**Работа с Git**

Все глобальные настройки Git хранит в файле .gitconfig в вашей домашней директории.

Вывод текущей конфигурации:

git config –list

Изменение настроек:

git config --global user.name "Username"

git config --global user.email username@yandex.ru

Фикс кодировки вывода:

[chcp.com](http://chcp.com/) 1251

Инициализация репозитория:

$ cd ~/dev/first-project *# Переходим в нужную папку.*

git init *# Создаём репозиторий.*

Показать текущее состояние репозитория:

git status

Отметить все файлы для коммита как отслеживаемые:

git add --all

Зафиксировать изменения отслеживаемых файлов в репозитории с комментарием:

git commit -m "Мой первый коммит."

Связать локальный репозиторий с удаленным и присвоить ему псевдоним **origin**:

git remote add origin git@github.com:StRaiGT/first-project.git

Заменить адрес удаленного репозитория на нвоый:

git remote set-url origin https://github.com/kitty/rrrr.git

Поиск созданных ключей на компьютере:

ls -la ~/.ssh

Сгенерировать открытый и закрытый SSH ключ:

ssh-keygen -t ed25519 -C "kibat@yandex.ru"

Привязать SSH ключ к SSH-агенту (чтобы не вводить каждый раз пароль):

ssh-add ~/.ssh/id\_ed25519

Скопировать SSH-ключ:

clip < ~/.ssh/id\_ed25519.pub

Отправить код на GitHub в репозиторий origin на ветку master и связать её с локальной:

git push -u origin master

Получить код с GitHub:

git pull

Кнопка **fork** на GitHub позволяет скопировать код в свой репозиторий.

Клонировать репозиторий на локальный пк:

git clone https://github.com/StRaiGT/test\_homework.git

Скомпилировать файл в байтовый код:

javac -encoding UTF-8 HelloJdk.java

Скомпилировать все файлы с расширением \*.java в текущем расположении в байтовый код и сложить в папку bin:

javac -d bin -encoding UTF-8 \*.java

Запустить скомпилированный код:

java -Dfile.encoding=UTF-8 HelloJdk

Запустить скомпилированный код в папке bin, главный метод main находится в Practicum:

java -Dfile.encoding=UTF-8 -cp bin Practicum

Создать jar-файл с именем library и главным методом Practicum (точка указывает о включении всех файлов из каталога bin; возможно только для скомпилированного кода):

jar cfe library.jar Practicum -C bin .

Запустить jar-файл:

java -Dfile.encoding=UTF-8 -jar library.jar

Для исключения элементов из индексации можно настроить текстовый файл с расширением ***.gitignore*** и наполнить его именами или шаблонами не нужных файлов. Для шаблонов используются следующие символы:

* звездочка \* - любое количество символов
* вопросительный знак ? – любой отдельный символ
* косая черта / перед именем файла – игнорируется этот файл в этом расположении
* косая черта / после имени каталога – игнорируется весь каталог с содержимым
* парная звездочка \*\* – соответствует любому файлу, нулю или каталогам
* восклицательный знак ! – инвертировать правило игнорирования
* квадратные скобки […] – игнорирует любые символы внутри них
* решетка # – знак комментария

Показать все игнорируемые файлы в репозитории:

git status --ignore

Стандарт предлагает формат коммита: ***<type>: <subject>***. Субъект – краткое описание изменения в повелительном наклонении. Типы могут быть следующими:

* feat – добавление новой функциональности
* fix – исправление ошибок
* style – визуальные исправления в коде
* refactor – переработка кода
* delete – удаление кода
* perf – улучшение производительности
* docs – обновление документации
* test – добавление тестов

История коммитов проекта хранится в скрытой папке ***.git*** в репозитории в виде хэша. Для *Git* хэш является символьной строкой.

Вывести журнал коммитов:

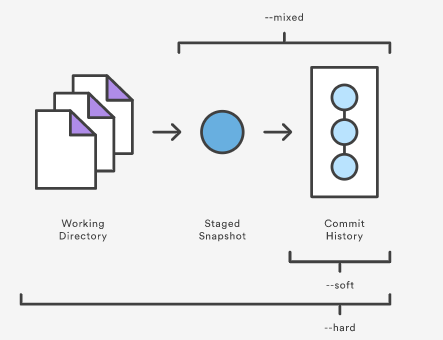
git log

Вывести краткий журнал коммитов (первые 7 символов хэша):

git log --oneline

Служебный файл *HEAD* хранит хэш и служит указателем на файл с последним коммитом. Посмотреть его содержимое:

cat HEAD



Вернуть проиндексированные файлы к состоянию при последнем коммите (не затрагивает файлы, с которыми вы работаете в текущий момент и которые еще не добавлены в отслеживаемые):

git reset HEAD

Вернуть проиндексированные файлы к состоянию при последнем коммите (удаляет все незакоммиченные изменения из рабочей зоны и в отслеживаемых):

git reset --hard

При указании краткого хэша коммита можно откатиться к нему.

git reset –hard хеш\_коммита

Перезаписать коммит:

git commit --amend

Выделить изменения сделанные после крайнего коммита (для файлов еще не добавленных в отслеживаемые командой *“git add”*):

git diff

Выделить изменения сделанные после крайнего коммита (для файлов уже добавленных в отслеживаемые):

git diff --staged

Сравнить изменения между двумя коммитами:

git diff хэш\_коммита\_1 хэш\_коммита\_2

Посмотреть ветки:

git branch

Посмотреть все удаленные ветки:

git branch -r

Посмотреть все ветки:

git branch -a

Создать новую ветку:

git branch имя\_новой\_ветки

Перейти в другую ветку:

git checkout имя\_ветки

Создать ветку и сразу перейти в нее:

git checkout -b имя\_новой\_ветки

Удалить ветку:

git branch -d имя\_ветки

Удалить ветку на гитхабе:

git push origin -d имя\_ветки

Объединить ветки:

git merge имя\_ветки\_ присоединяемой\_в\_текущую

Отменить слияние веток:

git reset --hard HEAD~

или

git revert -m 1 хэш\_коммита\_слияния

Во втором способе, если продолжить работу в дополнительной ветке, то при повторном слиянии в главную ветку попадут только те изменения, которые были сделаны после отмены слияния. Решение - создать новую ветку из той, слияние с которой было отменено, сделать коммит, затем перейти в main и выполнить новое слияние

Отложить (спрятать без фиксации в коммите) локальное состояние репозитория со всеми наработками:

git stash

Посмотреть список спрятанных изменений (ноль = последнее изменение):

git stash list

Вернуть последний спрятанный набор изменений:

git stash pop

Вернуть конкретный спрятанный набор изменений:

git stash apply stash@{номер\_изменений}

Копилка позволяет спрятать свои наработки, обновить ветку и снова вернуть наработки из копилки. Подробнее про копилку можно почитать [тут](https://git-scm.com/book/ru/v2/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-Git-%D0%9F%D1%80%D0%B8%D0%BF%D1%80%D1%8F%D1%82%D1%8B%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%B8-%D0%BE%D1%87%D0%B8%D1%81%D1%82%D0%BA%D0%B0).

Загрузить все коммиты удаленного репозитория:

git fetch

Запустить процесс слияния веток:

git merge

Онлайн-песочница по Git [тут](https://learngitbranching.js.org/?locale=ru_RU).

**Оформление файла Readme.md**

Заголовки разных уровней создаются решетками:

*# H1*

*## H2*

*### H3*

*#### H4*

*##### H5*

*###### H6*

Добавить черту под заголовком:

*Alt-H1*

*------*

Выделить текст курсивом:

*Курсив — это \*звёздочки\* или \_подчёркивания\_.*

Полужирный шрифт:

*Полужирный шрифт — двойные \*\*звёздочки\*\* или \_\_подчёркивания\_\_.*

*Можно совместить выделение \*\*звёздочки и \_подчёркивания\_\*\*.*

Зачеркнутый текст:

*~~Зачёркнутый текст.~~*

Создание нумерованного списка:

*1. Первый пункт нумерованного списка.*

*2. Второй пункт.*

Создание ненумерованного списка:

*\* первый пункт ненумерованного списка;*

*\* второй пункт ненумерованного списка*

Ссылки из текста:

*[Обычная ссылка](https://www.yandex.ru)*

Ссылка без текста:

*[текст ссылки]*

Ссылка с тайтлом (всплывающая подсказка):

*[Обычная ссылка с title](https://www.yandex.ru "Я Yandex!")*

Оформление текста как кода:

*```html // тут указывается язык программирования*

*<h1>А я просто текст</h1>*

*```*