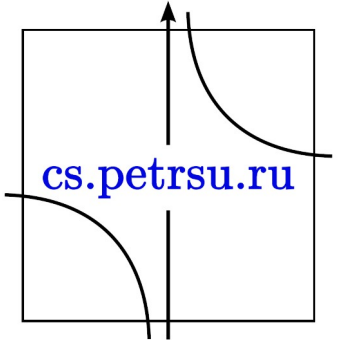




# Системы управления версиями

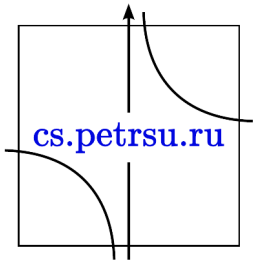
Лекция №3

# Особенности проекта

- Много разработчиков
- Ротация людей и мест(ПО, комп-в и т.д.)
- Распределенная работа + взаимодействие
- Эксперименты, хранение наработок

# Проблемы

- Совместная работа
- Откат к предыдущей версии
- Параллельная поддержка версий
- Принцип “работает – не трогай”



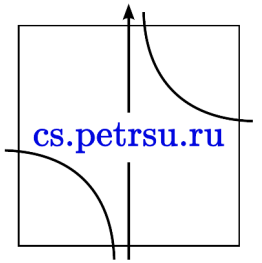
# Определения

- **Система управления версиями**(от англ. **Version Control System(VCS)**или **Revision Control System**) —программное обеспечение для облегчения работы с изменяющейся информацией
- **Система контроля версий (СКВ)**—это система, **регистрирующая изменения в одном или нескольких файлах** с тем, чтобы в дальнейшем была **возможность вернуться к** определённым **старым версиям этих файлов.**

# Термины

- **Репозиторий(**repository, depot), **сервер**— хранилище, хранит все исходные коды программы, а также историю их изменений.

- **Клиент**-имеет свою локальную копию (workingcopy) исходных кодов, с которой работает разработчик.



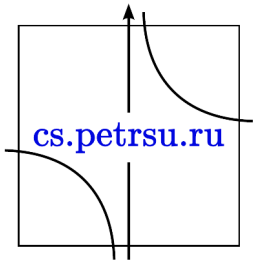
- **Рабочая (локальная) копия документов** (workingcopy)(check-out, clone) —извлечение документа из хранилища и создание рабочей копии.

- **Изменения(**changeset, activity**)** —набор изменений, проименованныйнабор правок, сделанных в локальной копии для какой-то цели.

- **Ревизия**(revision) —версия документа, новые изменения (changeset) создают новую ревизию репозитория.

- **Метка (***tag*, *label)*—пометка начала отсчета изменений в дереве, групируетнесколько файлов в пригодный для использования блок. Чаще всего используется для обозначения конечной версии файлов для сборки.

- **head**—самая свежая версия (revision) в хранилище.



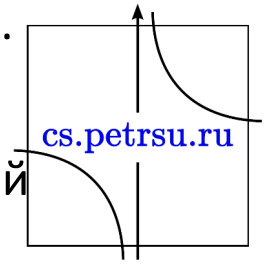
- **Ветвь**(*branch)*—направление разработки, независимое от других. Представляет собой копию части хранилища, в которую можно вносить свои изменения, не влияющие на другие ветви.

- **Обновление рабочей копии** (update, sync) -изменения, сделанные в основной версии, сливаются с локальными, происходит синхронизация рабочей копии до некоторого заданного состояния хранилища (в т.ч. и к более старому состоянию, чем текущее).

- **Коммит (Фиксация изменений,** check-in, commit, submit**) -**локальные изменения сливаются с изменениями, уже зафиксированными в основной версии.

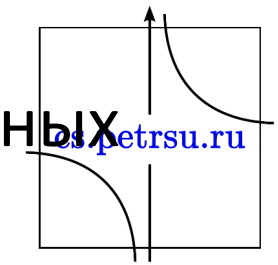
- **Слияние ветвей** (merge, integration) -объединение независимых изменений в единую версию документа -изменения, сделанные в одной ветви разработки, сливаются с изменениями, сделанными в другой.

- **Конфликт**(conflict) —ситуация, когда при слиянии нескольких версий сделанные в них изменения пересекаются между собой.



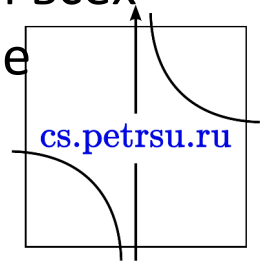
# Функции СКВ

- **хранение несколько версий** одного и того же документа (история версий);
- хранение истории разработки;
- при необходимости **возвращение к более ранним версиям** документа (отмена изменений);
- определение, **кто и когда сделал изменение** (поиск «виновного»);
- **совмещение изменений** сделанных разными разработчиками (синхронизация работы команды);
- реализация альтернативных/эксперементальных вариантов проекта.



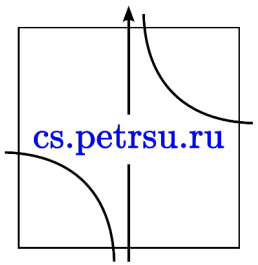
# Область применения

- **Разработка программного обеспечения** -хранение исходных кодов разрабатываемой программы.
- Области, в которых ведётся работа с большим количеством **непрерывно изменяющихся электронных документов**:
  - В **инструментах конфигурационного управления**(Software Configuration Management Tools).
  - В САПР, обычно в составе **систем управления данными об изделии**(Product Data Management(PDM)).
  - Как пример –**Википедия** ведёт историю изменений для всех её статей, используя методы, аналогичные тем, которые применяются в системах управления версиями.



$\Rightarrow$ Требуется использовать ПО для облегчения работы с изменяющейся информацией

- Revision Control System (1984)
- CVS (1990)
- Rational ClearCase (IBM, 1992)
- Visual SourceSafe (Microsoft, 1994)
- Subversion (2000)
- Arch (децентрализованная CVS, 2002)
- Git (2005)
- Mercurial (2005)



prehistory

scm classic times

middle ages

the Renaissance

1982 1990 1992 1994 1995 2000 2005 2010

rcs

cvs

pvcs

clearcase

vss

welcome to hell

perforce

fear of branching ...

svn

accurev

scm redone...

bitkeeper

bazaar

git

darcs

hg

dvcs age

plastic

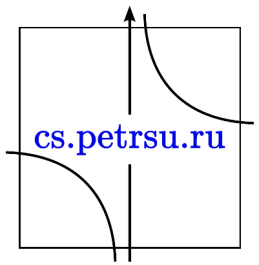
embrace branching, merging and distributed...

laggards

tfs

# Архитектура

- **Локальные** системы контроля версий
  - RCS
- **Централизованные** системы контроля версий
  - CVS, Subversion
- **Децентрализованные** системы контроля версий
  - Git, Mercurial, Bazaar



# Модель хранения

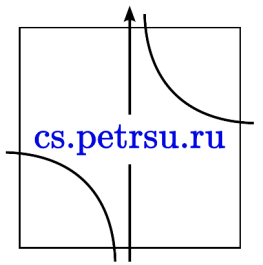

- **Моментальный снимок**
  - **V1.1** **V1.2** **V1.3**
- **Набор изменений**
  - **V1.1** **Δ1** **Δ2**

cs.petrsu.ru

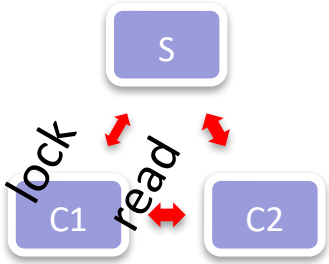
# Модель изменений

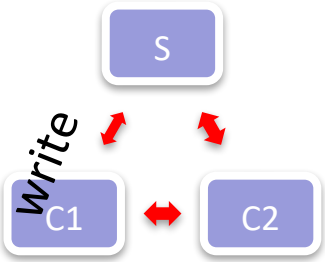
- **Блокирующие** — позволяют наложить запрет на изменение файла, пока один из разработчиков работает над ним.

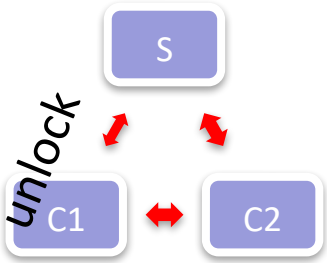
- **Не блокирующие** — один файл может одновременно изменяться несколькими разработчиками.



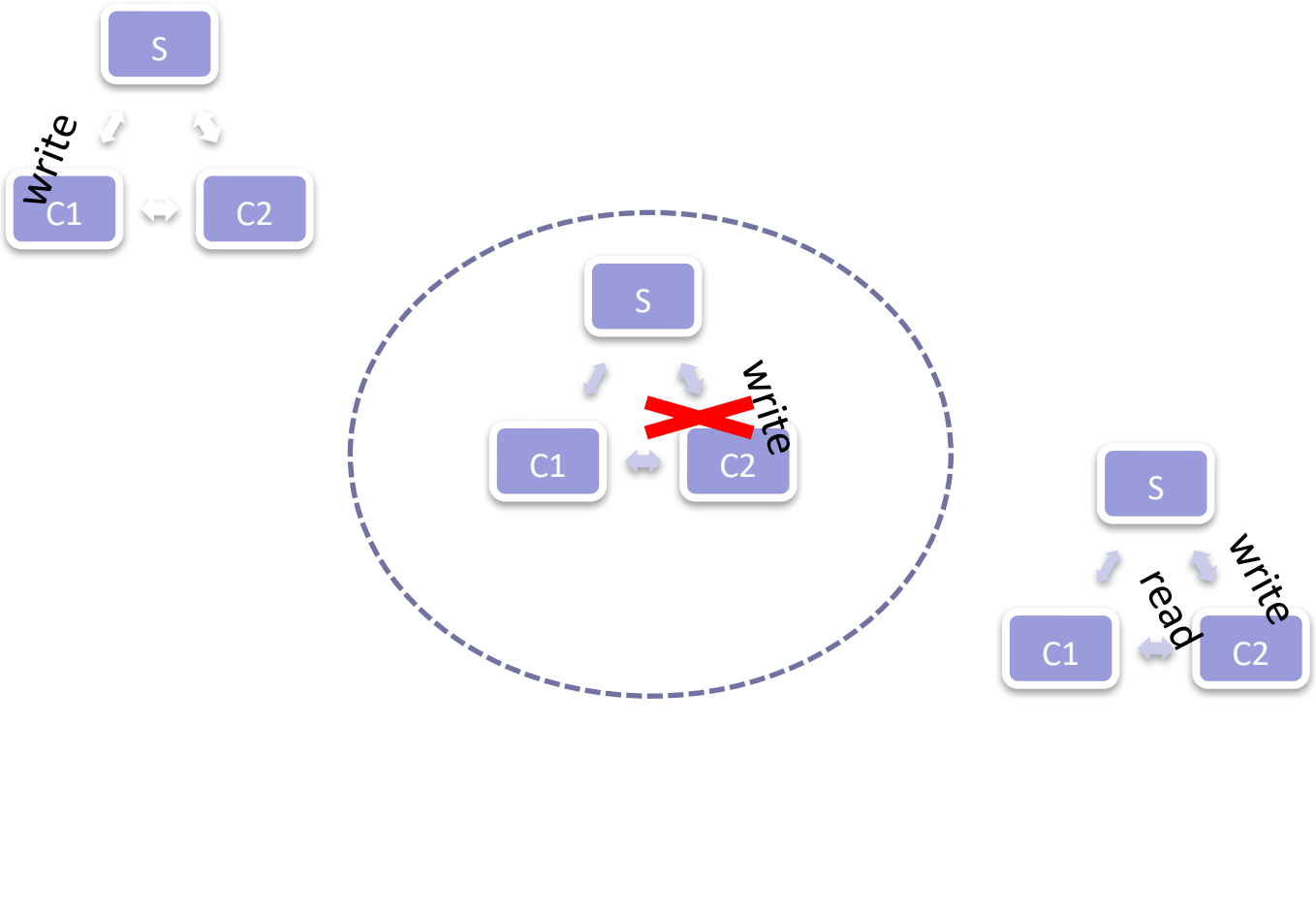
# Блокировка – изменение - разблокировка



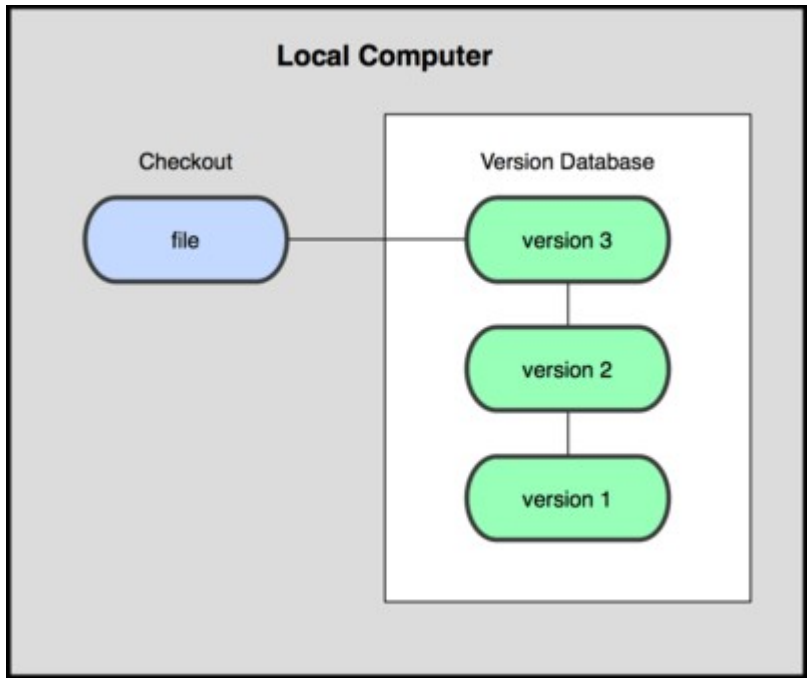



# Копирование – изменение - слияние



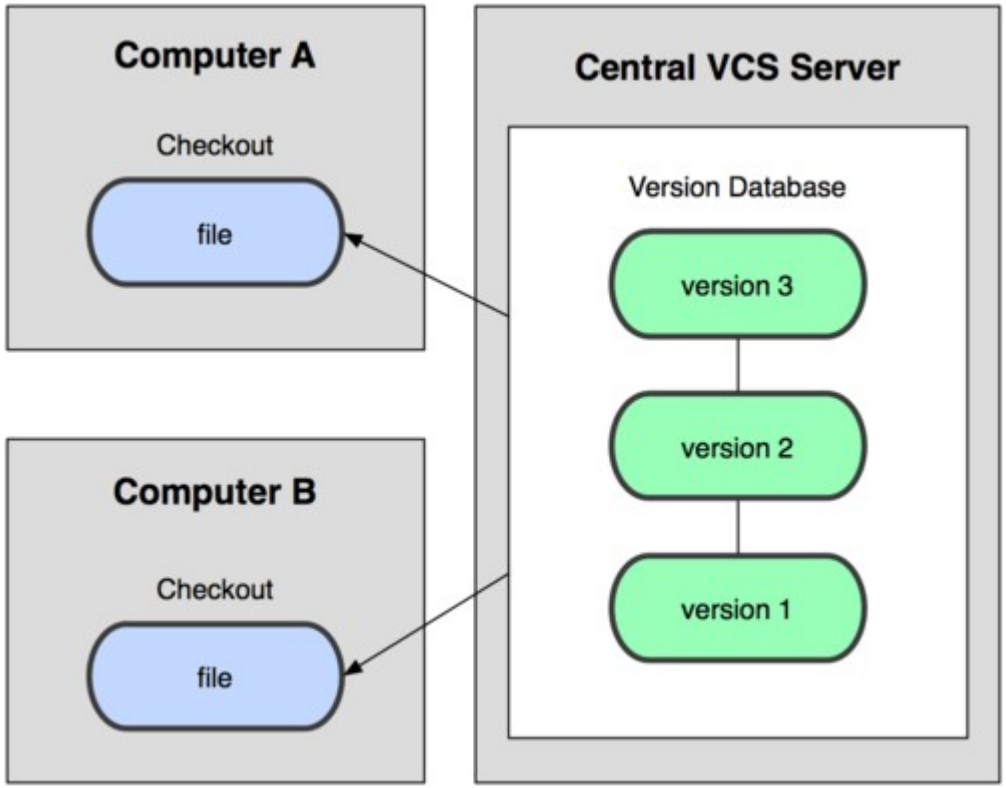
# Локальные СКВ

- Основываются на простой базе данных в которой хранятся изменения нужных фалов.



# Централизованные СКВ

- Есть **центральный сервер**, на котором хранятся все файлы под версионным контролем

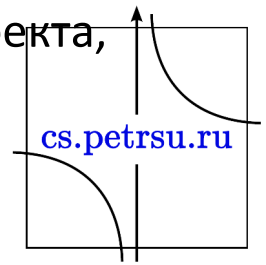


**Преимущества:**

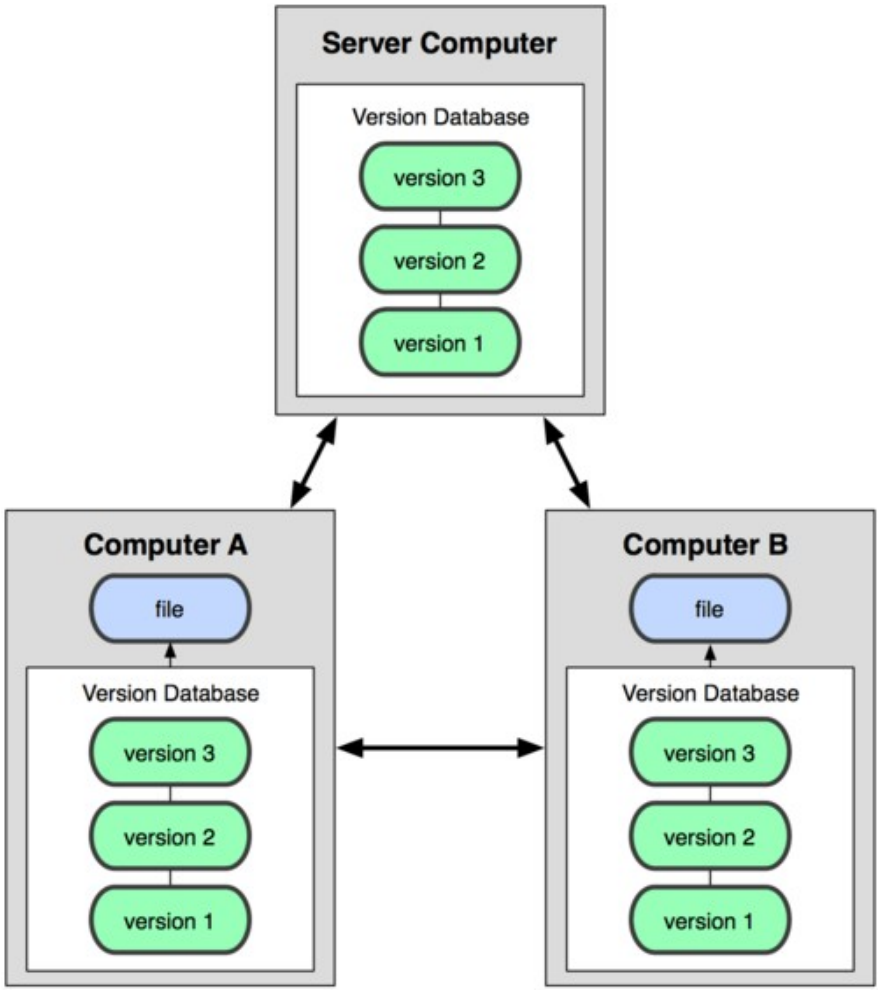
- все знают, кто и чем занимается в проекте;
- у администраторов есть чёткий контроль над тем, кто и что может делать,

**Недостатки:**

- **централизованный сервер является уязвимым местом** всей системы:
  - если **сервер не работает**, то **разработчики не могут взаимодействовать**, и никто не может сохранить новой версии своей работы;
  - если **отсутствует подключение сети у разработчика**, в это время он также **не может взаимодействовать** с сервером;
  - если же **повреждается диск с центральной базой данных** и нет резервной копии, вы **теряете абсолютно всё**—всю историю проекта, разве что **за исключением нескольких рабочих версий, сохранившихся на рабочих машинах пользователей**.



# Распределённые СКВ




- В отличие от централизованных систем клиенты не просто выгружают последние версии файлов, а **полностью копируют весь репозиторий**.

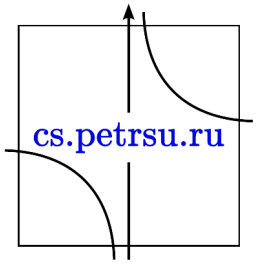
cs.petrsu.ru

**Преимущества:**

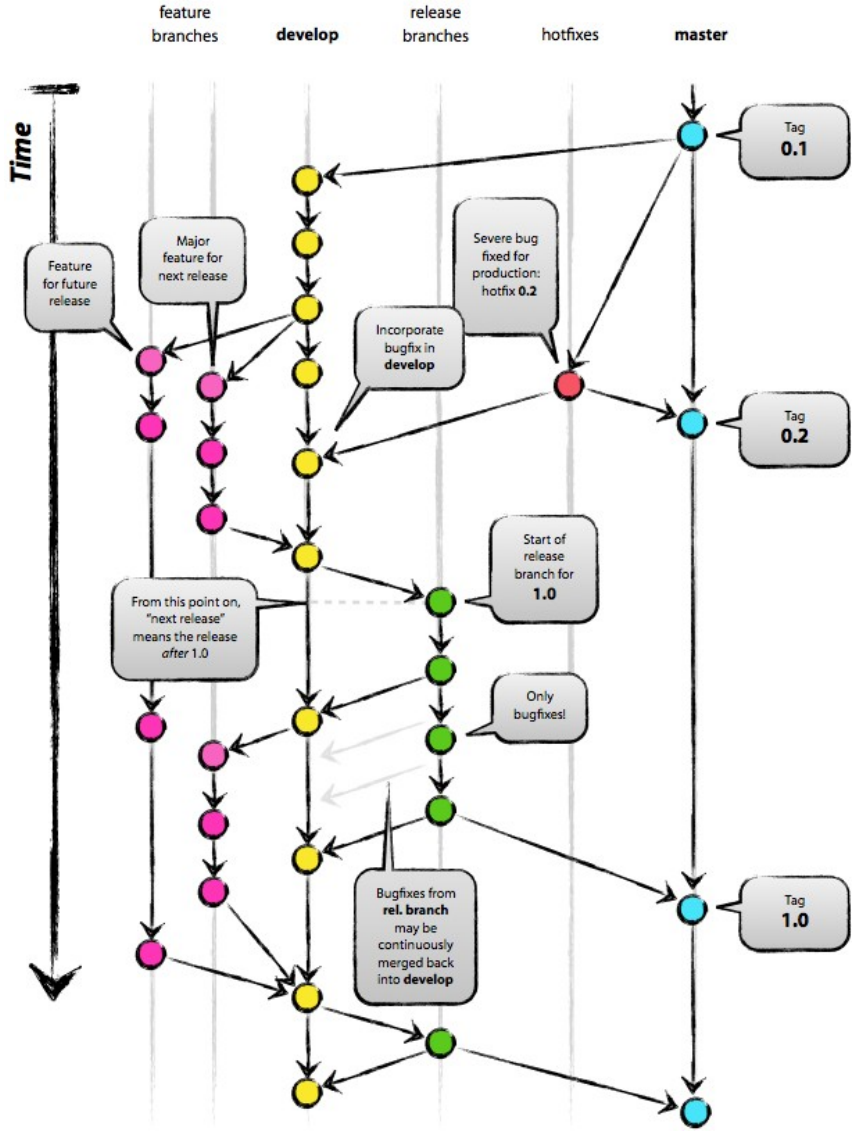
- Когда клиент забирает свежую версию файлов, он создаёт себе полную копию всех данных. В случае сбоев на сервере любой **клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных**.
- Возможность **работать с несколькими удалёнными репозиториями**, можно одновременно работать по-разному с разными группами людей в рамках одного проекта.

**Недостатки:**

- Большой объем передаваемых данных
- Сложности синхронизации удаленных репозиториев
- Сложность контроля работ



# Пример эволюции ветвей в



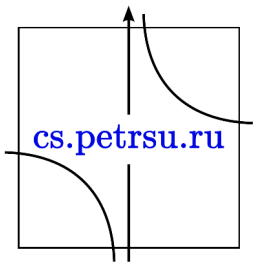

cs.petrsu.ru

**Главные ветки:**

- Ствол (trunk, mainline, **master**) —основная ветвь разработки проекта.
  - Создается при инициализации репозитория.
  - В любой момент времени в в ней должен быть только Production-Ready код.
  - Считается главной, интеграционной.
  - Когда код в ветке develop становится пригодным для релиза, его интегрируют в master и помечают тегом версии.
- Develop
  - Создается для основной деятельности по разработке

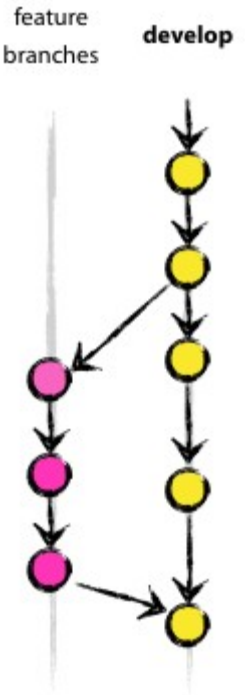
**Дополнительные ветки:**

- Ветки функциональностей (Feature branches)
- Ветки релизов (Release branches)
- Ветки исправлений (Hotfix branches)



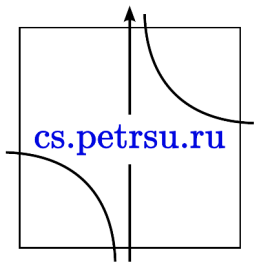
# Ветки функциональностей (Feature branches)

- Могут порождаться от develop.
- Должны интегрироваться в develop.
- Используются для разработки новых функций, которые должны появиться в текущем или следующем релизе.
- Существуют столько, сколько разрабатывается функция(feature).
- Как правило существует в репозитории разработчиков, а не в центральном.



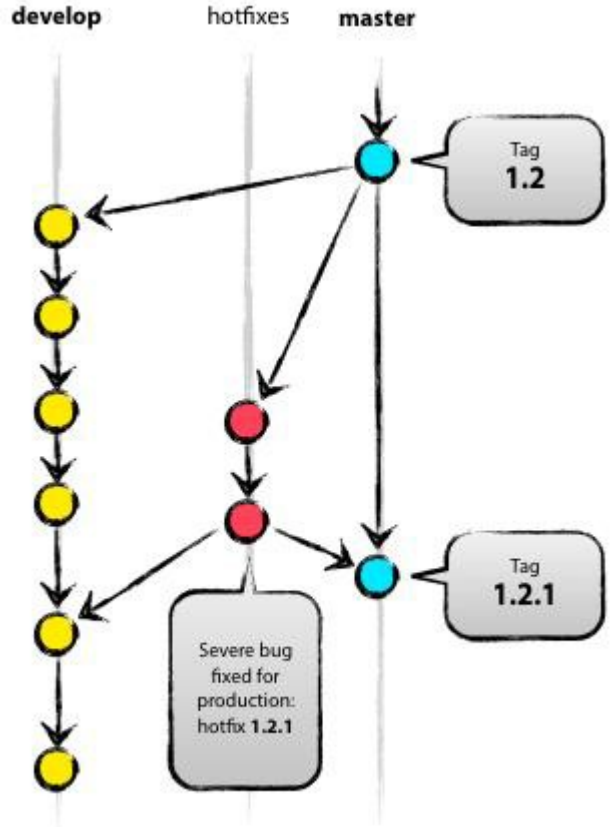
# Ветки релизов (Release branches)

- Могут порождаться от develop.
- Должны интегрироваться в develop, master.
- Название: release-*.
- Используются для подготовки версий к релизу.



# Ветки исправлений (Hotfix branches)

- Могут порождаться от master.
- Должны интегрироваться в develop, master.
- Название: hotfix-*.
  - Похожи на релизные ветки, отличие в том, что они не запланированные.
  - Создаются в момент, когда необходимо срочно исправить ошибку в Production-Readyкоде.
  - Команда может работать над developверсией, а в это время в ветке исправлений кто-то пытается быстро устранить найденную ошибку.

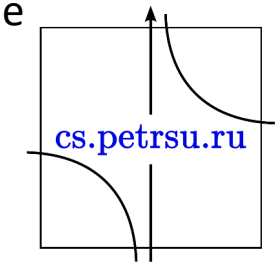


# RCS, 1984

## Набор скриптов для работы с файлами

- Одна из первых систем контроля версий.
  - Пришла на смену SCCS (Source Code Control System – система управления исходным кодом).
- Вытеснена более мощной системой контроля версий CVS
  - но все еще – используется(является частью проекта GNU).
- Позволяет работать только с отдельными файлами, создавая для каждого историю изменений.
  - Для текстовых файлов сохраняются не все версии файла, а только последняя версия и все изменение, внесенные в нее.
  - Может отслеживать изменения в бинарных файлах, каждое изменение хранится в виде отдельной версии файла.
- Когда изменения в файл вносит один из пользователей,

  для всех остальных этот файл остается заблокированным.

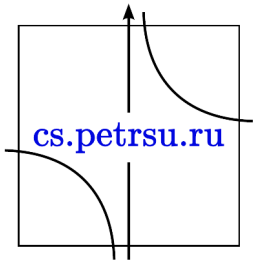


**Преимущества:**

- RCS -проста в использовании и хорошо подходит для ознакомления с принципами работы систем контроля версий.
- Хорошо подходит для резервного копирования отдельных файлов, не требующих частого изменения группой пользователей.
- Широко распространена и предустановленнав большинстве свободно распространяемых операционных системах.

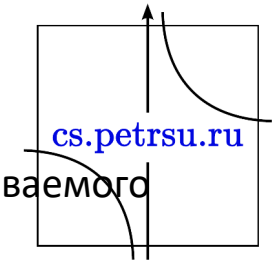
**Недостатки:**

- Отслеживает изменения только отдельных файлов, что не позволяет использовать ее для управления версиями больших проектов.
- Не позволяет одновременно вносить изменения в один и тот же файл несколькими пользователями.
- Низкая функциональность, по сравнению с современными системами контроля версий.

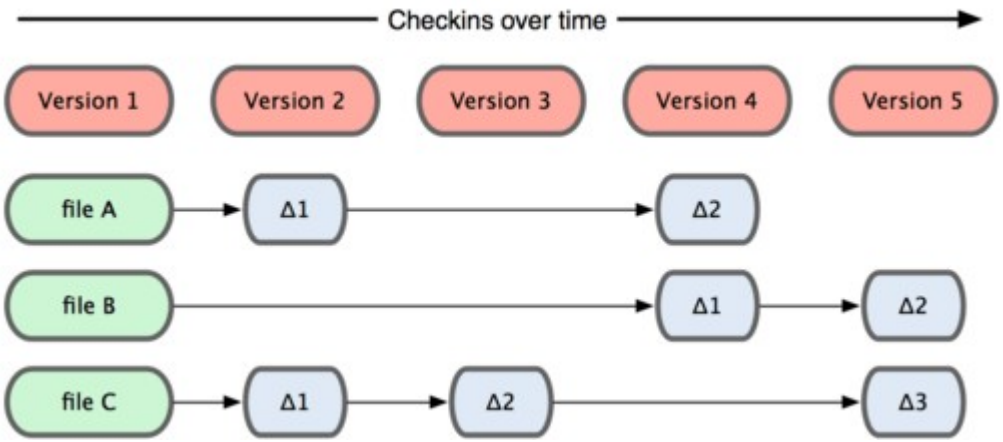


# CVS, 1990

- Система управления параллельными версиями (Concurrent Versions System) - логическое развитие системы RCS, использующая ее стандарты и алгоритмы по управлению версиями, но значительно более функциональная, и позволяющая работать не только с отдельными файлами, но и с целыми проектами.
- Основана на технологии клиент-сервер, взаимодействующих по сети. Клиент и сервер также могут располагаться на одной машине, если над проектом работает только один человек, или требуется вести локальный контроль версий.
- Работа организована следующим образом.
  - Последняя версия и все сделанные изменения хранятся в репозитории сервера. Клиенты, подключаясь к серверу, проверяют отличия локальной версии от последней версии, сохраненной в репозитории, и, если есть отличия, загружают их в свой локальный проект.
  - При необходимости решают конфликты и вносят требуемые изменения в разрабатываемый продукт.
  - После этого все изменения загружаются в репозиторий сервера.
  - При необходимости, позволяет откатываться на нужную версию разрабатываемого проекта.

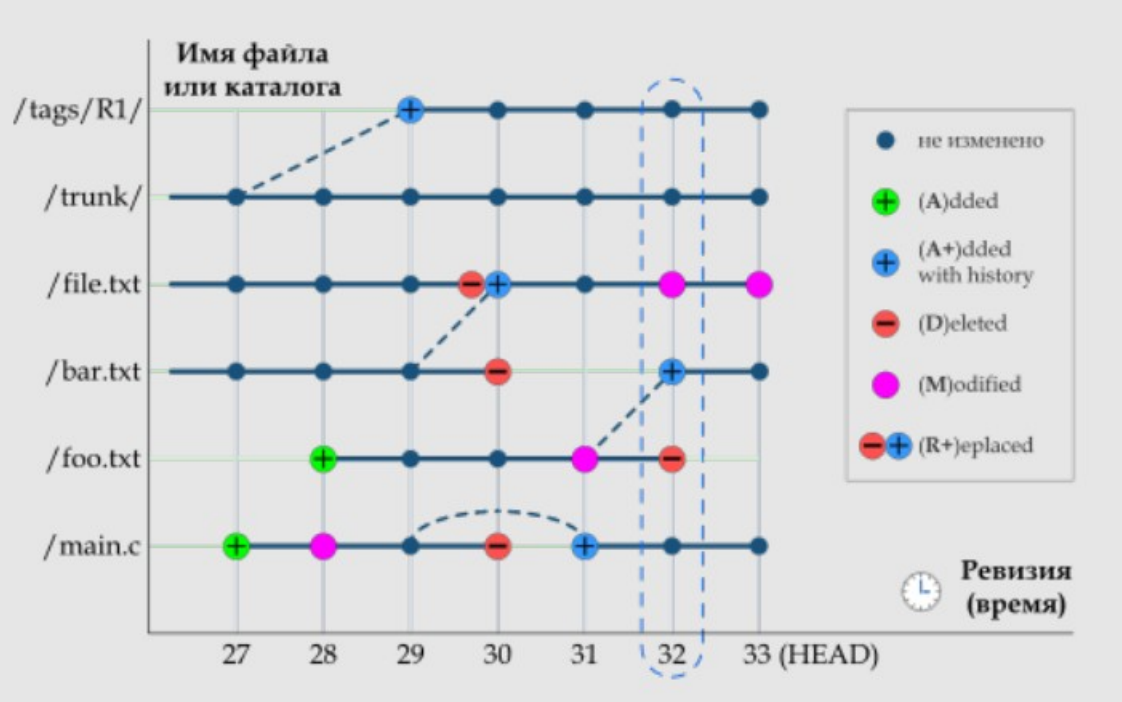


- CVS **хранит информацию как список изменений (патчей) для файлов**. Т.е. относится к хранимым данным как к **набору файлов и изменений, сделанных для каждого из этих файлов во времени**



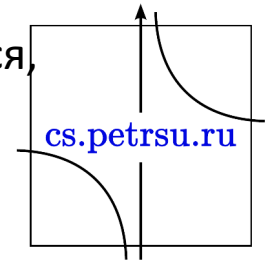
# Subversion, 2000

- Централизованная система управления версиями основанная на технологии клиент-сервер. Обладает всеми достоинствами CVS и решает основные ее проблемы (переименование и перемещение файлов и каталогов, работа с двоичными файлами и т.д.).
- Принцип работы похож на CVS. Клиенты копируют изменения из репозитория и объединяют их с локальным проектом пользователя.
  - Если возникают конфликты локальных изменений и изменений, сохраненных в репозитории, то такие ситуации разрешаются вручную.
  - Затем в локальный проект вносятся изменения, и полученный результат сохраняется в репозитории.
- При работе с файлами, не позволяющими объединять изменения, может использоваться следующий принцип:
  - Файл скачивается из репозиторияи блокируется (запрещается его скачивание из репозитория).
  - Вносятся необходимые изменения.
  - Загружается файл в репозиторий и разблокируется (разрешается его скачивание из репозитория другим клиентам).
- Из-за широкой функциональности, простоты и схожести в управлении успешно вытесняет CVS .


Имя файла
или каталога
/tags/R1/
/trunk/
/file.txt
/bar.txt
/foo.txt
/main.c
не изменено
(A)dded
(A+)dded
with history
(D)eleted
(M)odified
(R+)eplaced
Ревизия
(время)
27
28
29
30
31
32
33 (HEAD)

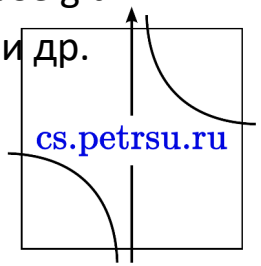
# GIT, 2005

- 3 апреля 2005 года Линус Торвальдс приступил к разработке собственной системы контроля версий – Git.
- Git–это гибкая, распределенная (без единого сервера) система контроля версий, дающая массу возможностей не только разработчикам программных продуктов, но и писателям для изменения, дополнения и отслеживания изменения «рукописей» и сюжетных линий, и учителям для корректировки и развития курса лекций, и администраторам для ведения документации, и для многих других направлений, требующих управления историей изменений.
- У каждого разработчика, использующего Git, есть свой локальный репозиторий, позволяющий локально управлять версиями. Затем, сохраненными в локальный репозиторий данными, можно обмениваться с другими пользователями.
- Часто при работе с Git создают центральный репозиторий, с которым остальные разработчики синхронизируются.
- Наличие локальных репозиториев значительно повышает надежность хранения данных
- Работа над версиями проекта в Git может вестись в нескольких ветках, которые затем могут с легкостью полностью или частично объединяться, уничтожаться, откатываться и разрастаться во все новые и новые ветки проекта.



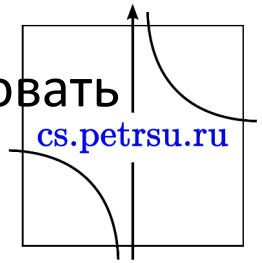
# Достоинства Git

- Надежная система сравнения ревизий и проверки корректности данных, основанные на алгоритме хеширования SHA1 (SecureHashAlgorithm1).
- Гибкая система ветвления проектов и слияния веток между собой.
- Наличие локального репозитория, содержащего полную информацию обо всех изменениях, позволяет вести полноценный локальный контроль версий.
- Высокая производительность и скорость работы.
- Удобный и интуитивно понятный набор команд.
- Множество графических оболочек, позволяющих быстро и качественно вести работы с Git’ом.
- Возможность делать контрольные точки, в которых данные сохраняются без дельта компрессии, а полностью. Это позволяет уменьшить скорость восстановления данных, так как за основу берется ближайшая контрольная точка, и восстановление идет от нее.
- Широкая распространенность, легкая доступность и качественная документация.
- Гибкость системы позволяет удобно ее настраивать и даже создавать специализированные контроля системы или пользовательские интерфейсы на базе git.
- Универсальный сетевой доступ с использованием протоколов http, ftp, rsync, ssh и др.

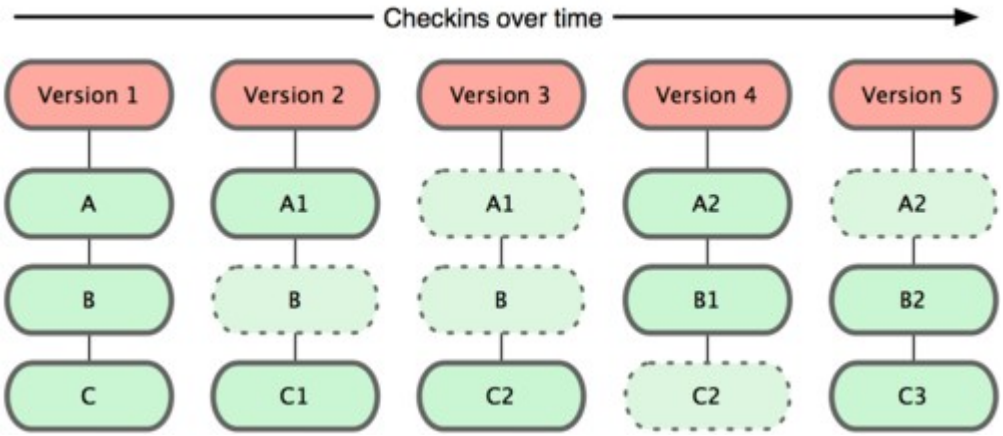


# Недостатки Git

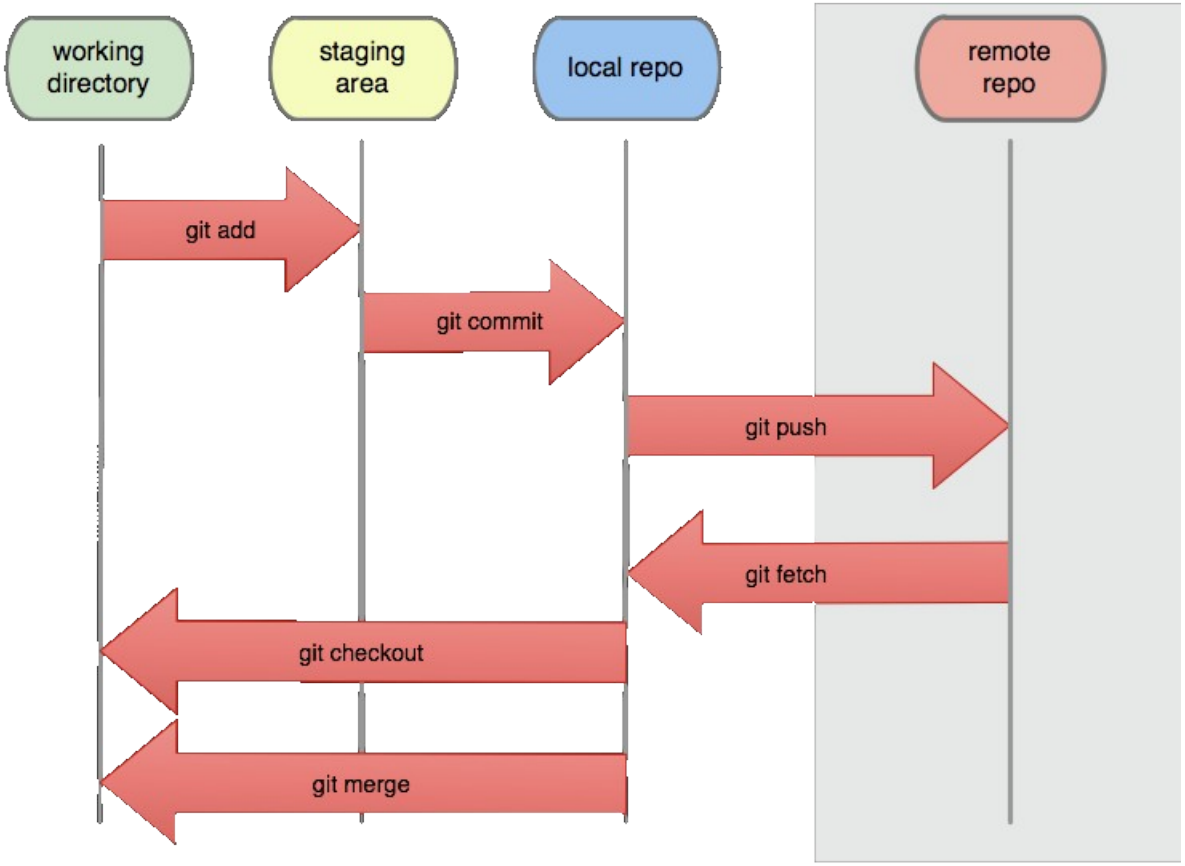
- Unix–ориентированность. На данный момент отсутствует зрелая реализация Git, совместимая с другими операционными системами.
- Возможные (но чрезвычайно низкие) совпадения хеш-кода отличных по содержанию ревизий.
- Не отслеживается изменение отдельных файлов, а только всего проекта целиком, что может быть неудобно при работе с большими проектами, содержащими множество несвязных файлов.
- При начальном (первом) создании репозитория и синхронизации его с другими разработчиками, потребуется достаточно длительное время для скачивания данных, особенно, если проект большой, так как требуется скопировать весь репозиторий.



- Git считает **хранимые данные набором слепков** небольшой **файловой системы**.
- При фиксации текущей версии проекта, Git сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Ради эффективности, если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохранённый файл.



# Схема работы




cs.petrsu.ru

# Команды Git

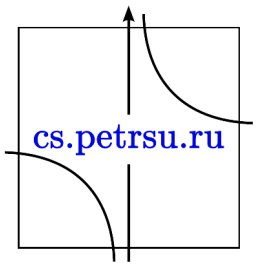
- Задать конфигурации

> git config --global --list

> git config --global user.name “Test.User”

> git config --global user.email “user@test.com”

- Определить протокол передачи данных
  - FILE
  - SSH
  - HTTP(S)



- Клонировать или создать новый репозиторий или обновить текущий

> git init

> git clone git@github.com:user/repo.git

> git fetch – получение изменений с сервера

> git pull – аналог fetch+merge

- Проверка состояния проекта

> git status

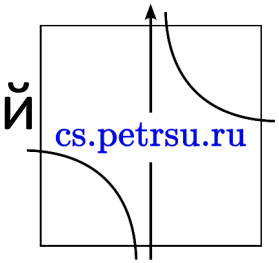
- Внесение изменений и их добавление

> git add README

> git commit –m “Add README file”

- Отправка изменений в центральный репозиторий

> git push



> git checkout –b new33  - переключение на новую ветку

> git checkout master – переключение на ветку “master”

> git merge new33 – объединение веток

> git log – получение списка изменений

> git hist – получение хешей предыд. Версий

> git reset – отмена индексации изменений

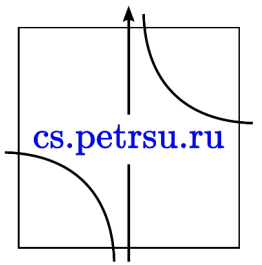
> git revert – отмена коммитов

> git diff – сравнение изменений

> git rm – удаление файла/директории

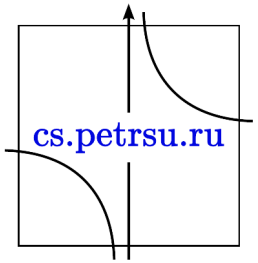
> git branch – управление ветками

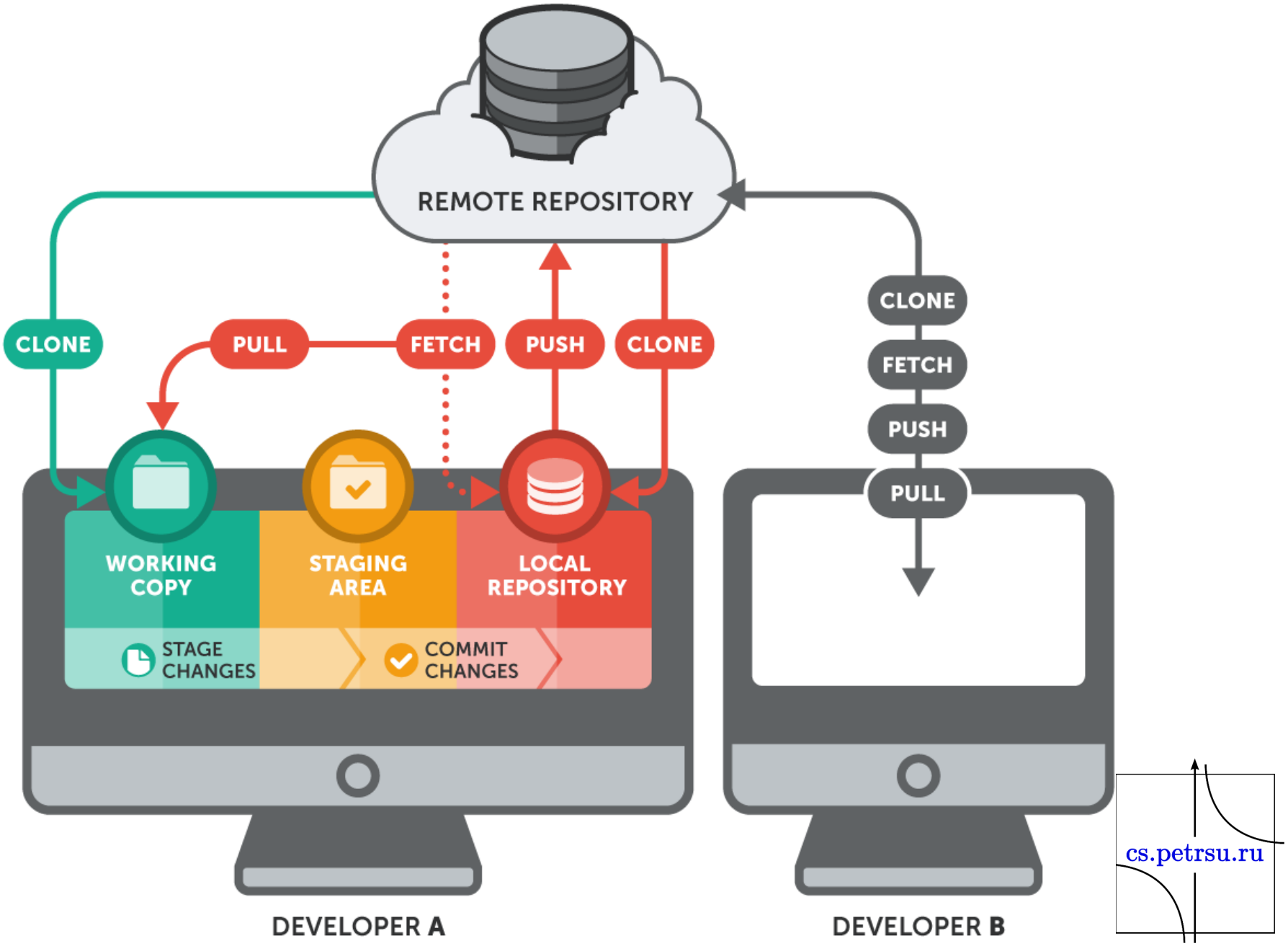
> git remote – управление удалёнными репоз.



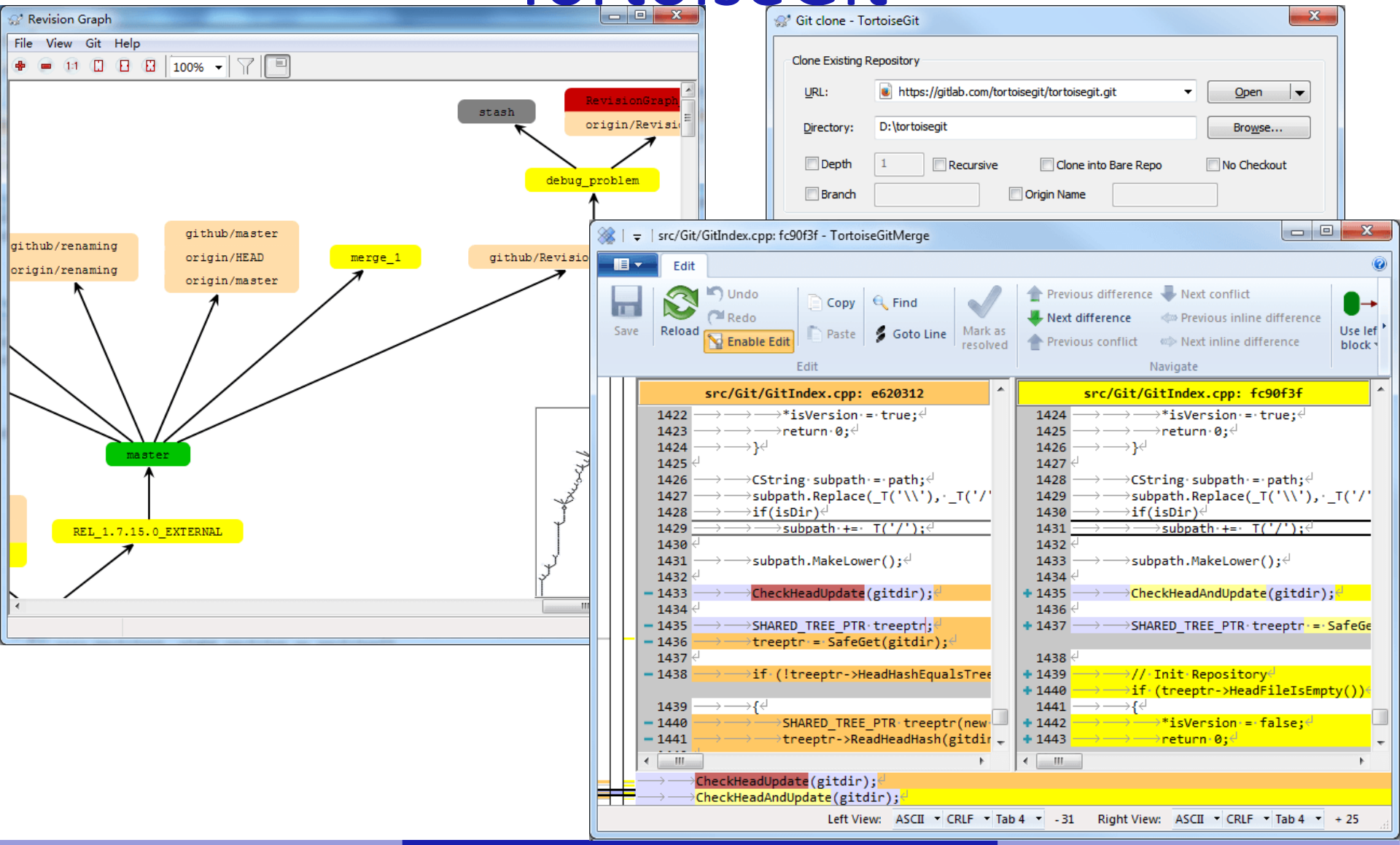
# Цикл работы

- Обновление репозитория и рабочей копии
  - **git pull**
- Добавление файла в проект
  - **git add myfile.c**
- Коммит
  - **git commit –m “комментарий”**
- Передача изменений во внешний репозиторий
  - **git push**



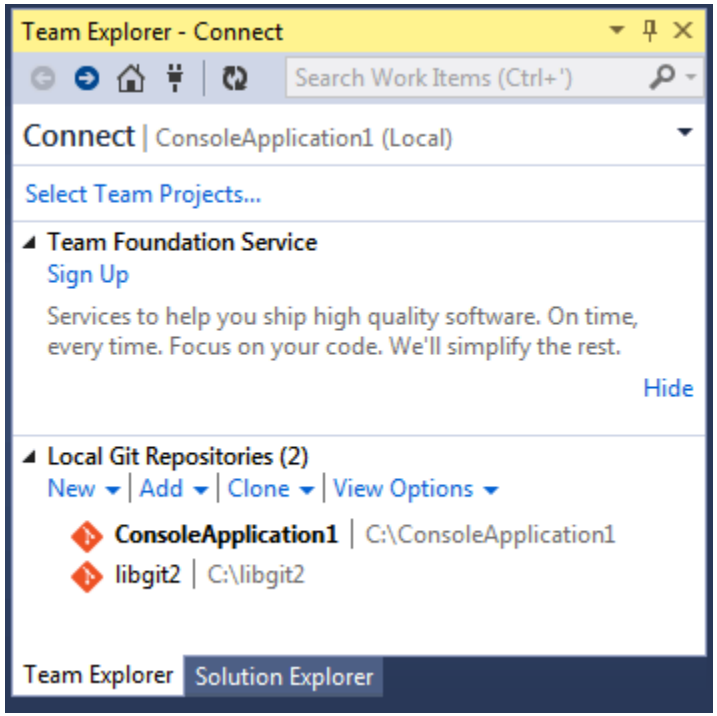

REMOTE REPOSITORY
CLONE
PULL
FETCH
PUSH
CLONE
CLONE
FETCH
PUSH
PULL
WORKING COPY
STAGING AREA
LOCAL REPOSITORY
STAGE CHANGES
COMMIT CHANGES
DEVELOPER A
DEVELOPER B

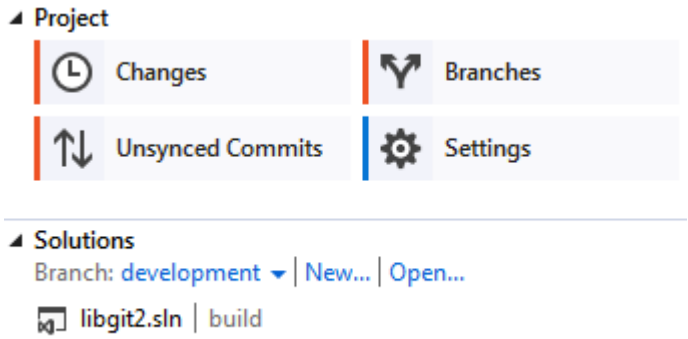
# TortoiseGit



# Git в Visual Studio

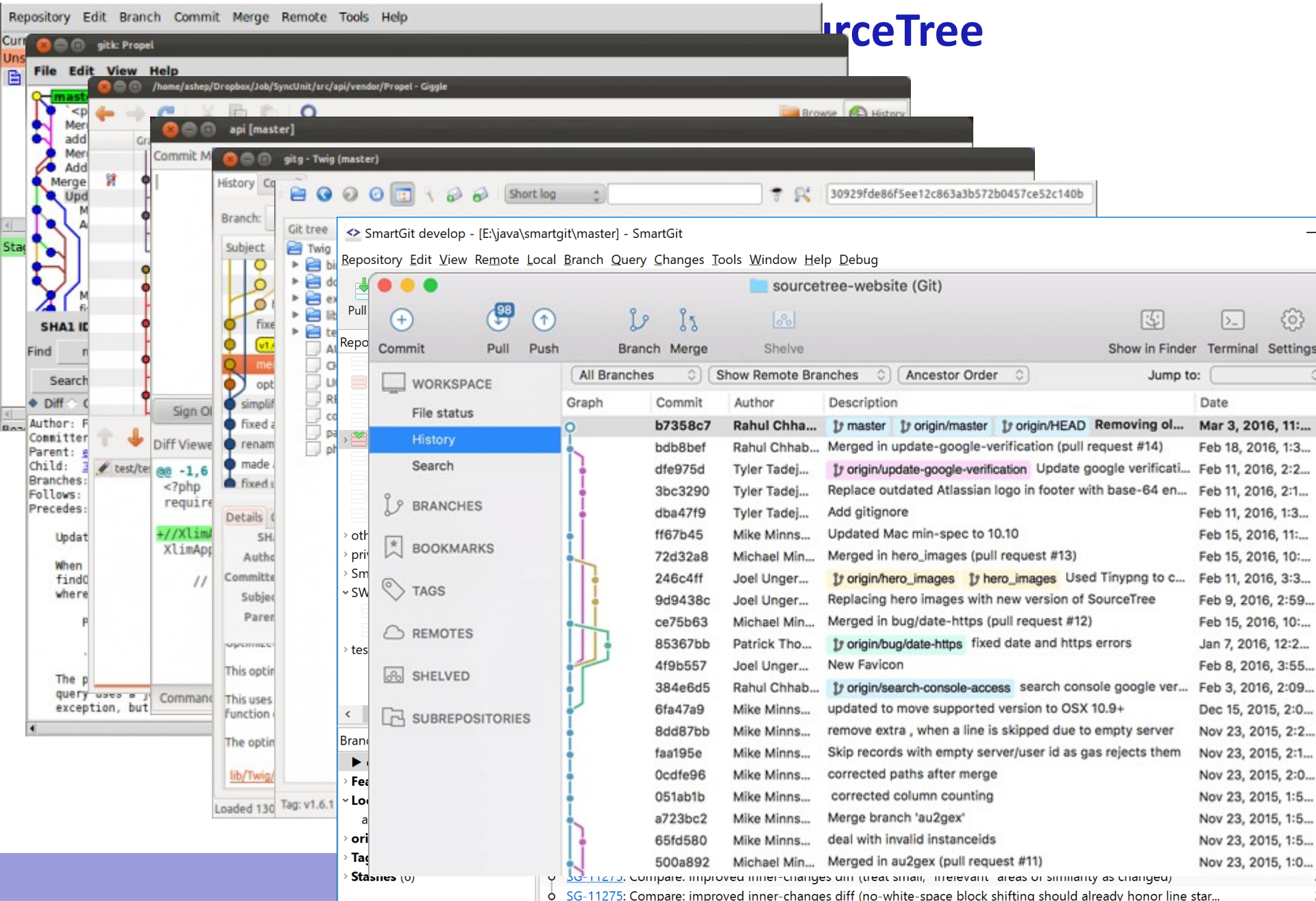
- Начиная с Visual Studio 2013 Update 1, пользователям Visual Studio доступен Git-клиент, встроенный непосредственно в IDE.
- Чтобы воспользоваться – откройте проект, который управляется Git (или выполните git init для существующего проекта) и выберите пункты View (Вид) > Team Explorer (Командный обозреватель) в главном меню. В результате откроется окно "Connect" ("Подключить"), которое выглядит примерно вот так:



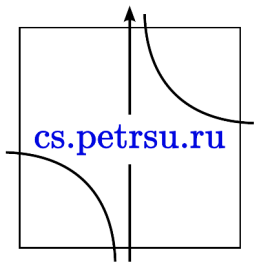

"Home" ("Главная") страница Git-репозитория в Visual Studio.

# git-gui, gitk, Giggle, Git Cola, gitg, Qgit ...rceTree



# Сервисы git-репозиториев

- **GitHub** https://github.com/
- **Google Code** http://code.google.com/intl/ru-RU/
- **Bitbucket** https://bitbucket.org/
- Codebase
- SourceForge
- Gitorious



# Полезные ссылки

- http://git-scm.com/book/ru/
- http://githowto.com/ru
- Пример с работы с Git
  http://habrahabr.ru/post/174467/
- Пример с ветвлениями:
  https://habr.com/post/106912/

# Заключение

- Git–гибкая, удобная и мощная система контроля версий, способная удовлетворить абсолютное большинство пользователей.
- Существующие недостатки постепенно удаляются и не приносят серьезных проблем пользователям. Если вы ведете большой проект, территориально удаленный, и тем более, если часто приходится разрабатывать программное обеспечение, не имея доступа к другим разработчикам (например, вы не хотите терять время при перелете из страны в страну или во время поездки на работу), можно делать любые изменения и сохранять их в локальном репозитории, откатываться, переключаться между ветками и т.д.).
- Git–один из лидеров систем контроля версий.