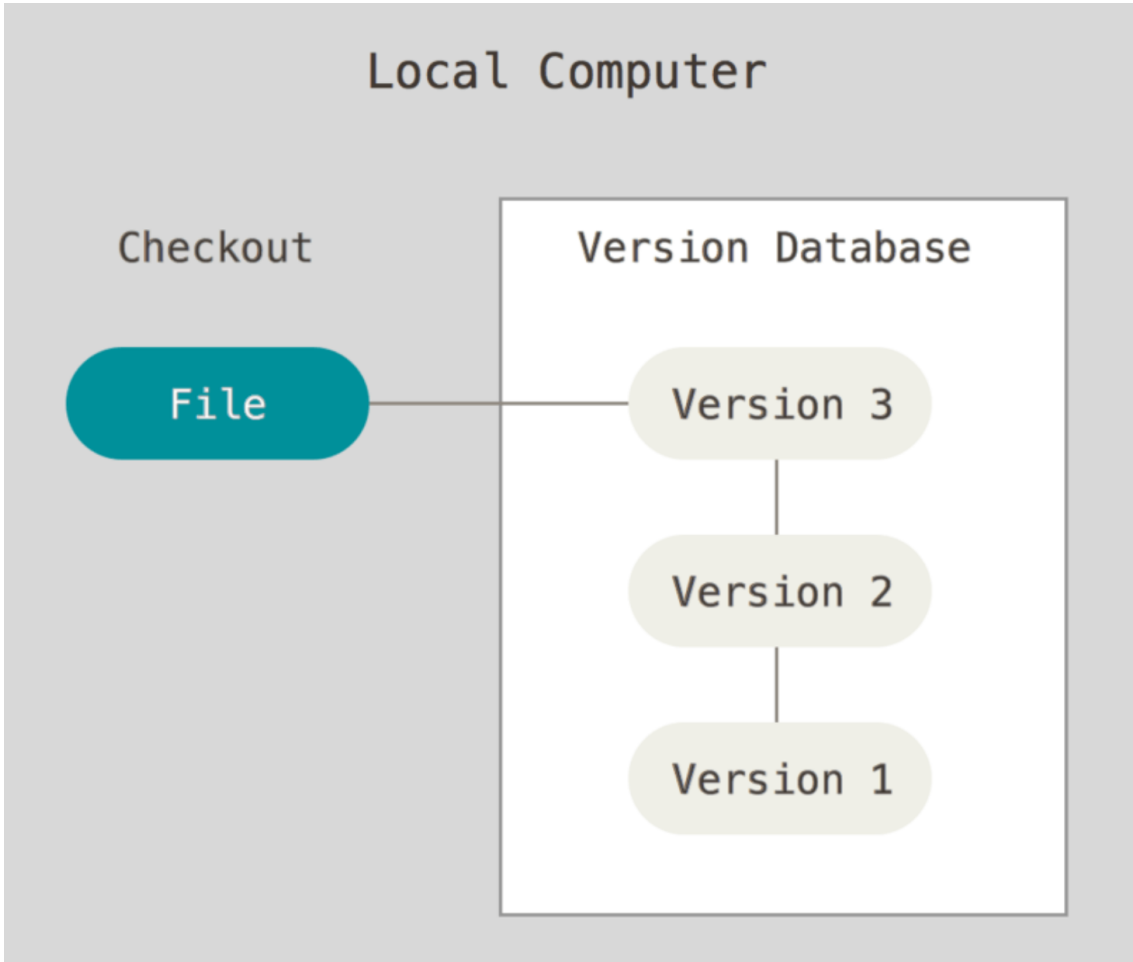
Юрий Литвинов
y.litvinov@spbu.ru

22.09.2023

Мотивация

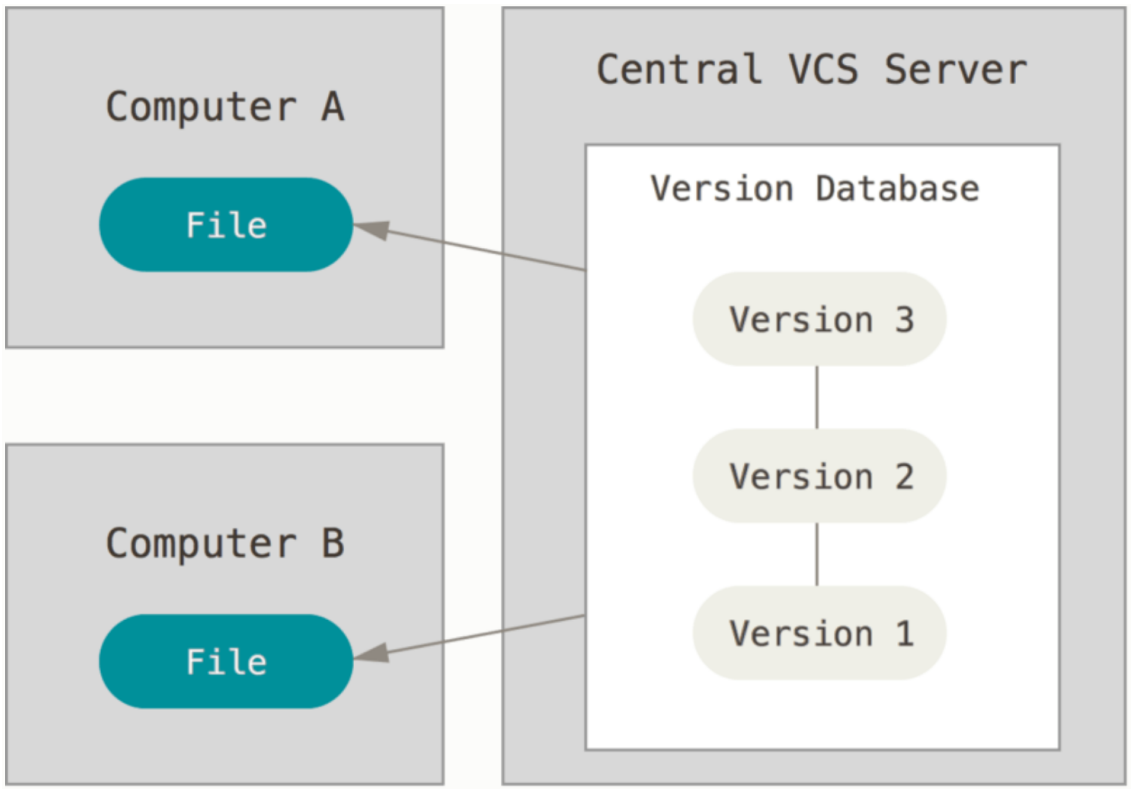
- Откат изменений
  - Система контроля версий — это машина времени
- Управление версиями
  - Машина времени с альтернативными временными ветками
- Централизованное хранение кода
- Командная разработка

Локальные копии



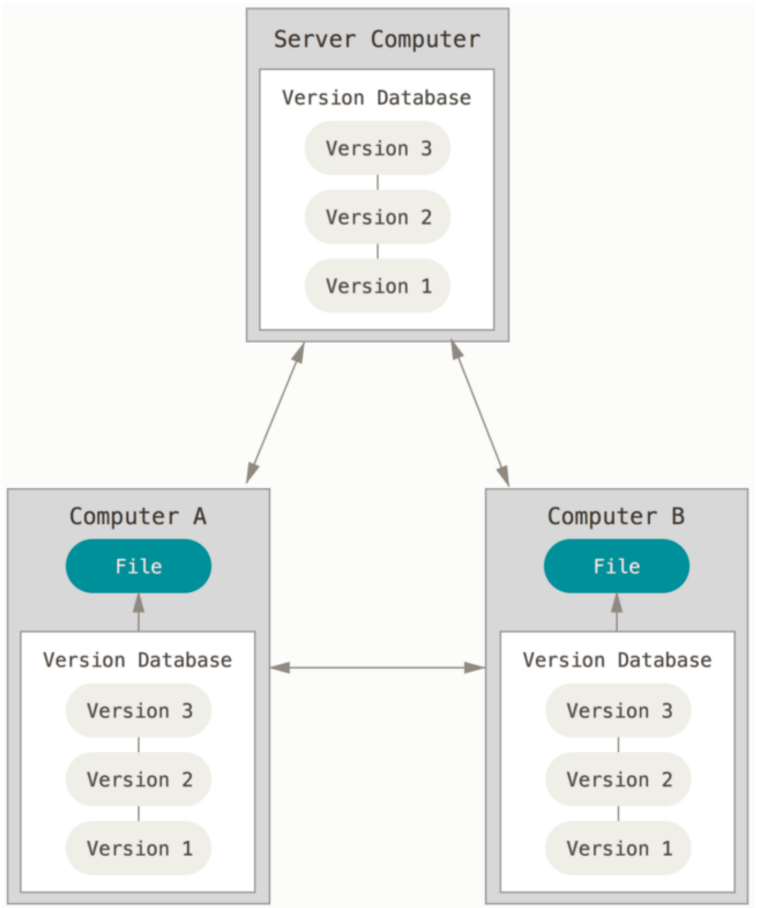
© https://git-scm.com/book/ru

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

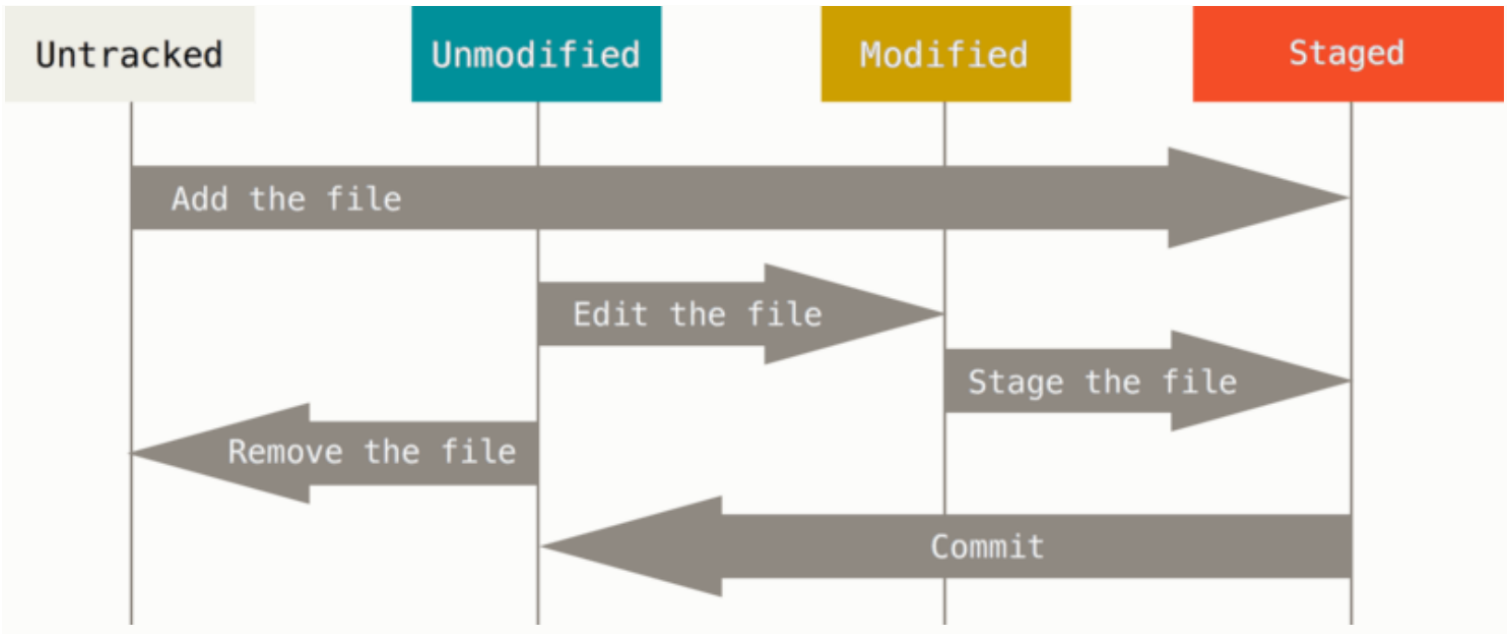
Конкретно Git

- 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

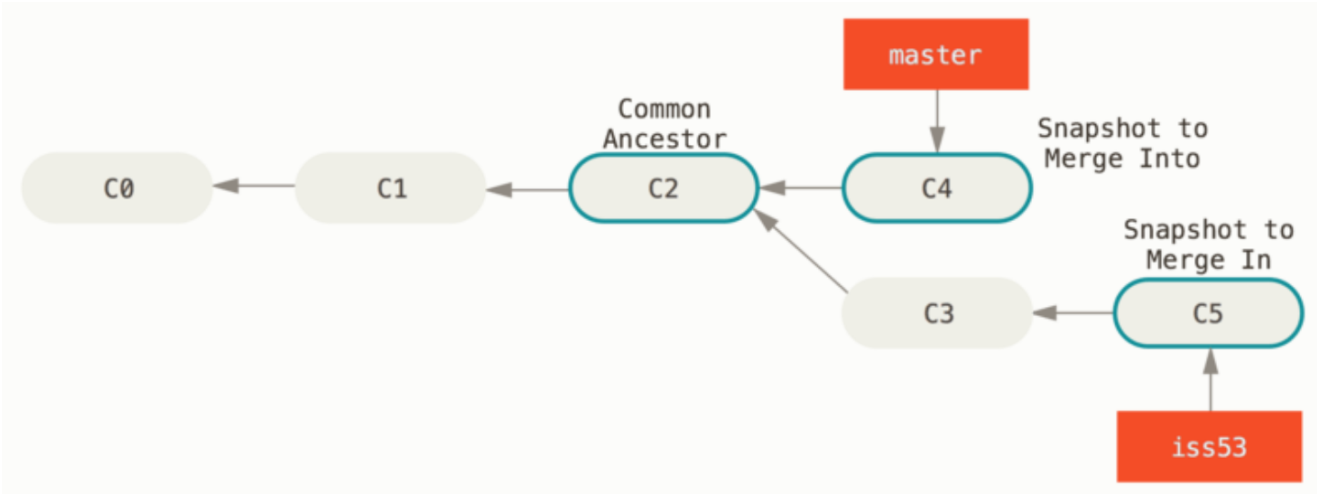
Ветки

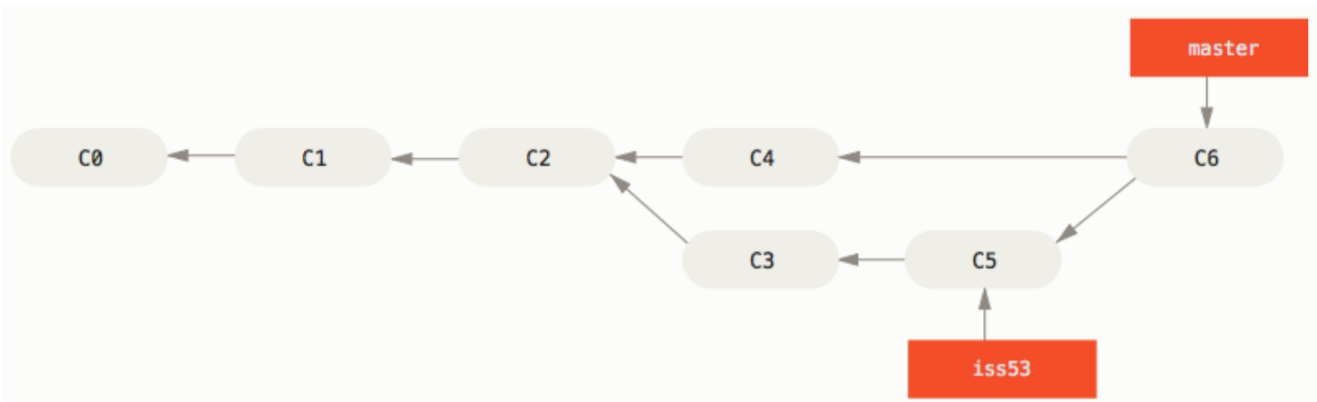
- Ветка — цепочка коммитов
- История коммитов может ветвиться, как дерево
  - Хотим проверить вариант решения
  - Хотим не выкладывать в основную ветку недоделанную работу
  - Хотим при командной разработке не мешать друг другу
- git checkout -b <имя ветки> — отвести новую ветку от текущей
- git checkout <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

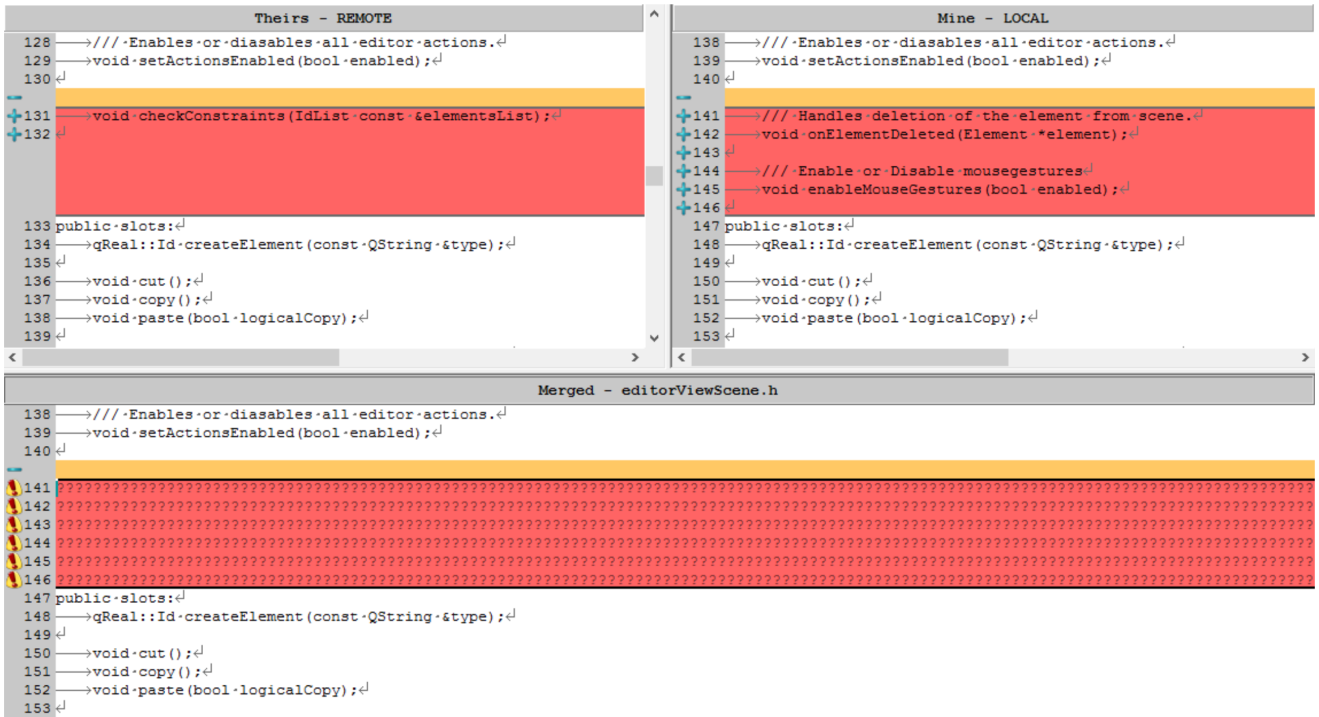
git merge <имя ветки> — притянуть изменения из указанной ветки в текущую

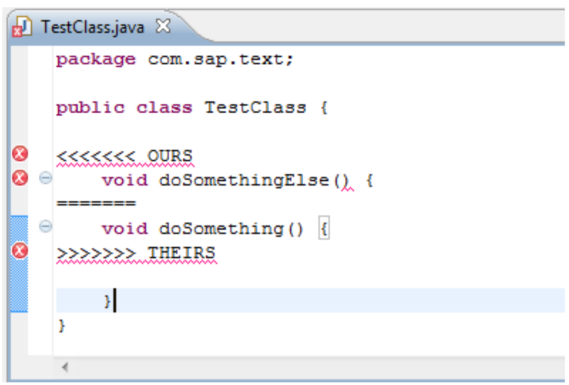
```
$ git checkout master
$ git merge iss53
```





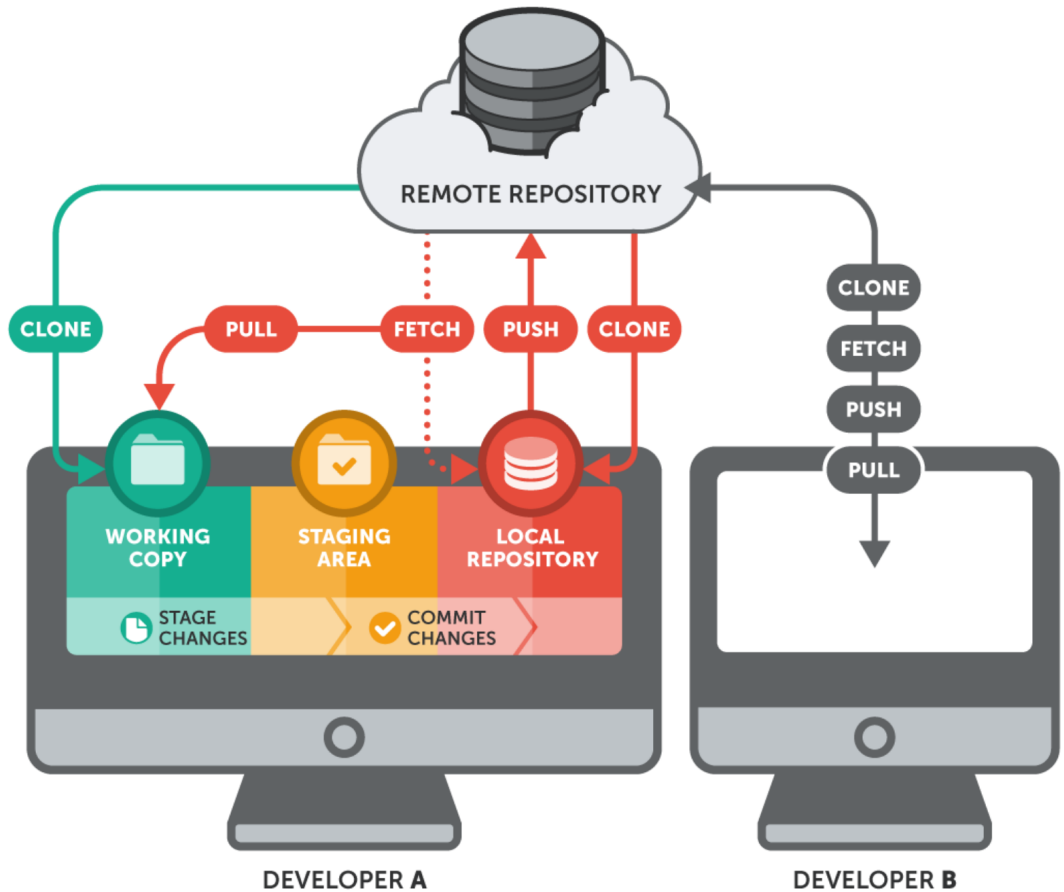
Конфликты


Theirs - REMOTE
/// Enables or diasables all editor actions.
void setActionsEnabled(bool enabled);
void checkConstraints(IdList const &elementsList);
public slots:
qReal::Id createElement(const QString &type);
void cut();
void copy();
void paste(bool logicalCopy);
Mine - LOCAL
/// Enables or diasables all editor actions.
void setActionsEnabled(bool enabled);
/// Handles deletion of the element from scene.
void onElementDeleted(Element *element);
/// Enable or Disable mousegestures
void enableMouseGestures(bool enabled);
public slots:
qReal::Id createElement(const QString &type);
void cut();
void copy();
void paste(bool logicalCopy);
Merged - editorViewScene.h
/// Enables or diasables all editor actions.
void setActionsEnabled(bool enabled);
public slots:
qReal::Id createElement(const QString &type);
void cut();
void copy();
void paste(bool logicalCopy);


TestClass.java
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



GitHub

- GitHub — один из облачных хостингов git-репозиториев
  - 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-а
- Реализует там фичу
- Тестит и рефакторит её, когда считает, что она готова, делает пуллреквест
- Пока пуллреквест ревьюят, программист делает новую фичу (опять-таки, отведя новую ветку от main-а)
- По пуллреквесту появляются замечания, программист переключается на ветку пуллреквеста и правит там замечания
- Когда поправил, коммитит и пушит исправления, они автоматом 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 замерджена

Хорошие практики

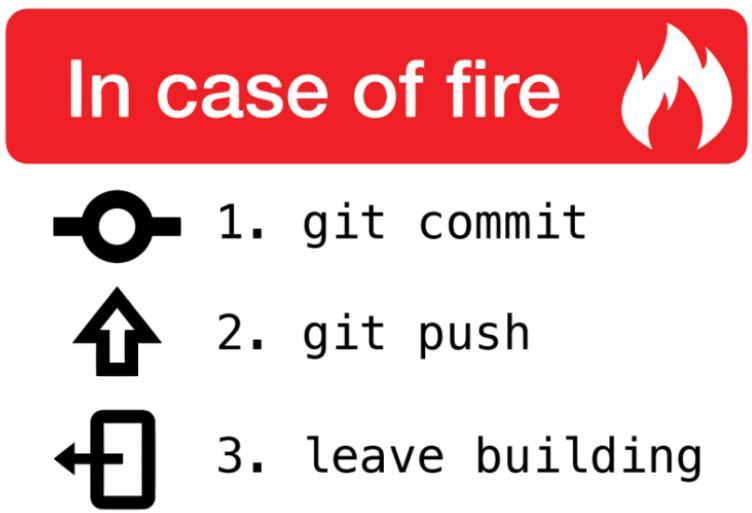
- Коммитим только исходные тексты, конфиги, картинки и т.п.
- Если что-то может быть автоматически сгенерировано по тому, что уже есть в репозитории, это НЕ коммитим
- Всегда пишем адекватные комментарии к коммитам
  - Одно-два осмысленных предложения, в повелительной форме
  - Комментарии к коммиту 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, склонив 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
- Стиль исходного кода и отступов должен совпадать с текстом вокруг
- Делаете пуллреквест — посмотрите diff



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