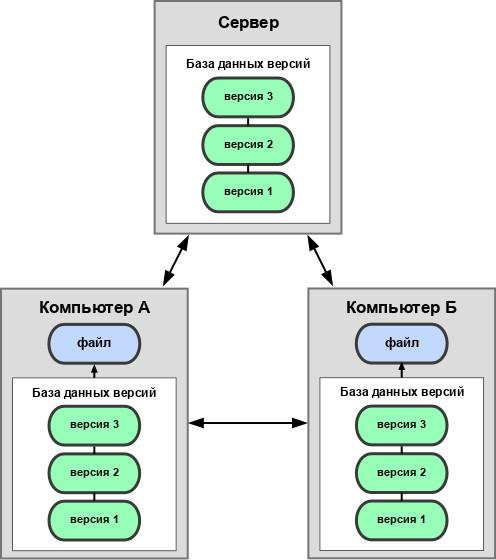
**Система управления версиями** или **Система контроля версий** (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

**

*Рисунок 1. Схема распределённой системы контроля версий*

**Глоссарий Git.**

**Git** — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux. На сегодняшний день его поддерживает Джунио Хамано.

**Репозиторий Git** — каталог файловой системы, в котором находятся: файлы конфигурации, файлы журналов операций, выполняемых над репозиторием, индекс расположения файлов и хранилище, содержащее сами контролируемые файлы.

**Локальный репозиторий** — репозиторий, расположенный на локальном компьютере разработчика в каталоге. Именно в нём происходит разработка и фиксация изменений, которые отправляются на удалённый репозиторий.

**Удалённый репозиторий** **(Remote repository)** — репозиторий, находящийся на удалённом сервере. Это общий репозиторий, в который приходят все изменения и из которого забираются все обновления.

**Коммит (Commit)** — фиксация изменений или запись изменений в репозиторий. Коммит происходит на локальной машине.

**Пул (Pull)** — получение последних изменений с удалённого сервера репозитория.

**Пуш (Push)** — отправка всех неотправленных коммитов на удалённый сервер репозитория.

**Ветка (Branch)** — это параллельная версия репозитория. Она включена в этот репозиторий, но не влияет на главную версию, тем самым позволяя свободно работать в параллельной. После внесения нужных изменений в параллельной ветке можно объединить их с главной (мастер) версией.

**Мастер (Master)** — главная или основная ветка репозитория.

**Мёрдж (Merge)** — слияние изменений из какой-либо ветки репозитория с любой веткой этого же репозитория. Чаще всего слияние изменений из ветки репозитория с основной веткой репозитория.

**Клонирование (Clone)** — скачивание репозитория с удалённого сервера на локальный компьютер в определённый каталог для дальнейшей работы с этим каталогом как с репозиторием.

**Форк (Fork)** — копия репозитория. Его также можно рассматривать как внешнюю ветку для текущего репозитория. Копия открытого удалённого репозитория может быть сделана любым пользователем, после чего он может прислать изменения в ваш репозиторий через пулреквест.

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

Пользоваться мы будем Git Bash ([скачать](https://git-for-windows.github.io/)), но существует множество графических оболочек (посмотреть и скачать можно [здесь](https://git-scm.com/downloads/guis)). Все инструкции для Git Bash, представленные ниже, действительны для ОS Windows и вводятся в консоль или в оболочку Git Bash (как в примерах).

**Первоначальная настройка Git.** После установки Git в системе необходимо настроить среду для работы под себя.

Команда ***git config***

Эта утилита входит в состав Git и позволяет просматривать и устанавливать различные параметры для работы. Параметры отправляются в конфигурационные файлы.

Опция *--global* записывает настройки в общий конфигурационный файл, который используется для всей системы Git. Указать настройки с такой опцией достаточно один раз. Для их изменения нужно прописать команду заново, с другими значениями параметров.

* *Имя пользователя и адрес электронной почты* необходимо внести в первую очередь. Эта информация включена в коммиты для определения их автора и не может быть вычеркнута оттуда.

Пример:

Brier@Pc3 MINGW64 /d

$ git config --global user.name "flamebrier"

$ git config --global user.email "flamebrier@gmail.com"

Проверка:

Brier@Pc3 MINGW64 /d

$ git config user.name

flamebrier

* *Параметры замены перевода строк.* В разных операционных системах приняты разные символы для перевода строк, в Windows – это два символа *\r\n* (или *CR* и *LF*), в \*nix – просто *\n* (*LF*), а в Mac – *\r* (*CR*). Настройка *core.autocrlf* предназначена для того, чтобы в главном репозитории все переводы строк текстовых файлах были одинаковы (приведены к виду \n). Настройка *core.safecrlf* не допускает коммита изменений если в файлах по какой-либо причине остались неверные символы перевода строк.

Пример:

Brier@Pc3 MINGW64 /d

$ git config --global core.autocrlf true

$ git config --global core.safecrlf true

Если перед коммитом замена перевода строки сработала недостаточно точно, появится ошибка “fatal: CRLF would be replaced by LF in Filename.cs” (обычно автозамена редко даёт сбой). Можно вручную с помощью поиска и замены исправить окончание строки в указанном файле (через редактор кода Notepad++ или специальные сторонние утилиты, например, dos2unix).

Второй путь – смириться с тем, что не все файлы будут кроссплатформенными и установить настройку:

Brier@Pc3 MINGW64 /d

$ git config core.safecrlf warn

После этого вместо ошибки будет выводиться простое предупреждение и коммит изменений будет успешно завершён.

**Локальный репозиторий.** На конкретном компьютере создаётся локальный репозиторий, в котором проводится разработка и фиксируются изменения.

Команда ***git init***

Создает в директории пустой репозиторий в виде скрытой директории .*git*, где и будет в дальнейшем храниться вся информация об истории коммитов, тегах — о ходе разработки проекта. Команду можно вызвать как из пустой директории, так и из директории с существующими файлами – файлы при этом не пострадают.

* *Создание локального репозитория.* В консоли переходим в директорию, в которой будем управлять версиями файлов. В примере это *testgit*. Если директория пуста, помещаем туда нужные файлы. В примере это файлы *Logo.png* и *output.txt.* Вызываем команду ***git init***

Пример:

Brier@Pc3 MINGW64 /d/Enthusiasm/testgit

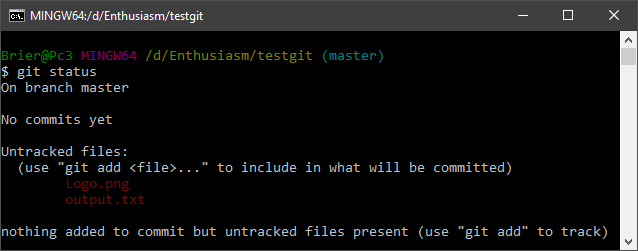
$ git init

Initialized empty Git repository in D:/Enthusiasm/testgit/.git/

Команда ***git status***

Эту команду можно считать самой часто используемой, наряду с командами коммита и индексации. Она выводит информацию обо всех изменениях, внесенных в дерево директорий проекта по сравнению с последним коммитом рабочей ветки; отдельно выводятся внесенные в индекс и неиндексированные файлы.

* *Просмотр начального состояния локального репозитория (рисунок 4).* Вызываем команду *git status*. Строка *On branch* сообщает название ветки, в которой мы находимся (в примере – *Master*). Затем идёт список сообщений выполненных коммитов (пока ни одного). В блоке *Untracked files* перечислены не отслеживаемые файлы. Кроме этого, присутствуют некоторые комментарии от системы Git.



*Рисунок 2. Пример. Результат git status*

Команда ***git add***

Индексирует файлы, то есть делает их доступными для коммита.

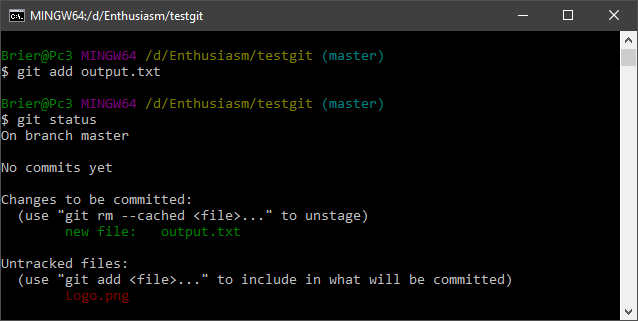
*git add* ***.***

Индексирует **все** не отслеживаемые файлы директории – точка обязательна.

*git add <file>*

Индексирует файл или директорию с названием <file>.

*Проиндексируем текстовый файл (рисунок 5).* Вызываем команду *git add <file>* (в примере файл output.txt). В результате команды *git status* появился блок *Changes to be committed*, где указан проиндексированный файл.

*Рисунок 3. Git add*

* *Внесём изменения в файл.* Напишем строку в текстовый файл и снова вызовем *git status.* Появился блок *Changes not staged to commit*, а текстовый файл помечен как изменённый (*modified*). Чтобы проиндексировать файл, и оставшийся тоже, вызываем *git add* ***.*** (рисунок 6). Теперь все файлы проиндексированы.

Команда ***git commit***

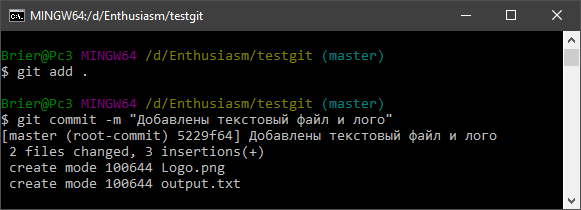
Эта команда фиксирует изменения, то есть создаёт версию всех проиндексированных файлов. Файлы индексируются с помощью команды *git add*, индексированные файлы перечислены в блоке *Changes to be committed* в выводе *git status* и выделены зелёным. Все прочие файлы не войдут в коммит!

Без параметров команда git commit запускает текстовый редактор, что не очень удобно. Более быстрый способ – использовать опции.

Опция *-m “Message text”*

Добавляет комментарий к коммиту. Комментарий нужен для опознания одного коммита среди других и обычно содержит описание того, что было изменено в файлах этого коммита по сравнению с файлами предыдущего.

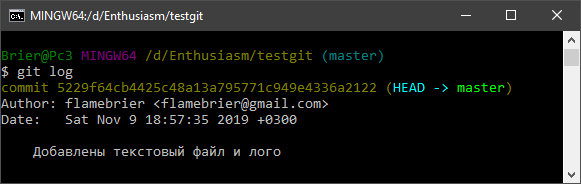
* *Фиксируем изменения файлов (делаем коммит) (рисунок 6).*



*Рисунок 4. Git commit*

Команда ***git log***

Позволяет просмотреть историю коммитов. По умолчанию, без опций, *git log* выводит список коммитов, созданных в данном репозитории в обратном хронологическом порядке. То есть самые последние коммиты показываются первыми. При использовании опций или при большом выводе *log* выводится в специальном изолированном режиме. Чтобы выйти из этого режима необходимо нажать клавишу q.



*Рисунок 5. Git log*

Команда ***git reset --hard***

Заменяет текущие файлы из репозитория на их версии из последнего коммита. То есть, если файл был испорчен, то все изменения можно откатить до состояния, зафиксированного коммитом. Опция *--hard* (перед hard две чёрточки) заставляет Git привести рабочий каталог к состоянию коммита.

*git reset --hard SHA1\_HASH*

Каждому коммиту соответствует SHA1\_HASH – хеш-код, по которому коммит можно однозначно идентифицировать. Хеши можно увидеть в выводе команды git log. Если ввести код хеша после опции --hard, то текущее состояние рабочего каталога откатится не до последнего коммита, а до коммита с указанным хешем. Все коммиты позже указанного по хешу автоматически удалятся.

Команда ***git checkout***

Переключает указатель (HEAD) гита на какую-либо существующую версию.

Если после команды указать SHA1\_HASH (хеш копируем из вывода git log, он пишется после слова commit), то состояние каталога перейдет к состоянию соответствующего хешу коммита. При этом более поздние коммиты не удаляются (как в предыдущем случае), а остаются параллельно.

Если вместо хеша указать название ветки, то каталог будет переведён в состояние указанной ветки.

Чтобы отменить команду *git checkout* нужно выполнить *git switch -*

Пример:

Brier@Pc3 MINGW64 /d/Enthusiasm/testgit (master)

$ git checkout 5229f64cb4425c48a13a795771c949e4336a2122

Note: switching to '5229f64cb4425c48a13a795771c949e4336a2122'.

…

HEAD is now at 5229f64 Добавлены текстовый файл и лого

Brier@Pc3 MINGW64 /d/Enthusiasm/testgit ((5229f64...))

$ git switch -

Switched to branch 'master'

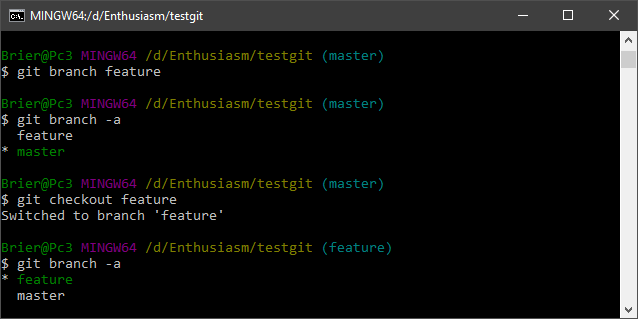
Аналогия: компьютерная игра. *git reset --hard* загружает ранее сохраненную игру и удаляет все версии, сохраненные после только что загруженной. Но, вполне вероятно, что вы захотите вернуться к старому состоянию и очень не хотелось бы, чтобы оно пропало. В этом нам поможет команда: *git checkout SHA1\_HASH.* То есть, *git checkout* загружает старую игру, но, если вы продолжаете играть, состояние игры будет отличаться от более новых сохранений, которые вы сделали в первый раз. Любая игра, которую вы теперь сохраняете, попадает в отдельную ветвь, представляющую альтернативную реальность, в которую вы вошли.

**Работа с ветками (branches).** Ветки позволяют создать параллельную версию репозитория и работать в ней без вреда для главной. Например, если в готовый проект нужно встроить новый функционал, то разрабатывать его можно в параллельной версии проекта, а затем, когда разработка будет завершена, внести в основную (слить с главной веткой, сделать мёрдж (merge)).

Команда ***git branch***

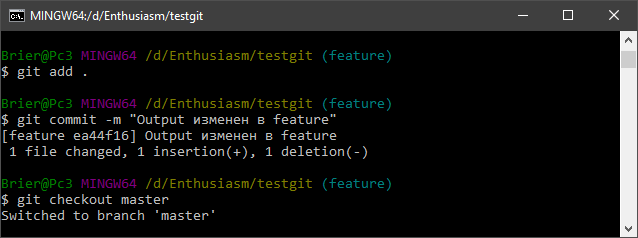
Управляет ветками. Если после команды прописать название, то будет создана новая ветка. Просмотреть список веток можно с помощью опции *-a* Тогда текущая ветка будет выделена зелёным и отмечена звёздочкой. Когда Git переключен на ветку все коммиты будут записываться в неё.

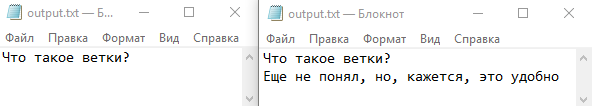
* *Создание и переключение на новую ветку (рисунок 8).* Ветку можно создать как с помощью команды git branch <branch\_name>, так и с помощью опции *git checkout* ***-b*** *<branch\_name>*, где *<branch\_name>* – название создаваемой ветки. Переключение на ветку происходит с командой *git checkout <branch\_name>*. При создании ветки через *checkout* Git сам переключится на неё.

**

*Рисунок 6. Создание ветки и переключение на неё*

* *Коммит в ветке (рисунок 9).* Изменим текстовый файл из примера, закоммитим изменения в ветку feature. Затем переключимся на основную ветку master. Снова просмотрим файл. При просмотре с разных веток файл содержит различный текст.

*Рисунок 7. Коммит в ветке*

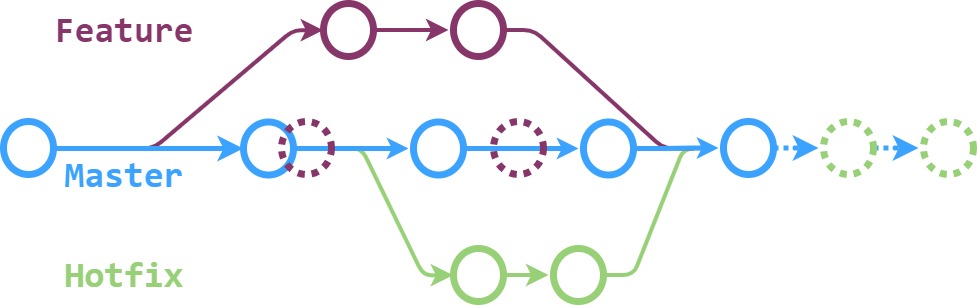


*Рисунок 8. Просмотр файла из ветки master (слева) и из ветки feature (справа)*

Команда ***git merge*** *[branch]*

С помощью этой команды текущая ветка сливается с другой. После *git merge* пишется название ветки, которую вы хотите слить с текущей.

Для слияния по умолчанию используется режим *Fast-forward* (перемотка). Это значит, что Git будет стараться встроить изменения из другой ветки так, будто бы они происходили в текущей (рисунок 11).



*Рисунок 9. Ветви с Fast forward (Feature) и без него (Hotfix)*

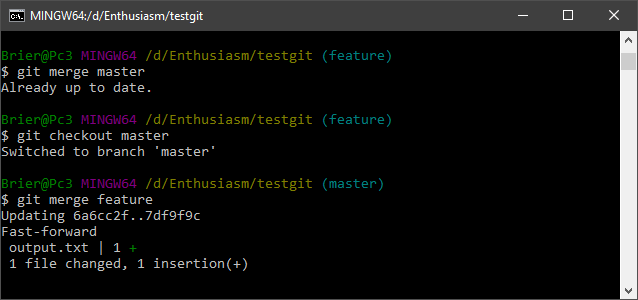
Если вы изменили одну и ту же часть файла по-разному в двух ветках, которые собираетесь слить, Git не сможет сделать слияние чисто и выдаёт ошибку. В *git status* можно просмотреть конфликтующие файлы.

Всё, что имеет отношение к конфликту слияния и что не было разрешено, отмечено как *unmerged*. Git добавляет стандартные маркеры к файлам, которые имеют конфликт, так что вы можете открыть их вручную и разрешить эти конфликты. Ваш файл содержит секцию, с специальными знаками (*<<<<<<< HEAD*, *=======*) указывающими на конфликт.

После того как вы разобрались с каждой из таких секций в каждом из конфликтных файлов, выполните *git add* для каждого конфликтного файла. Индексирование будет означать для Git'а, что все конфликты в файле теперь разрешены.

Если вы хотите использовать графические инструменты для разрешения конфликтов, можете выполнить команду *git mergetool*, которая запустит соответствующий графический инструмент и покажет конфликтные ситуации

* *Слияние двух веток (рисунок 12).* Перед тем, как сливать *master* с какой-либо веткой, рекомендуется сначала слить эту ветку с *master*, разрешить все конфликты, и потом уже слить *master* с этой веткой.



*Рисунок 10. Слияние веток master и feature*

**Удалённый репозиторий GitHub.** Нужно зарегистрироваться или войти в свой аккаунт на [GitHub](https://github.com/) и создать репозиторий БЕЗ ФАЙЛА README.MD. На GitHub есть исчерпывающая [инструкция](https://guides.github.com/activities/hello-world/) по созданию первого репозитория и работы с ним через веб-приложение. В данной лабораторной работе рассматривается работа с созданным удалённым репозиторием через Git Bash.

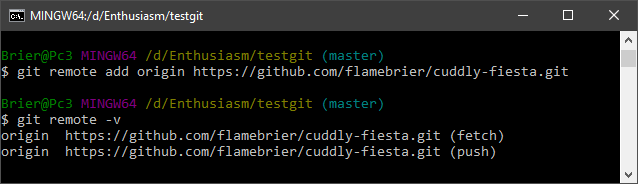
Команда ***git remote***

Команда *git remote*управляет удалёнными репозиториями. Просмотреть список добавленных удалённых репозиториев можно с помощью опции *-v*

Команда ***git remote add*** *[shortname] [url]*

Добавить удалённый репозиторий. После *add* пишется короткое имя репозитория, которое можно использовать для работы с ним. После имени пишется ссылка на добавляемый репозиторий.

* *Добавление удалённого репозитория на GitHub (рисунок 13).* На странице нужного репозитория есть зелёная кнопка *Clone or download*, которая при клике на неё предоставляет ссылку, заканчивающуюся на *.git* Эту ссылку указываем при добавлении репозитория.

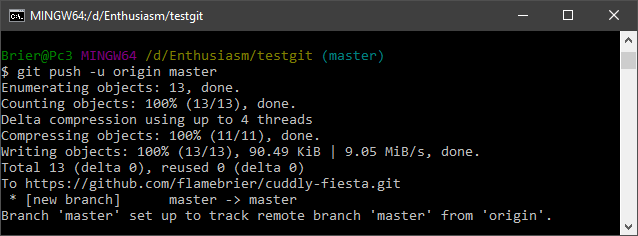


*Рисунок 11. Добавление удалённого репозитория и просмотр добавленных репозиториев*

Команда ***git push*** *-u [shortname] [branch]*

Вливание с локального репозитория на удалённый. Опция *-u* связывает ветку на удалённом репозитории с веткой на локальном, чтобы в дальнейшем отправлять изменения можно было без указания ветки. *[shortname]* – название добавленного удалённого репозитория. *[branch]* – соответствующая ветка на удалённом репозитории.

* *Залив изменений на удалённый репозиторий (рисунок 14)*

**

*Рисунок 12. Git push*

Команда ***git clone*** *[url]*

Команда загружает копию удалённого репозитория из ссылки, указанной после.

Пример:

Brier@Pc3 MINGW64 /d/Enthusiasm/testgit (master)

$ git clone https://github.com/flamebrier/cuddly-fiesta.git

Команда ***git pull***

Обновляет уже загруженную копию репозитория до последней версии.

Brier@Pc3 MINGW64 /d/Enthusiasm/testgit (master)

$ git pull origin