Совместная работа. GIT

...

github, gitlab

Корпоративная информационная система

Много разработчиков

- договоренности о правилах написания кода

Модульная разработка

Обеспечение совместной работы

- где последняя актуальная версия
- какие есть альтернативные версии
- как объединить версии (и решить конфликты)

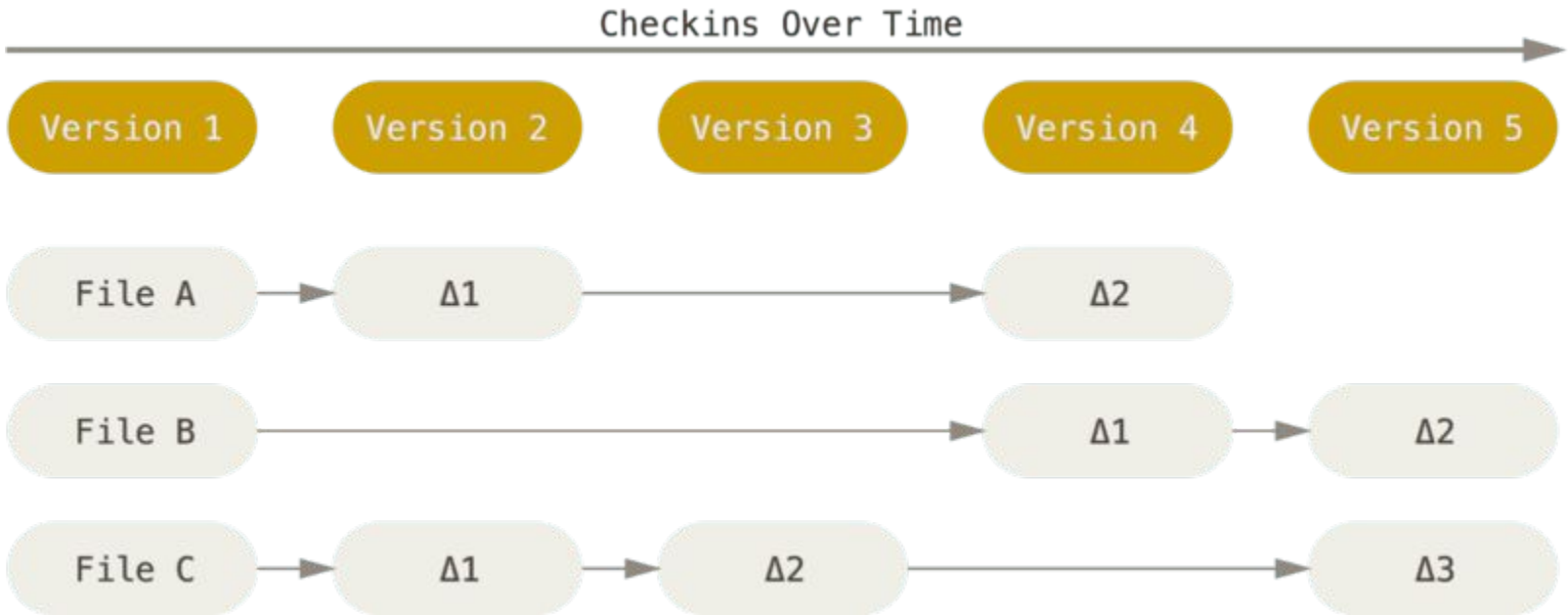
Обеспечение совместной работы

Нужна система управления версиями

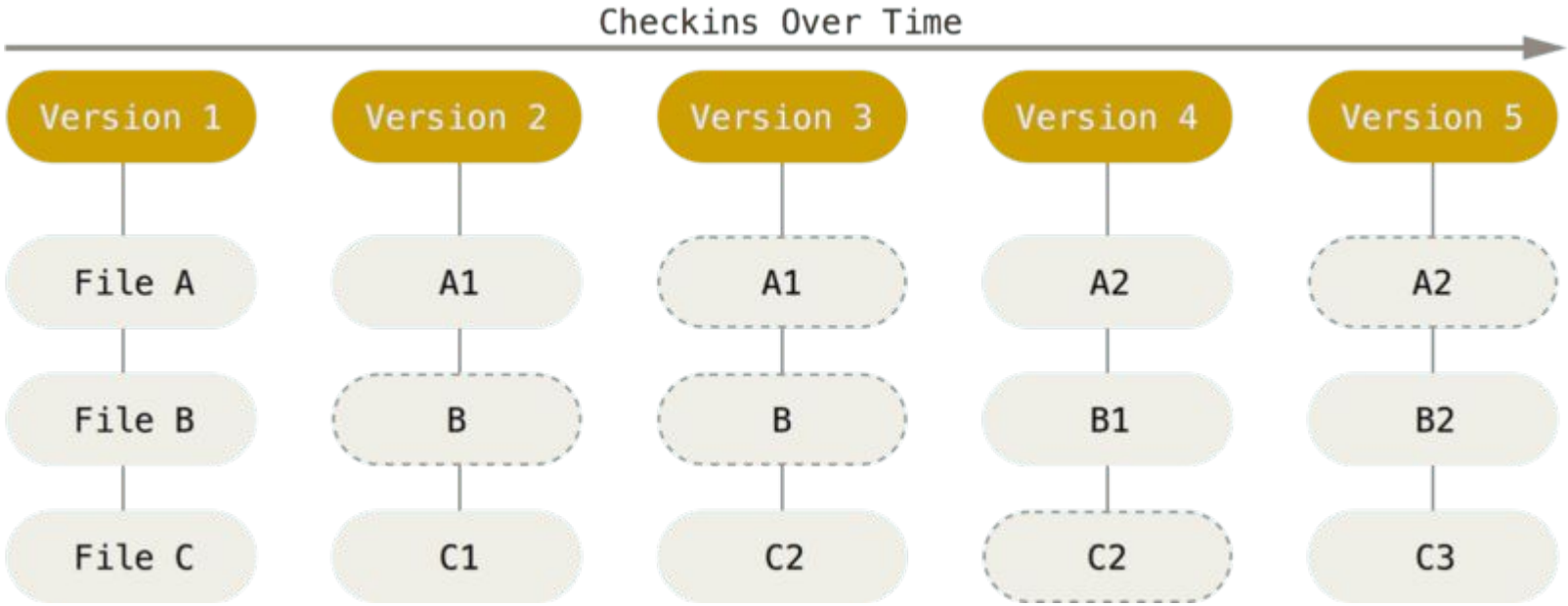
git

TortoiseSVN

Варианты учета версий (SVN, хранение дельт - разниц между файлами)



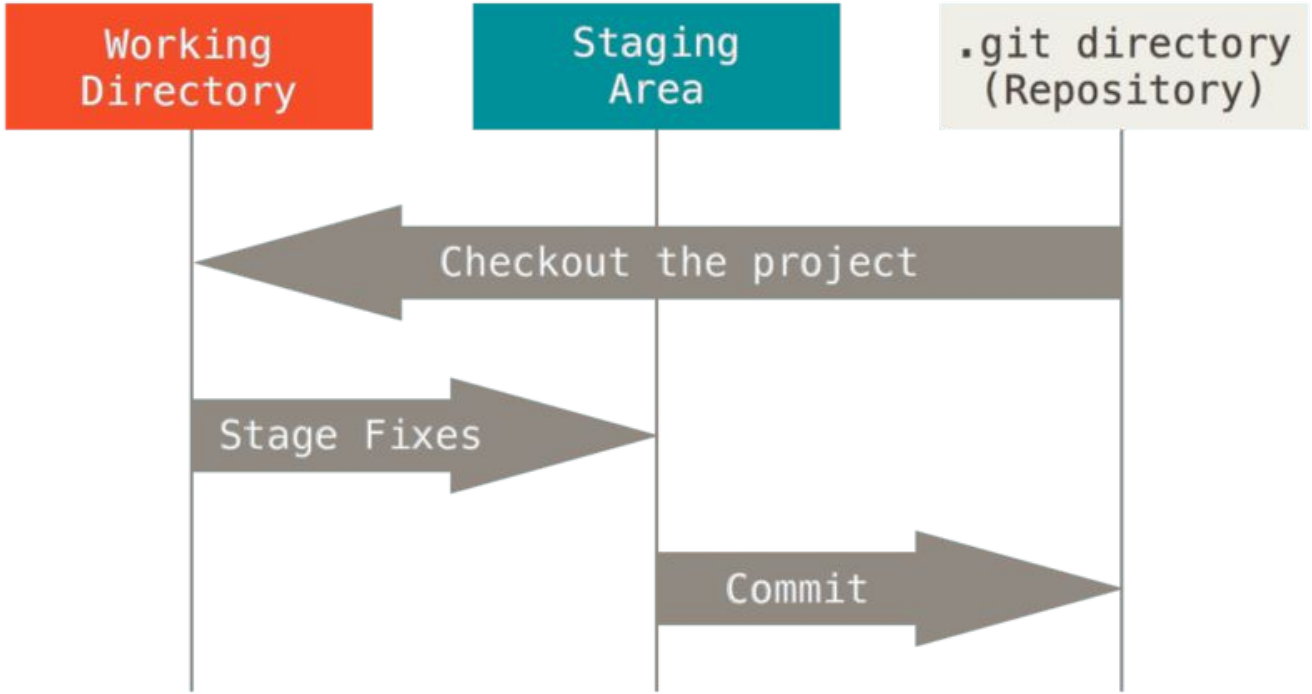
Варианты учета версий (GIT, хранение snapshots- “снимков” файлов)



распределённая система управления версиями.
Появилась в 2005 году для управления распределенной разработкой Linux.
Автор - Линус Торвальдс.

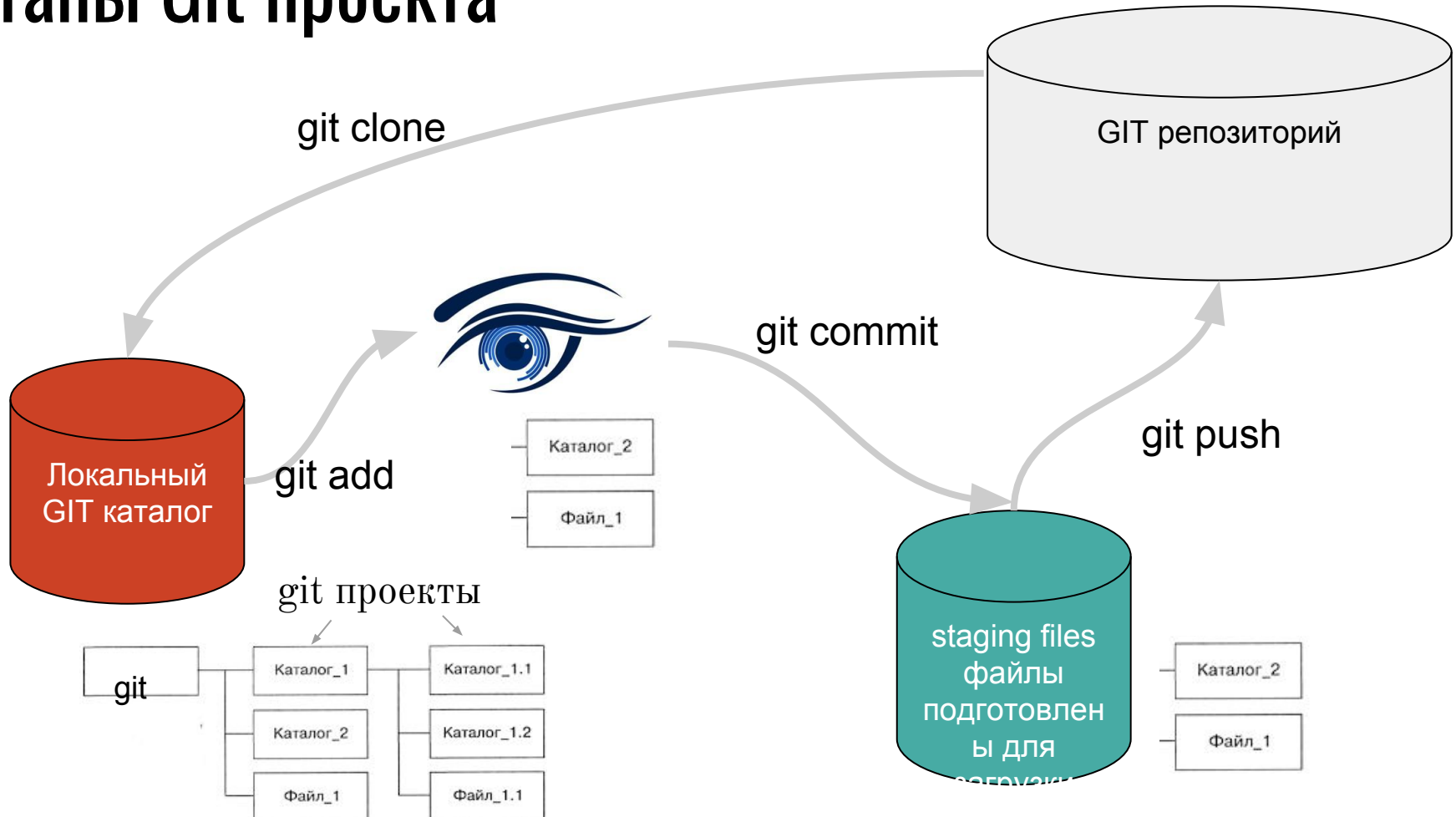
https://git-scm.com/book/en/v2

Пространства Git проекта

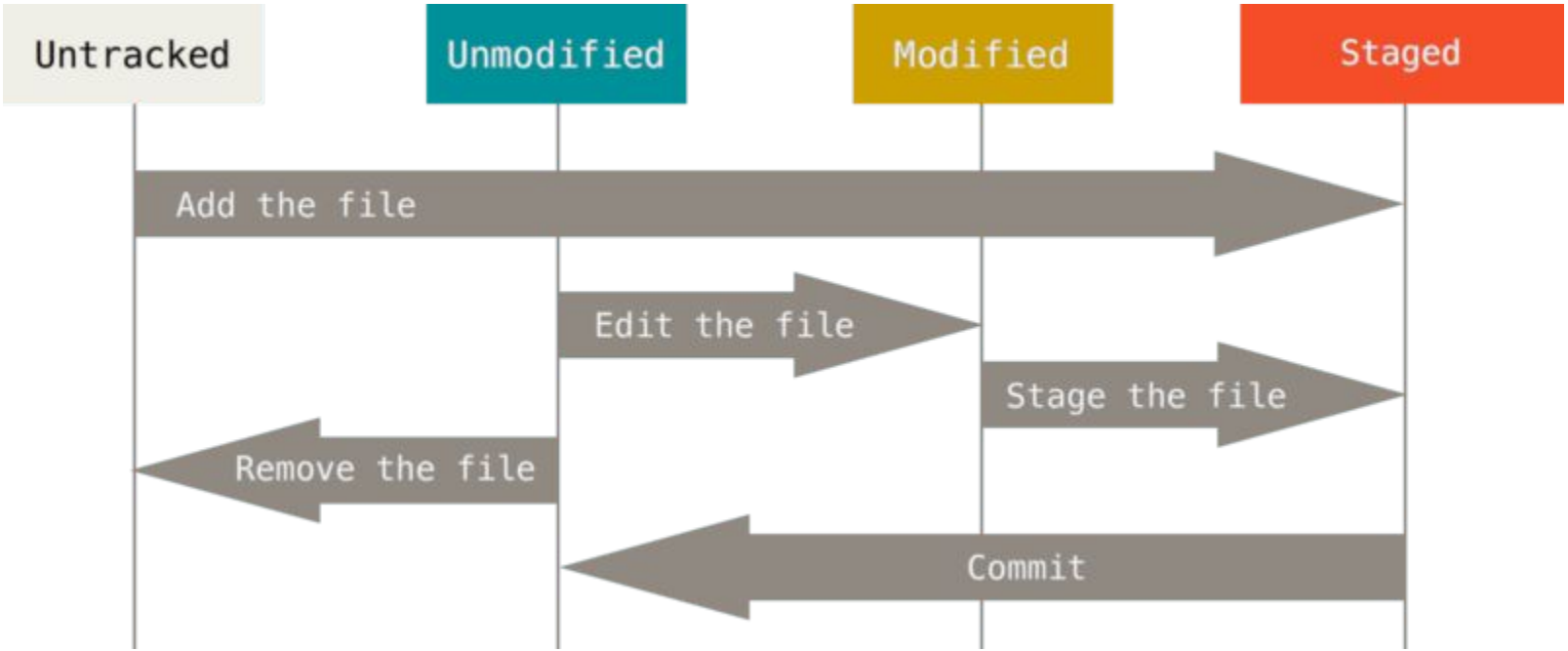


https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

Этапы Git проекта



Типы состояний файлов в Git проекте



https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

Работа с Git из командной строки

Инсталляция Git

Unix

`$ sudo dnf install git-all` или

`$ sudo apt install git-all`

Windows

Скачать с `http://git-scm.com/download/win`

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Первый запуск Git

Устанавливаем имя пользователя и адрес, куда будут отсылаться сообщения о действиях в Git

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Первый запуск Git (2)

Устанавливаем редактор по умолчанию

```
$ git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -nosession"
```

https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

Первый запуск Git (3)

Проверяем параметры

```
$ git config --list
```

Вопросы по Git

```
$ git help <verb>
$ man git-<verb>
```

```
$ git add -h
```

https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

Работа с Git. Шаг 1: получение репозитория

Инициализировать репозиторий в существующем каталоге

1) перейти в существующий каталог
2) в командной строке

$ git init

Клонировать готовый репозиторий

1) перейти в каталог, внутри которого будет размещаться рабочий каталог репозитория
2) $ git clone *URL-to-git*

Например:

$ git clone https://github.com/libgit2/libgit2

https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Работа с Git. Шаг 2: выбор файлов для отслеживания изменений

1) перейти в каталог репозитория git
2) в командной строке

```
$ git add <имена файлов через пробел или фильтр отбора файлов>
```

Например:

```
$ git add *.php
$ git add 1.php readme.txt
```

https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Работа с Git. Шаг 3: выбор файлов, которые не нужно отслеживать

1) перейти в каталог репозитория git
2) отредактировать файл .gitignore

Общее правило для файлов .gitignore - указать шаблоны файлов, которые НЕ нужно отслеживать

смотрите https://github.com/github/gitignore для примеров

Работа с Git. Шаг 3: правила .gitignore

- Пустые строки или строки, начинающиеся с #, игнорируются.
- Стандартные шаблоны ДОС применяются рекурсивно ко всему рабочему каталогу git.
- Шаблон, начинающийся с прямого слэша (/), означает - “не применять рекурсивно”.
- Шаблон, заканчивающийся прямым слэшем (/), означает - “применить к каталогу” to specify a directory.
- Отрицание шаблона задается, если перед шаблоном поставить знак восклицания (!).

смотрите https://github.com/github/gitignore для примеров

Работа с Git. Шаг 3: Примеры .gitignore

```
# игнорировать все файлы с расширением .a
*.a
# при этом отслеживать изменения файла lib.a
!lib.a
# игнорировать только файл TODO в текущем каталоге, без рекурсии,
т.е. например, subdir/TODO файл (файл TODO из подкаталога subdir)
будет отслеживаться
/TODO
# игнорировать все файлы в любом каталоге, который называется build
build/
# игнорировать doc/notes.txt, но не doc/server/arch.txt
doc/*.txt
```

смотрите https://github.com/github/gitignore для примеров

Работа с Git. Шаг 4: просмотр статуса изменений

1) в каталоге репозитория git в командной строке

```
$ git status
```

Если еще ничего не сделано (файлы не менялись)

```
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    new file:   1.php
    new file:   readme.txt
```

https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Работа с Git. Шаг 5: просмотр детального списка изменений, прошедших с последнего “коммита”

В каталоге репозитория git в командной строке

$ git diff - команда покажет все изменения, и отслеживаемые, и неотслеживаемые

$ git diff --staged - команда покажет все изменения в отслеживаемых (staged) файлах, произошедшие с момента последнего “коммита”

https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Работа с Git. Шаг 6: “коммит”

$ git commit -m “Сообщение, в котором описано, что изменилось”

команда собирает “в пакет” все изменения в отслеживаемых файлах, и снабжает этот пакет текстовым сообщением

Этот пакет называется “коммит”, commit

https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Работа с Git. Шаг 6: отменить “коммит”

```
$ git commit --amend
```

команда отменяет “коммит”, давая возможность внести новые изменения

https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Работа с Git. Шаг 6: изменить список отслеживаемых файлов

Вначале проверяем, что отслеживается

$ git status

Потом указываем, что <filename> отслеживать не нужно

$ git reset HEAD <filename>

команда “вычеркивает” из списка отслеживаемых файлов файл <filename>

https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Работа с Git. Шаг 7: “пуш”

о ветках (branch) - дальше

$ git push origin master

“коммит” отправляется в репозиторий git на сервер, причем в ветку master, и там становится доступен для всех

Общий вид: **$ git push <repo_name> <branch_name>**

Изменения, сделанные локально в отслеживаемых файлах, станут доступны всем

https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Работа с Git. Шаг 7: “пуш”. Имена репозиториев

Что такое origin в команде $ git push **origin** master ?

origin - имя удалённого репозитория по умолчанию, которое git присваивает репозиторию *URL-to-git*

Задать удалённое имя репозитория, отличающееся от origin, можно командой:

$ git remote add repo_name *URL-to-git*

Посмотреть список удалённых репозиториев можно

$ git remote -v

https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Работа с Git. Шаг 7: “пуш”.

Посмотреть информацию об удалённом репозитории <repo_name> (т.е. репозитории, хранящемся где-то на сервере) можно командой

```
$ git remote show <repo_name>
```

Переименовать/удалить репозиторий

```
$ git remote rename <old_repo_name> <new_repo_name>
```

https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Работа с ветками (branches)

Дерево версий

При совместной разработке программный продукт развивается от версии к версии.

Разные группы параллельно работают над усовершенствованием частей проекта.

Эти усовершенствования проверяются на согласованность с остальными частями, и если что-то не так - усовершенствования не добавляются в очередную версию, но работа над ними продолжается.

https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

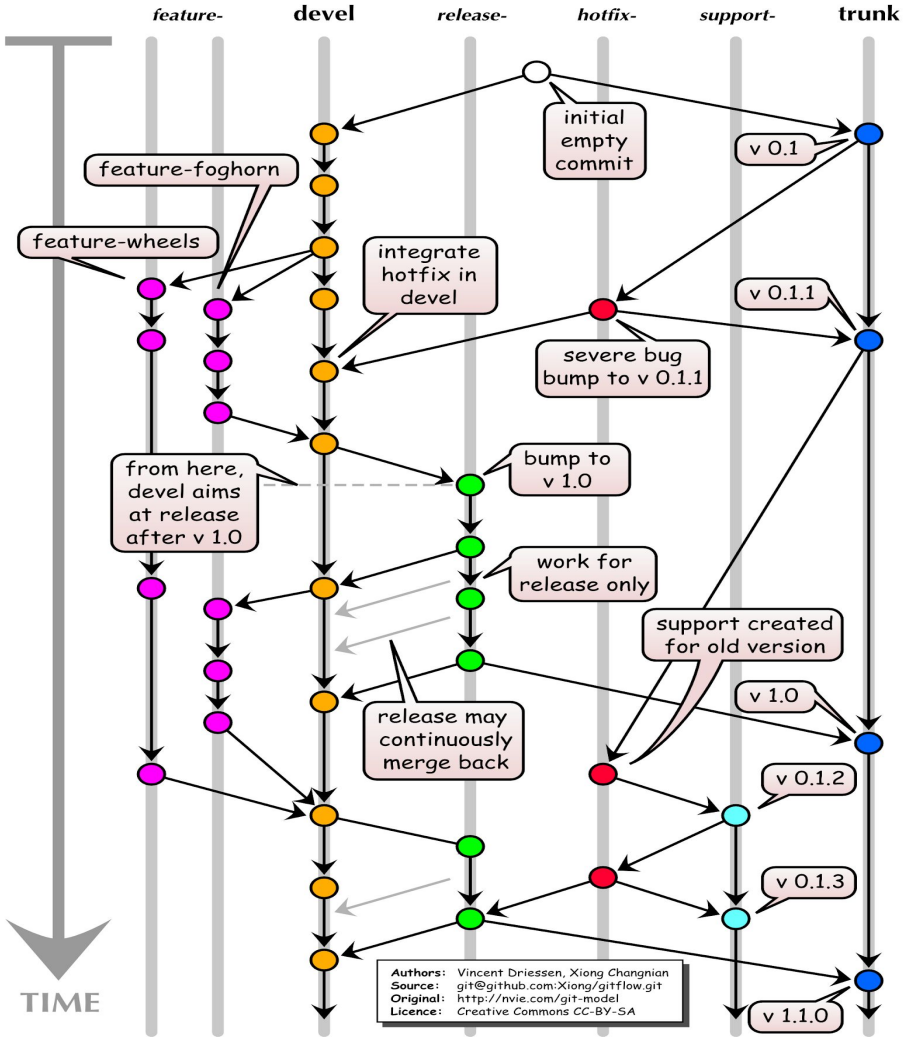
Дерево версий

иерархическая структура, отражающая очередность возникновения

- релизов (releases),
- успешных промежуточных версий (versions, alpha, beta),
- альтернативных версий (developer versions),
- версий, в которых фиксируются ошибки (bugfixes)

https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Дерево версий (пример)



https://imgur.com/gallery/EGOzs

Ветка (branch)

Делать мелкие исправления можно и в “стволе” дерева версий, однако:

- много параллельных мелких изменений требуют очень частого обновления локального репозитория разработчика, чтобы быть в курсе всех обновлений
- отменить ошибочное решение разработчика означает - отменить много других правильных решений, просто потому что они были сделаны после ошибочного решения. Поэтому...

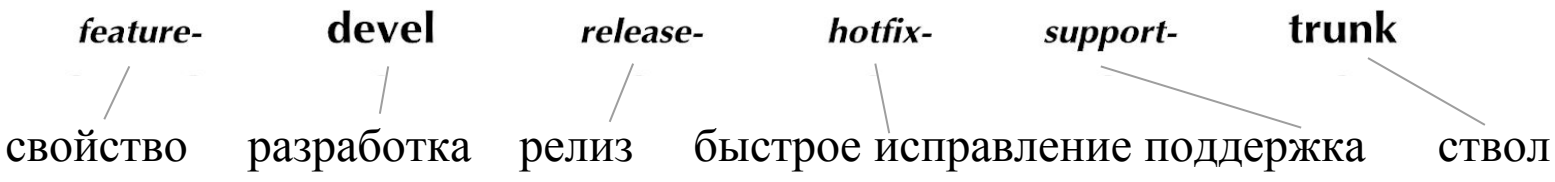
https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Ветка (branch)

Удобно отдельные задачи усовершенствования выносить в отдельные **ветки**, которые потом (после тестирования совместимости) можно будет “применить” к “стволу” дерева версий

Часто говорят “слить версии” (merge branches to master branch)

Примеры типичных именований веток:



https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Работа с ветками (bugfix)

1) Создать локально ветку для решения конкретной задачи
2) переключиться на неё с основной ветки
3) внести изменения в код
4) протестировать эти изменения на согласованность с основной веткой
5) слить ветку с основной
6) Удалить ветку

https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

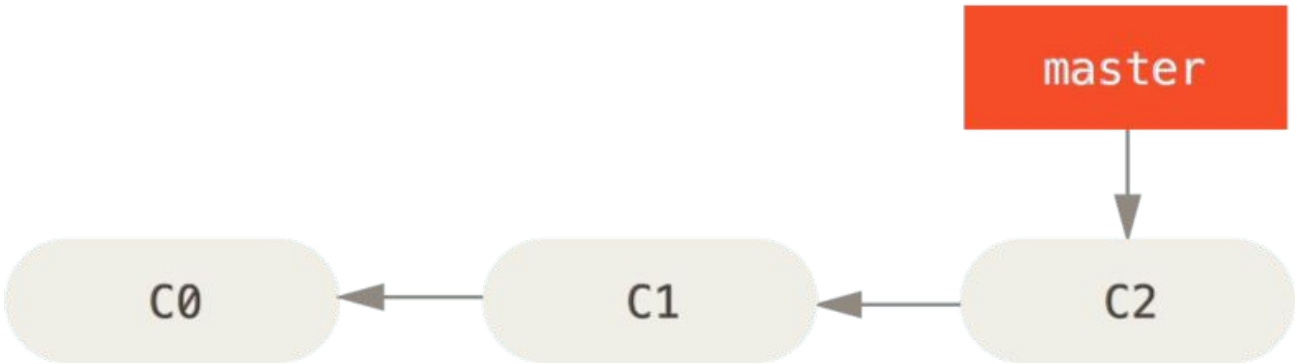
Работа с ветками (несколько разработчиков)

1) Разработчик создает для себя ветку для решения своих задач
2) переключается на неё с основной ветки
3) вносит изменения в код
4) тестирует эти изменения на согласованность с основной веткой
5) сливает ветку с основной в конце рабочего дня
6) продолжает работать в своей ветке

https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Работа с ветками в git (1)

Исходная ситуация: есть версии C0, C1, C2. Последняя версия - главная (master) на данный момент



https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

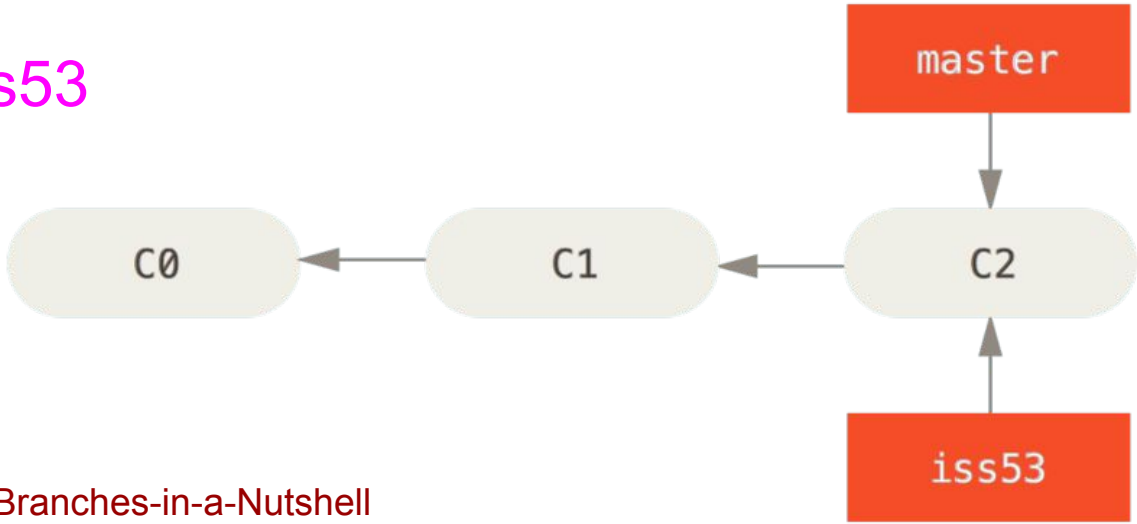
Работа с ветками в git (2)

Для исправления ошибки iss56 создается новая ветка, и разработчик переключается на неё

```
$ git branch iss53
$ git checkout iss53
```

или

```
$ git checkout -b iss53
```

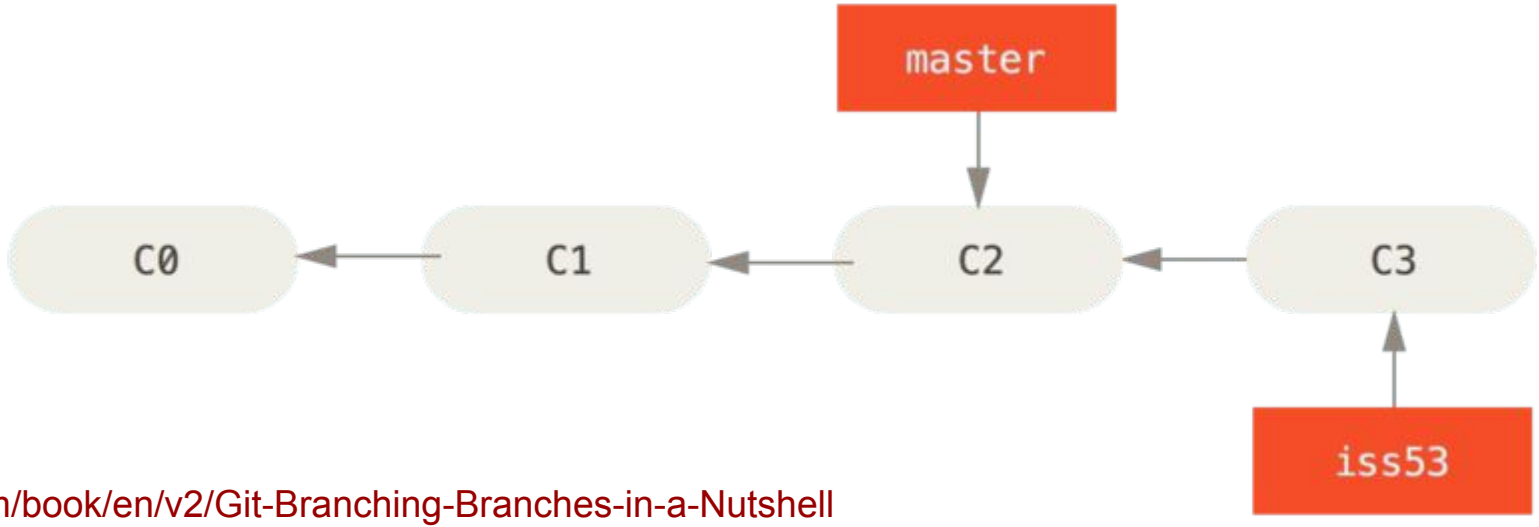


https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Работа с ветками в git (3)

Разработчик вносит изменения в код, теперь его локальная версия репозитория ушла вперед по сравнению с основной версией

$ git commit -m “Fix issue53”



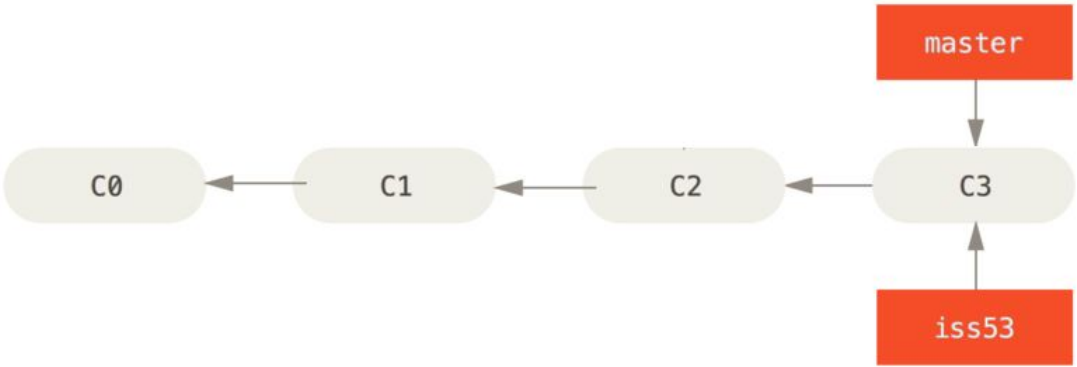
https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Работа с ветками в git (4)

Теперь, чтобы переключиться на другую задачу, нужно:
(Вариант А- внести изменения в основную ветку master): переключаемся на master и выполняем merge

```
$ git checkout master
```

```
$ git merge iss53
```

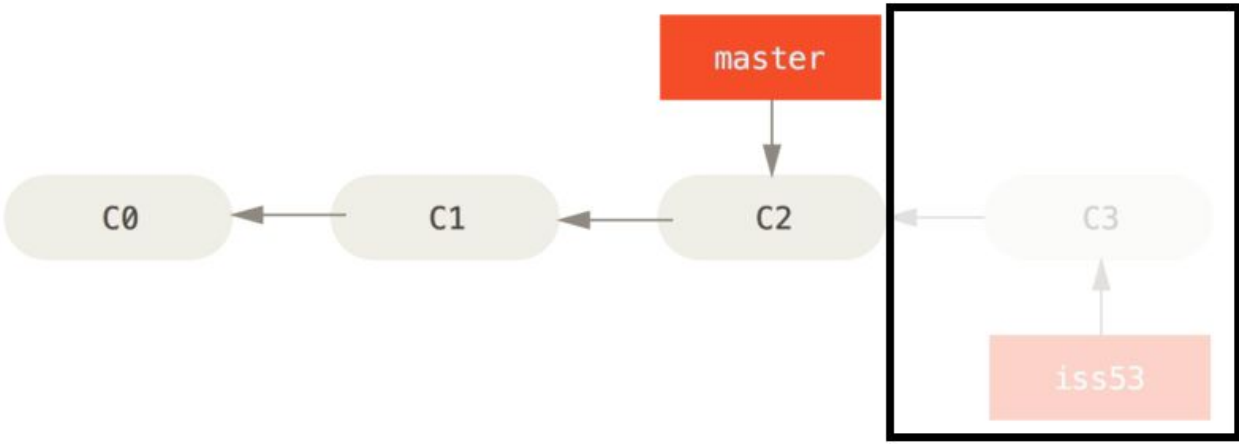


https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Работа с ветками в git (5)

Теперь, чтобы переключиться на другую задачу, нужно:
(Вариант Б - спрятать=stash изменения) : на своей ветке iss53 выполняем stash

$ git stash

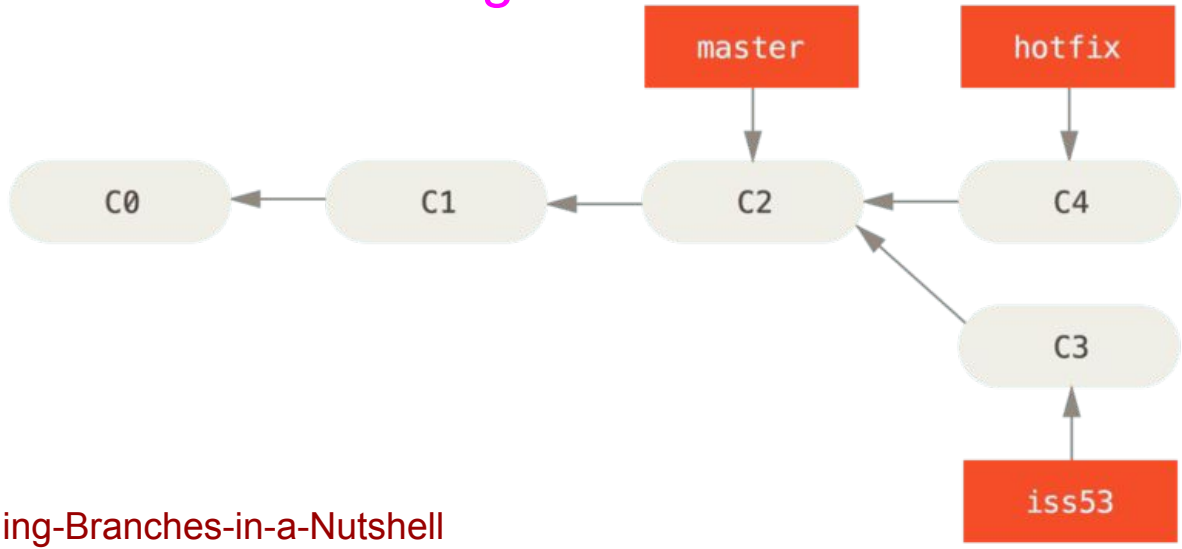


https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Работа с ветками в git (6)

Теперь предположим, нам нужно срочно исправить ошибку (bugfix):

```
$ git checkout master
$ git checkout -b hotfix
$ git commit -a -m 'fix a bug with ...'
```

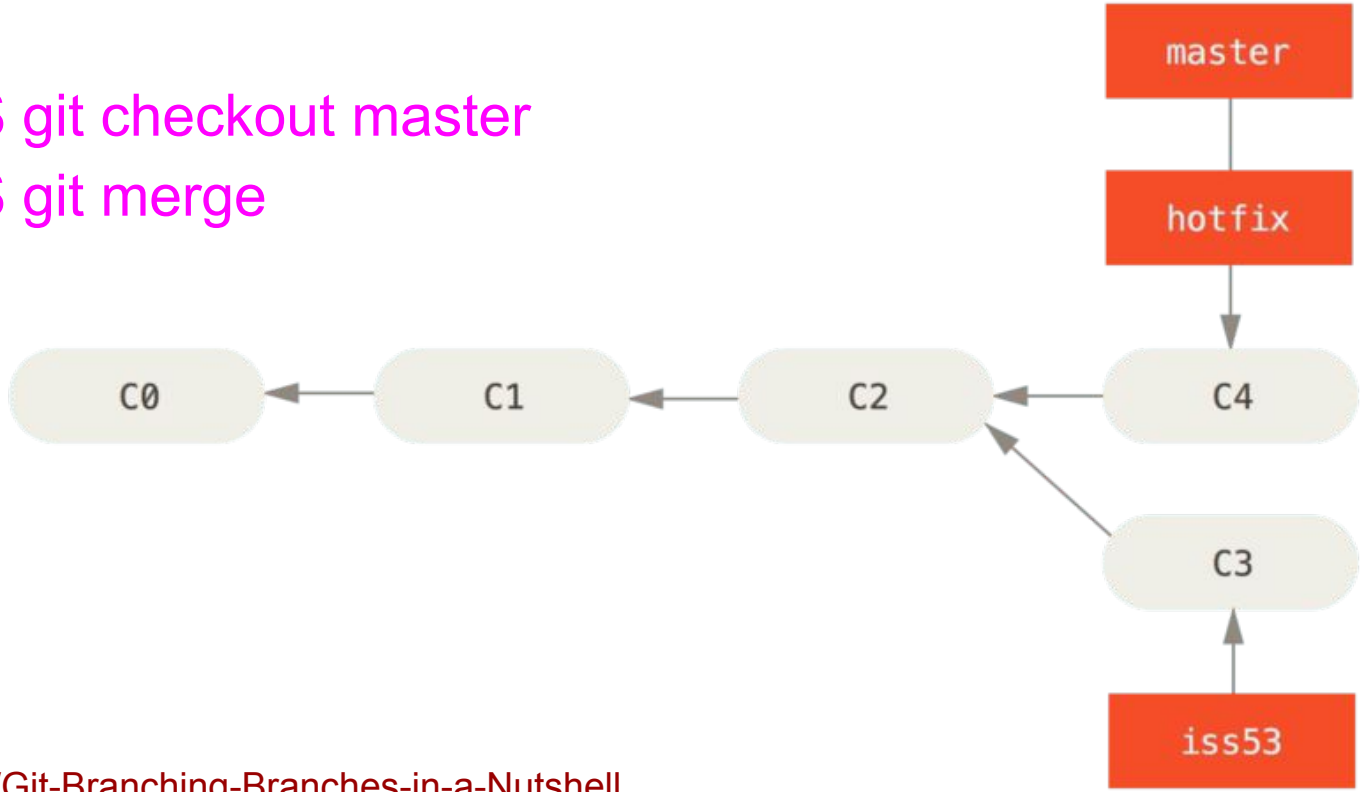


https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Работа с ветками в git (7)

Поскольку ошибка была важная, вносим изменения сразу в master

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

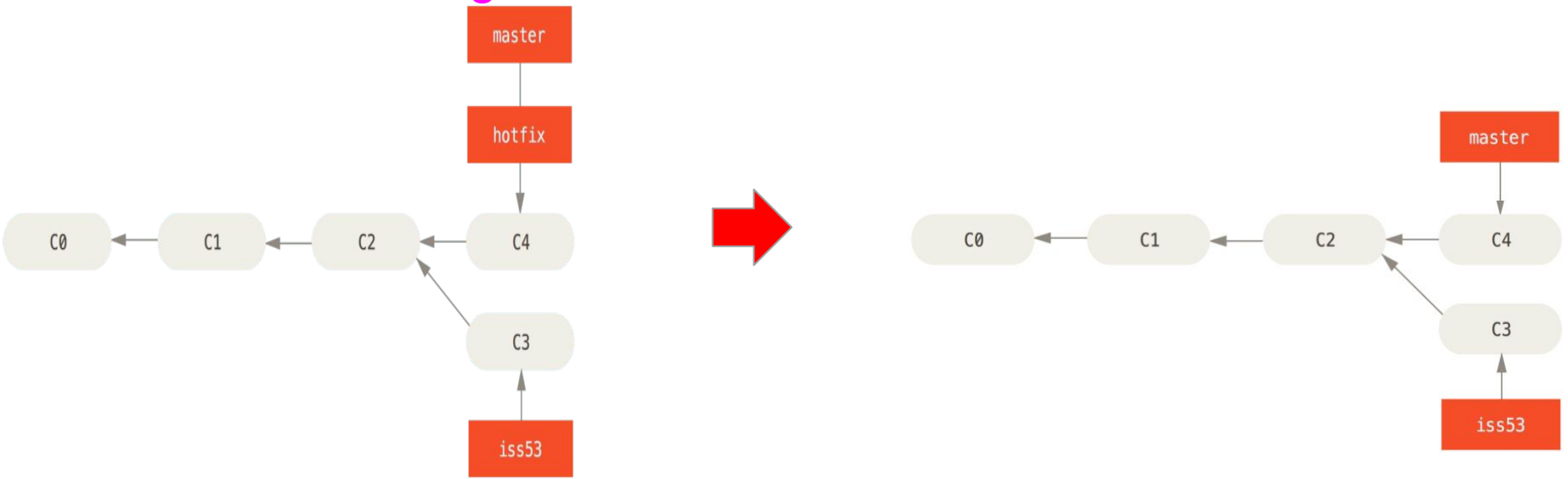


https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Работа с ветками в git (8)

Поскольку ошибка была важная, вносим изменения сразу в master и удаляем ветку hotfix

```
$ git checkout master
$ git merge
$ git branch -d hotfix
```



https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Работа с ветками в git (9)

если в работе был принят вариант Б (спрятать изменения), то возвращаемся в ветку iss53, применяем спрятанные изменения

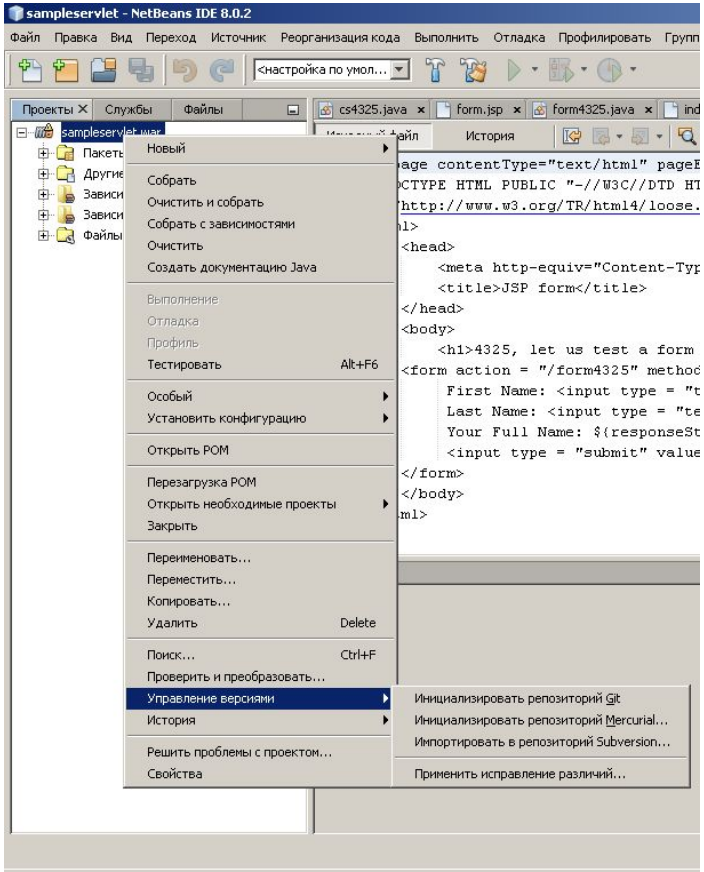
```
$ git checkout iss53
$ git commit -m “done”
$ git checkout master
$ git merge iss53
```

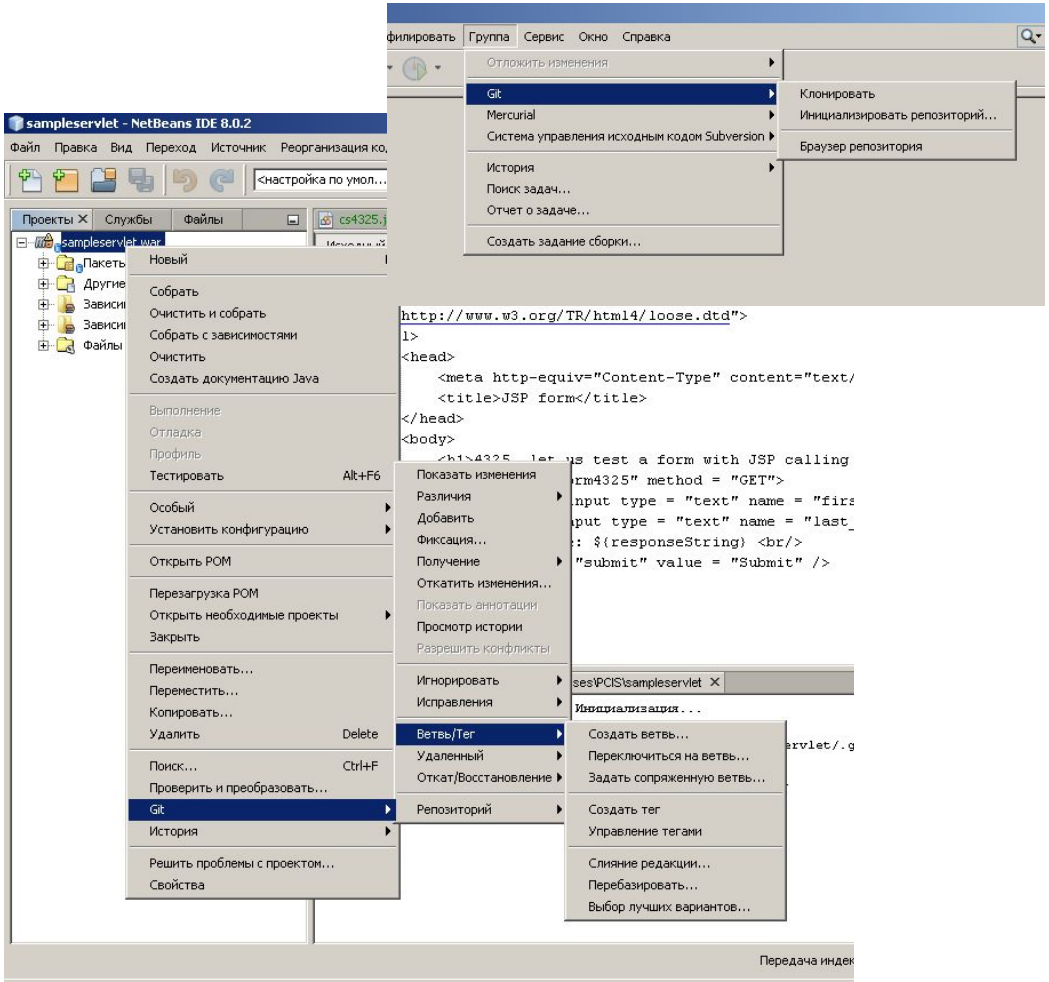
```
$ git stash apply
```

https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Интеграция с IDE

IDE Netbeans





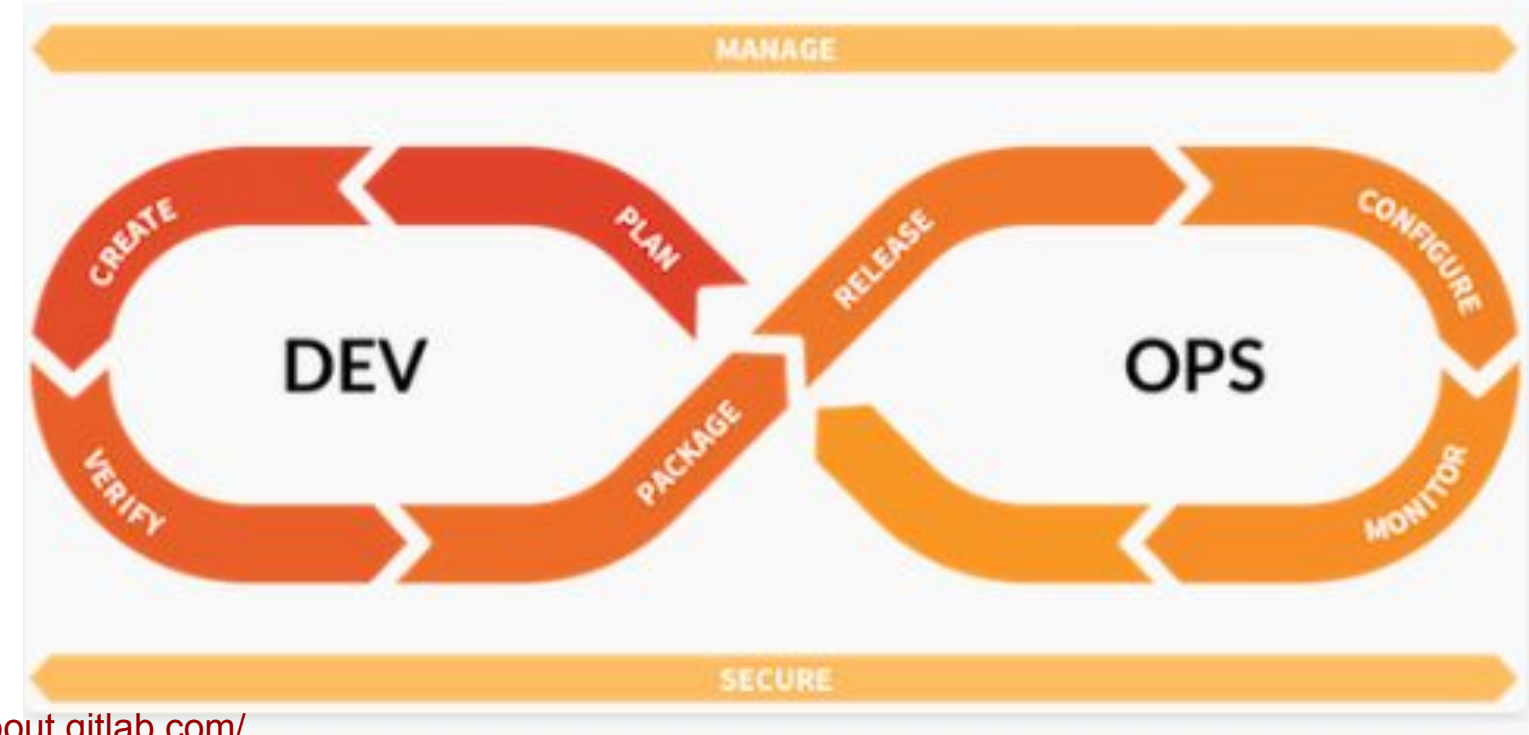
Gitlab - платформа полного цикла разработки для групп разработчиков (DevOps платформа)

DevOps

Development Operations — это набор методик автоматизации процессов между командами разработчиков и ИТ-специалистов, чтобы они могли быстрее и надежнее собирать, тестировать и выпускать релизы программного обеспечения.

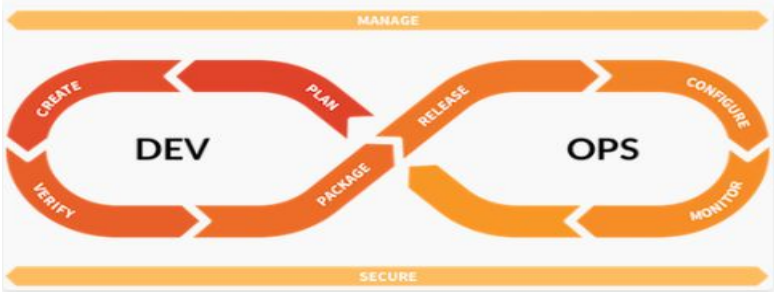
GitLab

(первое!) единое приложение для полного цикла разработки (DevOps cycle)



https://about.gitlab.com/

GitLab



1. Manage - управление циклом разработки
2. Plan - планирование разработки (ведение списка всех задач разработки - backlog)
3. Create - создание кода
4. Verify - проверка (верификация, анализ кода)
5. Package - создание пакетов для быстрого разворачивания приложения (Docker образы)
6. Release - выпуск релизов
7. Configure - конфигурация приложения и сред разработки
8. Monitor - мониторинг метрик разработки (и разработчиков)
9. Secure - тестирование безопасности