1. Что такое Git?

Git – распределенная система управления версиями. То есть основная задача Git'a – создание различных версий приложения. Но это происходит не автоматически, а необходимо создавать версии в определенный момент времени. Например: вы завершили разработки какой-либо функции, вы можете создать “новую версию приложения” и при необходимости вернутся

на “старые версии приложения”.

Что значит распределенная?

Это значит, что на каждом компьютере находится полная версия репозитория. То есть вы можете с сервиса хостинга Git репозиториев (например GitHub) скачать репозиторий локально на ваш компьютер и продолжить работу в этом репозитории локально даже без интернета. На компьютере будут хранится все объекты Git, необходимые, например, для переключения между разными версиями проекта.

1. Как работать с Git?

Мы разобрались что такое Git, теперь определимся как с ним работать. Скачать git нужно с сайта git-scm.com, а работать нужно с помощью консоли (Для windows по умолчанию PowerShell, для MacOS по умолчанию “terminal”)

1. Создание нового Git репозитория.

Git init вводится в консоль, находясь в той папке, в который вы хотите создать новый Git репозиторий. (Для того, чтобы через консоль дойти до нужной папки, необходимо с помощью команды cd перейти в папку, в которой нужно создать репозиторий).

-После инициализации создается скрытая папка .git в той папке, где вы создали новый Git репозиторий. По умолчанию в большинстве операционных системах скрытые папки не отображаются

4) Области Git.

Первая область – Рабочая директория. Это именно та область, которая содержит видимые папки и файлы. В идекс добавляются файлы, до того как вы хотите их сохранить. То есть в индексе подготавливайте файлы для сохранения. Далее вы можете сохранить папки, находящиеся в области индекс, в область репозиторий. Именно область репозитория находится в скрытой папке .git в objects (именно тут сохраняются все версии проекта)

5) Типы объектов Git

Blob – эти объекты создаются для файлов, для каждого файла создается отдельный..Blob.  
 Tree такие объекты представляют собой папки. Они содержат ссылки на те файлы, которые находятся в папке.

Commit создается тогда, когда вы сохраняете текущую версию вашего проекта

Каждый объект содержит свой уникальный хэш. Хеш – это специальная метка, позволяющая отличать одни объекты от других. (Хеш коммита состоит из 40 символов)

6) Что такое коммит? Как его создать?

Коммит – указатель на дерево (определенную папку). Рассмотрим пример.

Пусть у нас будут следующие объекты: синим цветам у нас обозначены файцы, а красным – папки. Сейчас рассмотрим как Git сохраняет текущую версию проекта в Git репозиторий. В данном примере у нас есть проект и он находится в определенной корневой папке (это папка вашего проекта, которая находится под контролем Git'a. Именно в этой папке находится скрытая подпапка .git).

При создании коммита Git создает все объекты типа Blob и Tree, а также еще объект объект – коммит, который ссылается на корневое дерево. То есть в примере показано 6 объектов: 2 типа Tree и 3 типа Blob и commit.

Корневой коммит – самый первый коммит проекта.

Каждый последующий коммит ссылается на предыдущий, кроме корневого.

Указатель называется в Git'e HEAD. Он ссылается на определенный коммит. В рабочей директории вы видите именно тот коммит, на который указывает HEAD. Перемещая указатель на другой коммит вы можете перемещаться в истории версий. Тем самый в рабочей директории вы будете видеть тот коммит, на который указывает HEAD.

Но на самом деле этот указатель обычно является указателем на определенную ветку, а уже ветка указывает на определенный коммит. При инициализации Git создается главная ветка Master.

При создании нового коммита Git автоматически передвигает указатель ветки на последний (новый) коммит.

Давайте продолжим наш пример. Перепишем наш пример и внесем изменения: удалим файл в корневом дереве и добавим сюда пустую папку. Далее создадим новый коммит и посмотрим что случится. Мы видим, что указатель Head и Ветки сместились. Также видим, что новый коммит ссылается на предыдущий, тоесть каждый новый коммит ссылается на предыдущий (еще его называют родительским). Еще видим одну вещь: Git для нетронутой папки не создавал новый объект, он просто переиспользует уже созданные раннее объекты.

7) Как создать коммит?

Git status отображает текущее состояние репозитория Git. Эта команда, например, покажет какие файлы были изменены/добавлены/удалены. Еще данная команда показывает какие файлы находятся в индексе.

Git add – добавление файлов в индекс. После инициализации репозитория вы удалять/изменять/добавлять файлы в рабочей директории. Когда вы уже хотите сохранить версию проекта вы должны добавить файлы в индекс.

Git commit – создание коммита с записью изменений в репозиторий. Все то, что было подготовлено в индексе, будет сохранено в репозиторий Git в виде отдельных объектов. После этой команды в терминале появится текстовый файл, который надо будет отредактировать и указать сообщение коммита там (в сообщение пишут что было добавлено в новую версию).

8) Как перемещаться между разными коммитами?

Git log – просмотр истории изменений. Показывает все коммиты, которые были сделаны до последнего. Также показывает хэш каждого коммита, можно взять часть хэша и перейти в определенную версию проекта. Для этого используется команда git checkout.

9) Ветки в Git

Как я говорил ранее, есть одна ветка, которую Git создает автоматически – ветка Master. Но кроме этой ветки можно создавать другие ветки. Рассмотрим пример

У нас на ветке Master есть четыре коммита, кроме этого есть еще две ветки: ветка1, содержащая 3 коммита, и ветка2, содержащая 3 коммита. Далее рассказать про историю созданий веток.

Если мы переключимся на ветку1 и введем git log, то мы увидим только родительские. То же самое с веткой2.

Ветки нужны для “разных направлениях” вашего проекта и можно разрабатывать разный функционал проекта в разных ветках. На практике этим занимаются разные разработчики, то есть каждый разработчик создает свою ветку и в ней разрабатывает новый функционал, независимо от других разработчиков. Когда функционал готов можно сделать слияние веток (об этом чуть позже)

git branch <branch name> – создание новой ветки (называть ветки нужно так, чтобы название отображало какой функционал разрабатывается)

git checkout <branch name> – переход в новую ветку

git branch – отображает список всех веток

10) Слияние веток

Продолжим наш пример. Предположим, что мы закончили делать функционал в ветке1 и хотим сделать слияние с веткой master. Для этого создается новый коммит. Как мы видим, он будет ссылаться на два родительских коммита. С помощью такого коммита сохраняется история обоих веток. В новом коммите с помощью команды git log будут видны коммиты обоих веток. Такой коммит называют merge commit. Когда такой коммит создается, HEAD и ветка Master перемещается на этот коммит. Только что мы сделали слияние ветки1 в ветку Master, то есть ветка Master принимает изменения, которые были сделаны в ветке1. После слияние остается указатель ветка1, который указывает на все тот же коммит, поэтому эту ветку можно удалить.

Иногда при попытки слияния веток могут возникнуть ошибки. Например, когда один и тот же файл был изменен в разных ветках. Git выдаст ошибку и ее надо решить. Вы, как разработчик должны решить какой файл будет использован в merge commit.

git merge <feature branch name> – слияние другой ветки в текущую