1. 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
2. Основное отличие Git от любой другой СКВ (включая Subversion и её собратьев) — это подход к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы (CVS, Subversion, Perforce, Bazaar и т.д.) представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени (обычно это называют контролем версий, основанным на различиях). Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён
3. Модифицированные (modified)

Индексированные (staged)

Зафиксированные (commited)

1. Индексация определяет какие изменения будут занесены в коммит. Выполняется командой **git add**
2. После индексации изменения фиксируются снимком. Команда **git commit**
3. git config

usage: git config [<options>]

Config file location

--global use global config file

--system use system config file

--local use repository config file

--worktree use per-worktree config file

-f, --file <file> use given config file

--blob <blob-id> read config from given blob object

Action

--get get value: name [value-regex]

--get-all get all values: key [value-regex]

--get-regexp get values for regexp: name-regex [value-regex]

--get-urlmatch get value specific for the URL: section[.var] URL

--replace-all replace all matching variables: name value [value\_regex]

--add add a new variable: name value

--unset remove a variable: name [value-regex]

--unset-all remove all matches: name [value-regex]

--rename-section rename section: old-name new-name

--remove-section remove a section: name

-l, --list list all

-e, --edit open an editor

--get-color find the color configured: slot [default]

--get-colorbool find the color setting: slot [stdout-is-tty]

Type

-t, --type <> value is given this type

--bool value is "true" or "false"

--int value is decimal number

--bool-or-int value is --bool or --int

--path value is a path (file or directory name)

--expiry-date value is an expiry date

Other

-z, --null terminate values with NUL byte

--name-only show variable names only

--includes respect include directives on lookup

--show-origin show origin of config (file, standard input, blob, command line)

--show-scope show scope of config (worktree, local, global, system, command)

--default <value> with --get, use default value when missing entry

1. Командой **git init**
2. Командой **git add <имя файла>** или **git add .** для всех файлов в репозитории
3. Командой **git commit –m “описание коммита”**
4. **git status**
5. Файл не имеет названия и имеет разрешение .gitignore

Используется для игнорирования объектов системой git

В нем могут быть записаны пути к папкам ***folder\***

Пути к конкретным файлам *ignorefile.txt*

Игнорирование файлов определённого разрешения *\*.txt*

Строчка комментария начинается с *#*

1. **git diff** помогает сравнить между собой разные коммиты, ветки.

Для вывода изменений в файлах по сравнению с последним коммитом, используется **git diff** без параметров

1. **git commit** сохраняет коммит всех индексируемых файлов

**git commit** без ключей – откроется окно, в котором пишется комментарий к коммиту

**git commit –a** совершит коммит, автоматически индексируя изменения в файлах проекта. Новые файлы при этом индексироваться не будут! Удаление же файлов будет учтено.

**git commit -m «commit comment»** комментируем коммит прямо из командной строки вместо текстового редактора.

**git commit FILENAME** внесет в индекс и создаст коммит на основе изменений единственного файла.

1. **git log** выводит историю коммитов
2. Команда git commit --amend — это удобный способ изменить последний коммит. Она позволяет объединить проиндексированные изменения с предыдущим коммитом без создания нового коммита. Ее также можно использовать для простого редактирования комментария к предыдущему коммиту без изменения состояния кода в нем. Но такое изменение приводит не только к редактированию последнего коммита, но и к его полной замене. Это означает, что измененный коммит станет новой сущностью с отдельной ссылкой.
3. Для того, чтобы просмотреть список настроенных удалённых репозиториев, вы можете запустить команду **git remote**. Она выведет названия доступных удалённых репозиториев. Если вы клонировали репозиторий, то увидите как минимум **origin** — имя по умолчанию, которое Git даёт серверу, с которого производилось клонирование

Вы можете также указать ключ **-v**, чтобы просмотреть адреса для чтения и записи, привязанные к репозиторию:

$ git remote -v

origin https://github.com/schacon/ticgit (fetch)

origin https://github.com/schacon/ticgit (push)

1. Для получения данных из удалённых проектов, следует выполнить

$ git fetch [remote-name]

Данная команда связывается с указанным удалённым проектом и забирает все те данные проекта, которых у вас ещё нет. После того как вы выполнили команду, у вас должны появиться ссылки на все ветки из этого удалённого проекта, которые вы можете просмотреть или слить в любой момент.

Когда вы клонируете репозиторий, команда **clone** автоматически добавляет этот удалённый репозиторий под именем “origin”. Таким образом, **git fetch origin** извлекает все наработки, отправленные на этот сервер после того, как вы его склонировали (или получили изменения с помощью **fetch**). Важно отметить, что команда **git fetch** забирает данные в ваш локальный репозиторий, но не сливает их с какими-либо вашими наработками и не модифицирует то, над чем вы работаете в данный момент. Вам необходимо вручную слить эти данные с вашими, когда вы будете готовы.

Если у вас есть ветка, настроенная на отслеживание удалённой ветки, то вы можете использовать команду **git pull** чтобы автоматически получить изменения из удалённой ветки и слить их со своей текущей. Этот способ может для вас оказаться более простым или более удобным. К тому же, по умолчанию команда **git clone** автоматически настраивает вашу локальную ветку **master** на отслеживание удалённой ветки **master** на сервере, с которого вы клонировали репозиторий. Название веток может быть другим и зависит от ветки по умолчанию на сервере. Выполнение **git pull**, как правило, извлекает (**fetch**) данные с сервера, с которого вы изначально склонировали, и автоматически пытается слить (**merge**) их с кодом, над которым вы в данный момент работаете.

1. Когда вы хотите поделиться своими наработками, вам необходимо отправить их в удалённый репозиторий. Команда для этого действия простая: **git push <remote-name> <branch-name>**. Чтобы отправить вашу ветку **master** на сервер **origin** (повторимся, что клонирование обычно настраивает оба этих имени автоматически), вы можете выполнить следующую команду для отправки ваших коммитов:

$ git push origin master

Эта команда срабатывает только в случае, если вы клонировали с сервера, на котором у вас есть права на запись, и если никто другой с тех пор не выполнял команду **push**. Если вы и кто-то ещё одновременно клонируете, затем он выполняет команду **push**, а после него выполнить команду **push** попытаетесь вы, то ваш **push** точно будет отклонён. Вам придётся сначала получить изменения и объединить их с вашими и только после этого вам будет позволено выполнить **push**

1. $ git remote show

4. Команда **git merge** используется для слияния одной или нескольких веток в текущую. Затем она устанавливает указатель текущей ветки на результирующий коммит.
5. Запрос на изменение кода в удаленном репозитории владельцу этого репозитория
6. **git log**
7. **git checkout -b style-** новая ветка с именем **style**
8. Команда **git checkout** используется для переключения веток и выгрузки их содержимого в рабочую директорию.

11. GitLab — веб-инструмент жизненного цикла DevOps с открытым исходным кодом, представляющий систему управления репозиториями кода для Git с собственной вики, системой отслеживания ошибок, CI/CD пайплайном и другими функциями.