**Git**

**Система управления версиями (от англ. Version Control System, VCS)**

— программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

**Контроль** **версий** — это отслеживание изменений программного кода и управление ими.

В перечень задач, выполняемых системой контроля версий файлов, входят:

* **Сохранение и защита исходного кода.** Информация сваливается на удаленный сервер и в репозитории остаются даже файлы, удаленные с компьютера разработчика.
* **Командная работа,** возможность привлекать группу программистов, не покупая отдельно специальные инструменты для командной работы. Каждый решает свою задачу на персональном компьютере, обновляя файлы, когда это нужно.
* **Отмена внесенных изменений**. Всегда есть возможность вернуться к контрольной точке, провести ревью исходного кода и текущего, а затем обновить основную ветку.
* **Распределенная работа** над проектом. То есть, программисты могут создавать видоизмененный плагин, пока основная его версия спокойно функционирует на сайте.

**Разновидности архитектур**

**Локальная система контроля версий**

Локальная система контроля версий представляет из себя простейшую базу данных, хранящую записи обо всех изменениях в файлах. Локальная система контроля версий хорошо решает поставленную перед ней задачу, однако ее проблемой является основное свойство — локальность. Она совершенно не предназначена для коллективного использования.

**Преимущества:**

а) не требуется помощь от сторонних серверов;

б) история изменений хранится локально, без глобальной сети;

в) возможность индивидуальной работы с проектом.

**Недостатки:**

а) возможность потери данных при неисправности компьютера;

б) исключено коллективное использование, так как данные доступны только самому пользователю.

**Распределенная система контроля версий**

Данная архитектура имеет важную особенность, состоящую в том, что клиенты хранят у себя полную копию всех версий проекта, а не какую-то малую часть. При выходе из строя сервера, работа не остановится, а как только сервер восстановится на него можно будет загрузить все версии с компьютеров сотрудников. При потере копий файла проекта их легко можно восстановить с компьютера любого разработчика.

К таким системам относятся **Git**, которая на данный момент является самой быстрой распределенной системой, использующей компактное хранилище ревизий, **Mercurial**, **Bazaar** и другие.

Основными **преимуществами** распределенной VCS являются:

а) работа над проектами не зависит от работы сервера;

б) возможность работы с несколькими удаленными хранилищами;

в) гибкость общей системы и повышение надежности.

Но, если нет хорошей организации проекта, отсутствие одного центрального хранилища может стать**недостатком** данной системы.

**Централизованная система контроля версий**

Чтобы обеспечить возможность коллективной работы с проектом, была разработана централизованная система контроля версий.  Суть централизованной VCS заключается в том, что клиенты обращаются к единственному центральному серверу, который хранит все версии файлов, и получают копии файлов из него.

Примерами таких систем являются **Subversion** и **Perforce**.

**Преимущества:**

а) наличие работы в команде с другими разработчиками;

б) человеку, которые ответственен за данный проект, легче следить за работой подчиненных;

в) по сравнению с локальной администрировать централизованную систему намного легче.

**Недостатки:**

а) выход из строя или поломка сервера может остановить работу над проектом или привести к потере всех файлов;

б) при работе необходимо наличие высокоскоростной бесперебойной работы интернета.

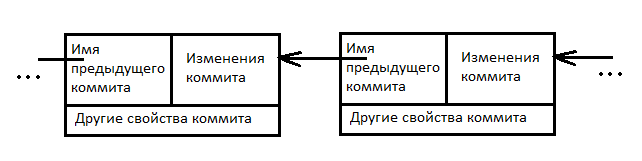
**Создание репозитория**

Все файлы проекта, находящиеся под контролем, вместе с историей их изменения и другой служебной информацией образуют ***репозиторий*** проекта.

**Изменения и их фиксация**

После создания репозитория возникает вопрос: что делать, если файл изменить?

Здесь возникает понятие ***коммит*** -  это основной объект в любой системе управления версиями. В нем содержится описание тех изменений, которые вносит пользователь в код приложения. В Git коммит состоит из нескольких так называемых объектов. Для простоты понимания можно считать, что коммиты это односвязный список, состоящий из объектов в которых содержаться измененные файлы, и ссылка на предыдущий коммит.



**Работа с GitHub**

GitHub неразрывно связан с системой контроля версий Git, которую разработчики устанавливают на персональный компьютер.

В чем разница между Git и GitHub?

С помощью Git программисты и разработчики ориентируются в коде и отслеживают изменения. Git помогает вернуть файлы в исходное состояние и видеть изменения, внесённые в определённый период. Разработчик выполняет разные команды, а все изменения синхронизируются с центральным репозиторием.

***Git*** — это система контроля версий, а ***GitHub*** — онлайн-сервис, по сути социальная сеть.

Одна из основных ***целей GitHub***— быть единым местом для проектов с исходным кодом. Предполагается, что пользователь делится чем-то полезным, а другие люди смогут участвовать в разработке.

Ещё один вариант — использовать GitHub как хранилище проектов для портфолио: легко дать на них ссылку.

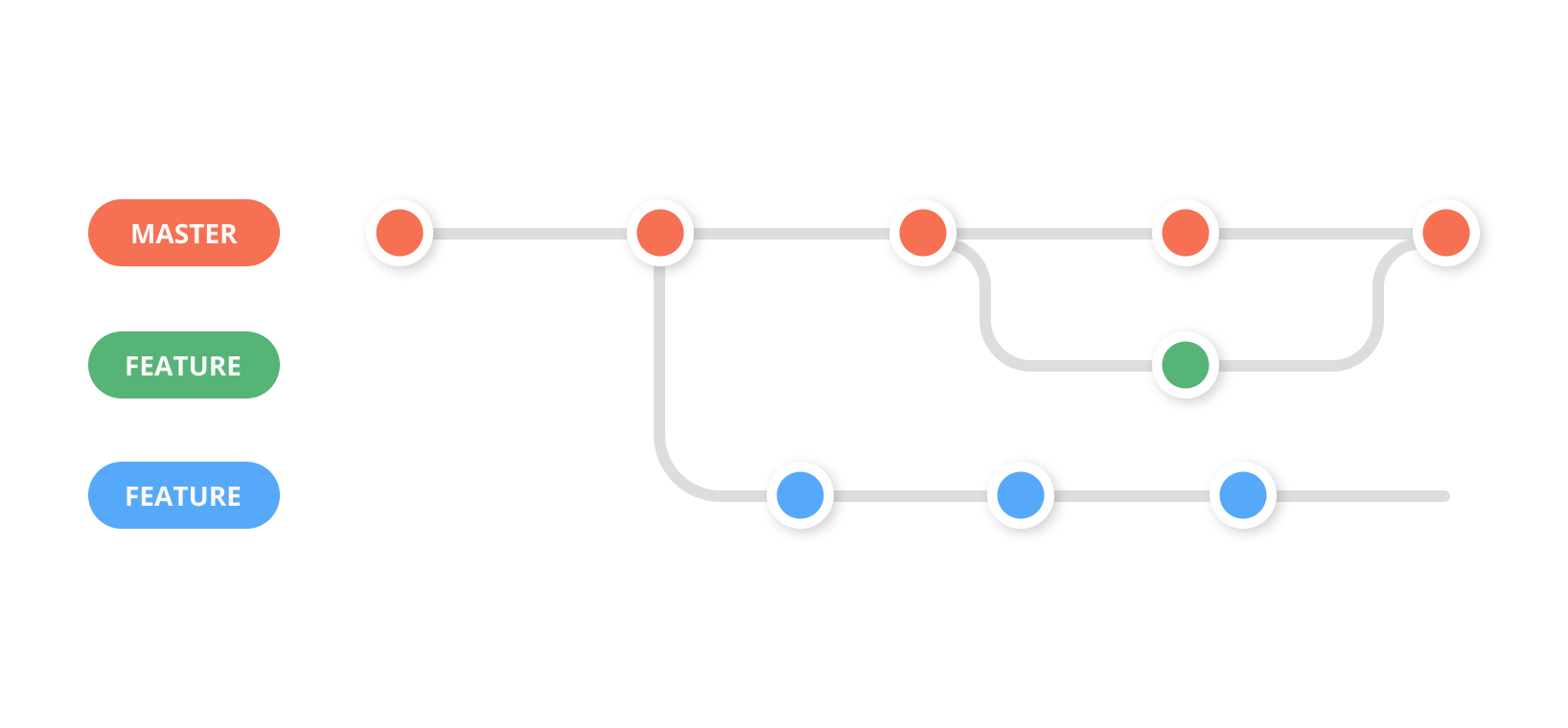
**Работа с ветками**

***Ветка в Git***это подвижный указатель на один из коммитов. Обычно ветка указывает на последний коммит в цепочке коммитов. Ветка берет свое начало от какого-то одного коммита.

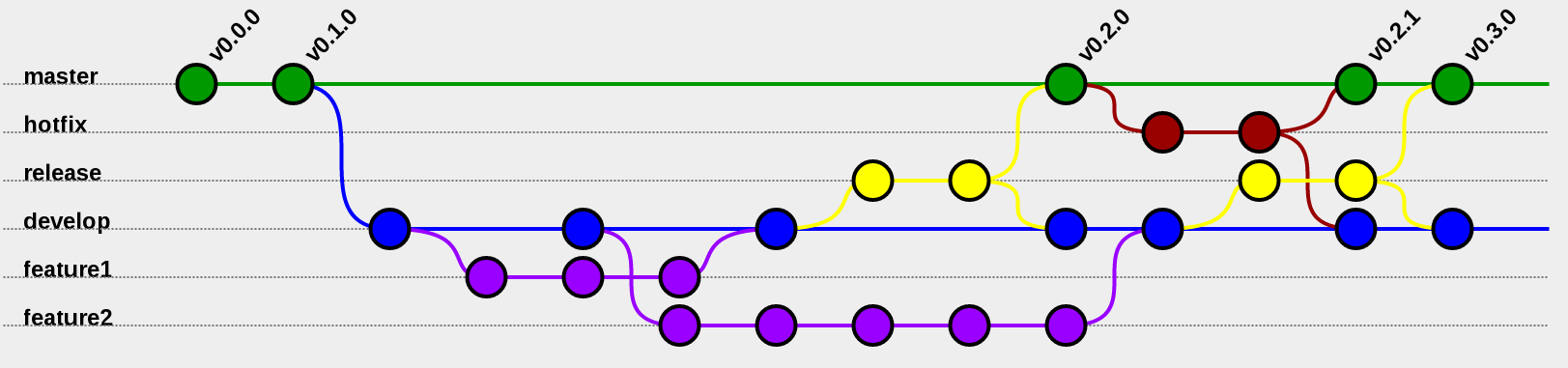
Во время создания проекта под управлением системы контроля версий создается ветка по умолчанию – master.

Ветка по сути — это ответвление от текущей версии проекта. Все изменения, которые будет вносить разработчик, будут сохраняться в ветке и когда функция будет полностью реализована и проверена, разработчик сможет слить все свои изменения с веткой master.

Наглядно это можно представить с помощью следующего рисунка:



На самом деле в большом проекте может быть огромное количество веток и они будут ветвиться и сливаться на протяжении всей работы:



**Работа в удаленном репозитории**

После клонирования репозитория можно работать с данными - это коммиты и ветки. Для работы с удаленным репозиторием применяются две команды: pull и push.

***Push (пуш)*** - это отправка данных на сервер, в удаленный репозиторий, на github.

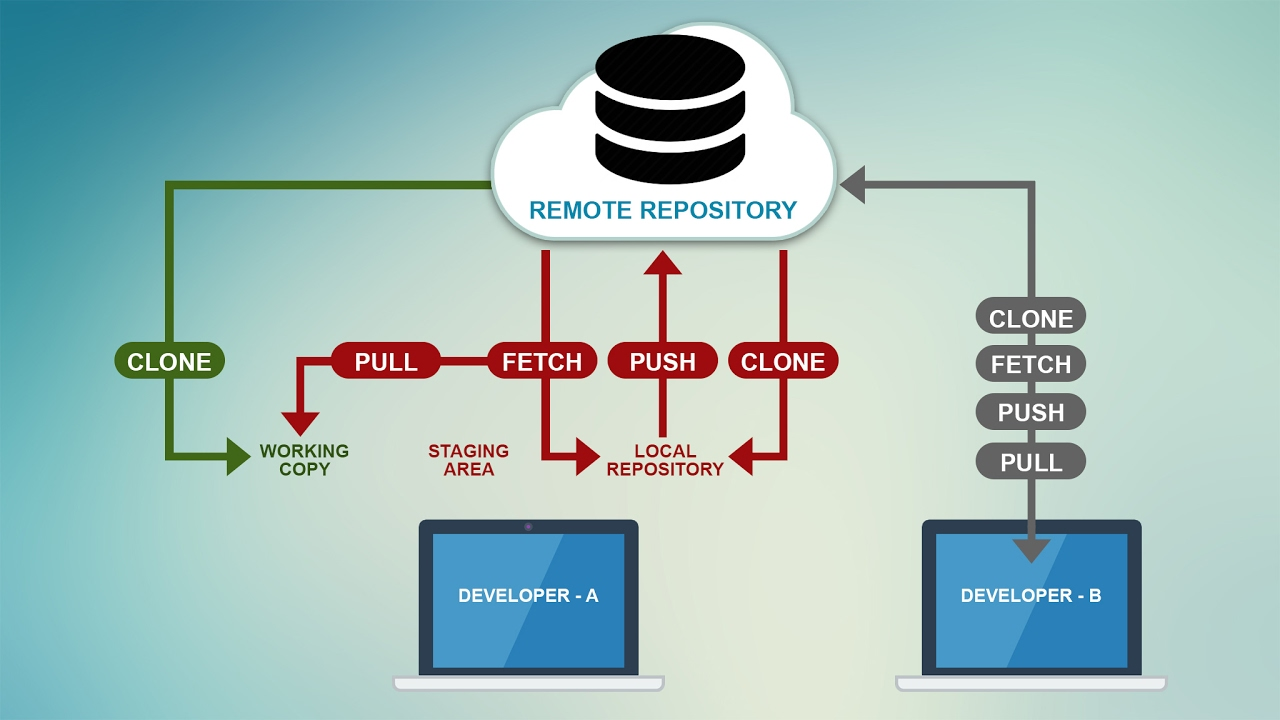
Важность команды заключается:

* делиться своим кодом с коллегами по команде
* иметь резервную копию на случай потери данных на своей машине

***Pull (пулл)*** - это скачивание данных с сервера.

Возникает вопрос: в чем же отличие пулл от клонирования? При написании пулл скачиваются не все коммиты, а только новые. Команда нужна чтобы получать изменения от других участников, если вы работаете в команде, или от себя самого, если работаете на разных машинах.

Эта команда является связкой команд git fetch и git merge. Так, отдельная команда git pull получает изменения и обновляет рабочую копию, которая станет соответствовать удаленному репозиторию.



Рассмотрим возможности работы с репозиториями, которые дает GitHub:

* Если мы хотим добавить чей-то репозиторий в собственный аккаунт, то используем функцию **fork** этого репозитория.

Со своим репозиторием мы можем делать все что угодно, а оригинал задет не будет.

* При работе над коллективным проектом можно послать **pull-request** (предложение о изменении кода), сделанные вами, другому пользователю.

Автор либо примет ваши изменения в оригинал, либо отклонит.

**Терминология**

**Дистрибутив** программы – это совокупность компонентов, необходимых для ее установки (инсталляции).

**Github Desktop** — программы под Windows 7+ и OS X, которая дублирует функциональность сайта github.com, но при этом работает локально на компьютере разработчика.

**Git Bash** — это приложение для сред Microsoft Windows, эмулирующее работу командной строки Git.

**Репозиторий** - специальное хранилище файлов и папок проекта, изменения в которых отслеживаются. Каталог, в котором хранится файловая система проекта. Для каждого проекта создаётся отдельный репозиторий. Существуют локальные и удалённые репозитории. В первом осуществляется работа над проектом на компьютере, а второй выступает в роли хранилища.

**Индекс в Git** — это специальная промежуточная область, в которой хранятся изменения файлов на пути от рабочей директории до репозитория. При выполнении коммита в него попадают только те изменения, которые были добавлены в индекс.

Один **коммит** — это пакет изменений, хранящий информацию с добавленными, отредактированными или удалёнными файлами кода. Операция позволяет зафиксировать текущее состояние проекта.

**Ветка**– это последовательность коммитов. очерняя версия основного репозитория. Она входит в его состав, но не влияет на работу. После того, как разработчики закончат работу над новой функцией или исправят все баги, можно совместить дочерний и родительский репозитории.

Описанные выше понятия встречаются очень часто, но также есть и определения, которые нужны не меньше:

**Форк**- копия репозитория, которую можно использовать для изменения исходного кода без отправки изменений в основной репозиторий.

**Пул и пуш** - первая операция позволяет выкачивать содержимое репозитория на компьютер, а вторая отправляет измененные файлы на сервер.

**Мастер** - основная ветка репозитория, в которой хранится ядро проекта. В неё добавляют изменения только после тщательного тестирования.

**Кодревью** - процесс проверки кода на соответствие техническому заданию или требованиям внутри команды. Когда один разработчик хочет добавить свой код в ядро, остальные члены команды проверяют его и если проблем нет, происходит обновление главной ветки.

**Пулреквест (Pull Request)** — запрос на слияние форка репозитория с основным репозиторием. Пулреквест может быть принят или отклонён вами, как владельцем репозитория.

**Checkout** - это команда, которая используется для переключения между ветвями.

**Merge** - это команда, которая используется для объединения изменений из одной ветки в другую.

**Fetch** - это команда, которая используется для извлечения изменений из удаленного репозитория в локальный репозиторий.

## **Голые (bare) хранилища**

Голое (bare) хранилище называются так потому, что у него нет рабочего каталога. Оно содержит только файлы, которые обычно скрыты в подкаталоге .git. Другими словами, голое хранилище содержит историю изменений, но не содержит снимка какой-либо определенной версии.

Голое хранилище играет роль, похожую на роль основного сервера в централизованной системе управления версиями: это дом вашего проекта. Разработчики клонируют из него проект и закачивают в него свежие официальные изменения. Как правило, оно располагается на сервере, который не делает почти ничего кроме раздачи данных. Разработка идет в клонах, поэтому домашнее хранилище может обойтись и без рабочего каталога.

Многие команды Git не работают в голых хранилищах, если переменная среды GIT\_DIR не содержит путь до хранилища и не указан параметр --bare.

## **Push или pull?**

Зачем вводится команда push, вместо использования уже знакомой pull? Прежде всего, pull не работает в голых хранилищах, вместо нее нужно использовать команду fetch, которая будет рассмотрена позже. Но даже если держать на центральном сервере нормальное хранилище, использование команды pull в нем будет затруднительным. Нужно будет сначала войти на сервер интерактивно и сообщить команде pull адрес машины, с которой мы хотим забрать изменения. Этому могут мешать сетевые брандмауэры (firewall), но в первую очередь: что если у нас нет интерактивного доступа к серверу?

Тем не менее, не рекомендутся push-ить в хранилище помимо этого случая — из-за путаницы, которая может возникнуть, если у целевого хранилища есть рабочий каталог.

Короче говоря, пока изучаете Git, push-те только в голые хранилища. В остальных случаях pull-те.