1. Работа с командной строкой.

3. Локальная работа с Git-репозиторием

4. Базовые операции

5. История Git-проекта

6. Ветки

7. Удаленные git репозитории

8. Удаленные ветки

9. Версионирование

10. Слияние и переносы

1. Работа с командной строкой (Git Bash).

|  |  |
| --- | --- |
| $ **pwd** | где мы находимся (домашний каталог) |
| **Перемещение по каталогам** | |
| $ **cd** D:/GEEK/git/work  $ **cd** ../file.txt  $ **cd** ./file.txt | перемещение в указанный каталог  перемещение из родительского каталога (на уровень выше)  перемещение из текущего каталога |
| **Работа с каталогами** | |
| $ **mkdir** test | создать каталог |
| $ **ls** -**l**  $ **ls** -**l** -**a** ($ **ls** -**la**) | посмотреть содержимое каталога (-a -скрытые файлы)  можно просто **ls** |
| **Работа с файлами** | |
| $ **echo** 'Hello, world!' | обычный вывод фразы в командной строке |
| $ **touch** hello.txt | создать файл в текущем каталоге |
| $ **unlink** hello.txt | удалить файл |
| $ **echo** 'Hello, world!' **>** hello.txt | записать фразу в файл **с перезаписью** содержимого файла. |
| $ **echo** 'Hello, world!' **>>** hello.txt | дописать фразу к концу файла |
| $ **cat** hello.txt | посмотреть содержимое файла |
| $ **cp** hello.txt text.txt | копировать файл (hello) и назначить ему новое имя (text) |
| $ **mv** hello.txt text.txt | переместить (с переименованием) |
| $ **rm** hello.txt  $ **rm -R** | удалить файл (не директорию)  удалить директорию и все, что внутри (сначала выйти из нее)  -R рекурсивное удаление |

3. Локальная работа с Git-репозиторием

|  |  |
| --- | --- |
| **Настройка git** | |
| $ **git --version** | версия git |
| $ **git config --global user.name** "Sergej Mikhasko"  $ **git config --global user.email** mikhassergej@yandex.com | обязательные разовые настройки на локальной машине |
| $ **git config​ --list** | просмотреть настройки |
| $ **git config --list --show-origin** | местоположение конфигурационного файла (текстовый файл с расширением .gitconfig) |
| $ **git config --global color.ui auto** | подсветка |
| $ **git config --global** core.editor "[program]" | программа для редактирования по-умолчанию |
| $ **git config --global** core.quotepath off | установка отображения Unicode |
| $ **git config --global** core.autocrlf true  $ **git config --global** core.safecrlf warn | параметры установки окончаний строк (Win) |

**Создание git проекта с консоли**

|  |  |
| --- | --- |
| $ **cd** D:/GEEK/0\_git/work | переместиться в папку, которую нужно отслеживать |
| $ **git init** | пометить папку, в которой мы находимся как git проект. Внутри нее появится файл. git с необходимой информацией |

**Создание коммита**

|  |  |
| --- | --- |
| $ **git add .** | проиндексировать все файлы в каталоге |
| $ **git add** hello.txt | проиндексировать конкретный файл |
| **$ git reset** [file] | отменить индексацию файла |
| $ **git commit** | откроется редактор по-умолчанию.  **I** – войти в режим записи (INSERT)  В нем следует ввести коммит и выйти с помощью команды **ESC** и  **:wq** (write, quit) |
| $ **git commit -m** 'Комментарий' | создать коммит (если комментария нет - откроется редактор). В коммит входят только проиндексированные файлы. |
| $ **git add .** +$ **git commit** = $ **git commit -a**  или $ **git commit -am** 'Комментарий' | объединение команд |

**Графические редакторы**

|  |  |
| --- | --- |
| $ **git config core.editor** notepad | Редактор по-умолчанию довольно сложный. Можно выбрать другой редактор по-умолчанию (notepad) |
| $ **gitk** | просмотр истории |
| $ **git gui** | графическое добавление коммитов  Save Change (индексация) -> Commit |

4. Базовые операции

|  |  |
| --- | --- |
| **Отслеживание изменений** | |
| $ **git status**  $ **git status -s** (компактнее) | Статус проекта. Так можно увидеть файлы, которые отслеживаются и модифицированы и не отслеживаемые файлы. Не отслеживаемые папки в консоли не видны (если в них ничего нет, а если есть, то папки видны, а содержимое - нет). Пустую папку тоже можно отслеживать – см. далее |
| $ **git diff** | Просмотреть изменения (только у отслеживаемых файлов). Если файлы в индексе – они не отобразятся (для них **git diff --cached**). |
| $ **git diff --cached** | Изменения, которые находятся в индексе и войдут в коммит. |
| **Удаление\откат файлов** | |
| $ **unlink** to\_delete.txt | Удаление не отслеживаемого файла (самое простое). Git не нужен. Если отслеживается тоже легко удаляется, но удаление отображается в отчете git и его можно откатить. |
| $ **git checkout --** hello.txt | Откат. Если файл отслеживается (но еще не проиндексирован), его можно откатить назад до нужного состояния (это не откат к предыдущим коммитам!). |
| $ **git reset HEAD** hello.txt | Отмена индексации. Если файл успели проиндексировать, то индексацию можно отменить, а сам файл потом откатить до нужного состояния.  HEAD означает текущее место (состояние, ветка) в репозитории |

5. История Git-проекта

[Просмотр истории](https://git-scm.com/book/ru/v2/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%9F%D1%80%D0%BE%D1%81%D0%BC%D0%BE%D1%82%D1%80-%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8-%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D0%BE%D0%B2)

[Откат к предыдущим коммитам](https://ru.stackoverflow.com/questions/431520/%D0%9A%D0%B0%D0%BA-%D0%B2%D0%B5%D1%80%D0%BD%D1%83%D1%82%D1%8C%D1%81%D1%8F-%D0%BE%D1%82%D0%BA%D0%B0%D1%82%D0%B8%D1%82%D1%8C%D1%81%D1%8F-%D0%BA-%D0%B1%D0%BE%D0%BB%D0%B5%D0%B5-%D1%80%D0%B0%D0%BD%D0%BD%D0%B5%D0%BC%D1%83-%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D1%83)

|  |  |
| --- | --- |
| $ **git ​log**  $ **git log -2**  $ **git log -2 -p**  $ **git log -2 --stat**  **$ git reflog** | история изменений (Пробел, q - выход) – **книга стр.42**  2 последних изменения  подробный вывод для 2-х последних коммитов  вывод со статистикой (кол-во удаленных и добавленных строк)  - более подробные сведения о репозитории (включая изменения коммитов). История действий. |
| **$git log --oneline**  **$git log --pretty=oneline**  **$** **git log --pretty=format:"%h - %an, %ar : %s"** | Форматированный вывод (в одну строку). Можно составлять любые свои комбинации. |
| **Изменение истории** | |
| **$ git commit --amend** | изменение последнего коммита |
| **$ git reset [хэш\_коммита]** | удаление (очистка) коммита из истории (но не из репозитория). Указывается хэш, до которого нужно откатиться.  Его можно вернуть, если войти через **git reflog,** увидеть хеш коммита, при котором произошло удалениеи выполнить команду удаления (очистки) но уже на хеш этого коммита. |
| **$ git reflog** | Просмотр истории изменений. |
| **$ git revert [хэш\_коммита]** | отмена действий коммита и создание нового. Указывается хэш удаляемого (заменяемого) коммита |
| **$ git restore [файл]** | откатить состояние фала до состояния последнего коммита |
|  |  |
| **Поиск коммита** | |
| **$ git log --since=2.week** | поиск коммита (все от 2 недель назад до текущего момента) |
| **$ git ​log​ --untilg=1.week** | поиск от начала до недельной давности |
| **$ git log --since=2019-03-01 --until=2019-03-25** | точная дата |
| **$ git log -SGit -p** | поиск коммита по строке (“Git”) |
| **$ git log --author="Vasya"** | поиск коммита по автору |
| **$ git blame readme.txt** | поиск автора изменений |
| **Игнорирование** | |
| Для этого в корне проекта создается файл **.gitignore** и внутри него прописываются шаблоны игнорируемых файлов (файл без имени с расширением **.gitignore**). Ресурс http://gitignore.io/ позволяет автоматически формировать содержимое **.gitignore**-файла. | |
| **\*.log** | игнорируем файлы с расширением log |
| **tmp​ /\*** | игнорируем файлы в папке temp |
| **Добавить пустую папку в отслеживаемые** | |
|  | для этого достаточно разместить внутри нее скрытый пустой файл с именем .**gitkeep** |

Опции для **git log –pretty=format (книга – стр. 41)**

|  |  |
| --- | --- |
| %H | Хеш коммита |
| %C | Параметр цвета текста.  Чтобы сбросить ранее указанный цвет - %Сreset;  %C - Color, reset - сброс; |
| %h | Сокращенный хеш коммита |
| %an | Имя автора |
| %ar | Относительная дата автора |
| %s | Содержание |

6. Ветки

|  |  |
| --- | --- |
| **Создание ветки** | |
| $ **git branch** | просмотр всех доступных веток |
| $ **git branch** feature | создать ветку (feature) |
| $ **git checkout** feature | переключится на ветку (feature) |
| $ **git checkout -b** ​ test | создать ветку (test) и переключиться на нее  Эта команда позволяет отпочковывать ветки от ранних коммитов (см. далее – версионирование) |
| **$ git branch new\_feature master** | отпочковать ветку new\_feature от master, не важно где мы находимся в данный момент |
| **Переименование** | |
| $ **git branch -m** feature to\_delete | переименовать ветку (feature на to\_delete) |
| **Удаление** | |
| $ **git branch -D** to\_delete | удалить ветку (to\_delete). Сами должны быть в другой ветке |
| **Слияние** | |
| $ **git merge ​** test | Слияние ветки (test) с той, в которой мы находимся |
| **Вывод на экран консоли** | |
| $ **git log --pretty=format:"%h -%d %s %cd" --graph** | Удобный вывод результата с псевдографиком |
| **Псевдонимы команд** | |
| Можно команды упаковывать в какую-то свою простую команду, введя для этого псевдоним. | |
| $ **git config --global alias.grog** 'log --graph --abbrev-commit --decorate --all  --format=format:"%C(bold blue)%h%C(reset) - %C(bold cyan)%aD%C(dim white) - %an%C(reset) %C(bold green)(%ar)%C(reset)%C(bold yellow)%d%C(reset)%n %C(white)%s%C(reset)"' | Вызов $ **git grog** |
| $ **git config --global alias.lg** "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr)%C(bold blue)<%an>%Creset' --abbrev-commit" | Вызов $ **git lg** |
| **Разрешение конфликтов** | |
| Предположим, на двух разных ветках (master, feature) имеется один файл, который мы правили по-разному. При слиянии git просигнализирует об ошибке.  Нужно открыть этот файл и привести к тому виду, который необходим. Затем проиндексировать его (или все) и добавить коммит. Слияние произведено. | |
|  |  |

7. Удаленные git репозитории

|  |  |
| --- | --- |
| <https://github.com/>    <https://github.com/MikhaskoS/DemoExt.git> (HTTPS)  [git@github.com:MikhaskoS/DemoExt.git](mailto:git@github.com:MikhaskoS/DemoExt.git) (SSH) | Git-хостинг.  Предположим, на хосте имеется репозиторий  [MikhaskoS](https://github.com/MikhaskoS)/[DemoExt](https://github.com/MikhaskoS/DemoExt) |
| **$ git clone** https://github.com/MikhaskoS/DemoExt.git  **$ git clone** git@github.com:MikhaskoS/DemoExt.git | Клонирование удалённого репозитория в текущую папку. Например, если мы находимся в папке test, то репозиторий разместится тут test/DemoEx/.git  *! Этот способ самый предпочтительный. Папку можно переместить. В этом случае стартовые коммиты ветки master на сервере и локалке будут идентичными. Иначе будут проблемы с Pull Request* |
| **$ git remote add origin** https://github.com/MikhaskoS/ work.git  **$ git remote add origin** git@github.com:MikhaskoS/ work.git  **$ git push -u origin master** | Добавление связи локального репозитория с репозиторием на удаленном сервере.  Сперва нужно создать репозиторий в github (work). Переходим в отлеживаемую git папку на локальной машине и вводим две команды – первая для подключения и регистрации удаленного репозитория (он пропишется в конфигурационном файле .gitconfig), а вторая для переноса данных (из ветки master). Обычно удаленный репозиторий имеет имя origin.  ! Стартовые коммиты будут различные на gitHub и локалке. |
| **$ git remote add github**  https://github.com/MikhaskoS/ work.git | Добавить псевдоним (вместо origin теперь можно писать github) |
| **$ git remote** | Список доступных удаленных репозиториев |
| **$ git remote -v** | Список подключенных удаленных (remote) соединений |
| **$ git push -u origin master**  **$ git push -u github master**  **$ git push origin master**  **$ git push github master** | Отправка изменений на сервер |
| **$ git pull origin master** | Получение изменений с сервера (если там произошли изменения) одной ветки |
| **$ git fetch** | Получение **всех** изменений с сервера |

8. Удаленные ветки

|  |  |
| --- | --- |
| $ **git checkout -b** fix\_error | создать локальную ветку (fix\_error) |
| $ **git branch**  $ **git branch -r** | список текущих локальных веток  список текущих удаленных веток |
| $ **git push origin** fix\_error | отправить текущую ветку на удаленный репозиторий (ее можно увидеть на github) |
| $ **git push origin :**to\_delete | удалить ветку (to\_delete) на сервере |
| **Переименование веток** | |
| $ **git branch -m** fix\_error to\_delete | Переименование локальной ветки. На сервере она не переименуется! При копировании ее на сервер, там просто появится ее копия с другим именем. |
| Пример переименования ветки на удаленном сервере:  $ **git branch** new\_name origin/fix\_error  $ **git push origin** new\_name  $ **git push origin** :fix\_error | скопировать ветку с удаленного сервера  закинуть ее на сервер  удалить старую ветку |
| **Навигация по проекту** | |
|  |  |
|  |  |
| **Скрытие изменений** | |
|  |  |

9. Версионирование

|  |  |
| --- | --- |
| **Создание тега (метки)** | |
| Тег — это ссылка на коммит. У коммита может быть несколько тегов. | |
| $ **git tag** v1.3.0 | создание тега (для текущей ветки) |
| $ **git tag** **-a** v2.0.0 **-m** 'Тег с комментарием' | тег с комментарием (для текущей ветки) |
| $ **git tag** **-a** v1.2.0 **-m** ​ 'Old version'​ beefeb1 | тег на коммит с хешем beefeb1 |
| **Просмотр тегов** | |
| $ **git tag** | список доступных тегов |
| $ **git tag -l** v1.4.\* | список доступных тегов по шаблону |
| $ **git show** v1.4.0 | посмотреть описание коммита (вместо тега можно использовать и хеш) |
| $ **git show HEAD** | посмотреть последний коммит  HEAD – это встроенный тег для последнего коммита |
| **Удаление тегов** | |
| $ **git tag -d** v1.3.0 | удалить тег |
| **Удаленный репозиторий** | |
| $ **git push origin** v1.2.0 | отправить тег на удалённый репозиторий |
| $ **git push origin** **--tags** | отправить все теги на удаленный репозиторий |
| $ **git ls-remote --tags** origin | посмотреть теги на удаленном репозитории |
| $ **git push origin :**v1.2.0 | удалить тег с удаленного репозитория |
|  |  |

10. Слияние и переносы

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

Сдача работы. Работа в SmartGit

|  |  |
| --- | --- |
| С помощью консоли | |
| 1. Создаем репозиторий на GitHub  2. Запоминаем адрес |  |
| 3. В консоли git перемещаемся в папку, которую хотим отслеживать  4. Проводим инициализацию, как указано на сайте GitHub и переносим данные на удаленный репозиторий |  |
|  |  |
|  |  |