**РАБОТА С GIT**

**Справочник:** [**http://git-scm.com/book/ru/v2/**](http://git-scm.com/book/ru/v2/)

Конфликтует с Google Drive

**Установка**

Качаем <http://windows.github.com>

Пакет так же устанавливает клиент git

При установке указать:

1. Windows Explorer integration - Git Bash Here и Git GUI Here
2. Use Git from the Windows Command Prompt
3. core.autocrlf set to true

**Быстрый старт ч.1: создание локального репозитория**

1. **Создаем папку** где нужно, заливаем внутрь все нужные файлы, заходим внутрь папки
2. Создаем создаем текстовый файл, там просто перечисляем папки, которые должны будут игнорироваться git: .idea. Сохраняем текстовый файл без расширения и под названием **".gitignore."**

Проводник удалит точку в конце самостоятельно

1. Жмем правой кнопкой по пустому пространству внутри директории - > **Git Bash here**
2. Пишем **git init** в открывшемся окне.
3. Пишем **git add --all** что бы добавить все файлы из текущей директории и всех поддиректориев, или **git add .** что бы добавить просто файлы из текущей папки. Файлы, прописанные в .gitignore включены не будут
4. Пишем **git status**, показывает добавленные файлы.
5. **git rm --cached название\_файла** удалит ненужный файл, который можно сразу же внести в git ignore. Что бы удалить директорию с файлами внутри, пишем **git rm -r --cached**
6. **git commit -m "root dir created**" - коммитим файлы, первый коммит удалить нельзя.
7. Что бы добавить изменения в последний коммит, пишем **git commit -m "comment" --amend**

Или так: **git commit --amend --no-edit**. Ключ --no-edit нужен, чтобы не вводить заново комментарий

Если изменения вносятся в файлы, которые уже были ранее добавлены в через git add, можно написать так: **git commit -a -m "comment"**

1. Вывод списка коммитов с именами хешов**: git log --oneline**
2. Что бы узнать изменения по конкретному коммиту, пишем **git show 657ab4c --oneline**

хеш можно узнать из команды сверху.

1. Для отмены последнего коммита писать **git reset HEAD~1**
2. Что бы удалить все файлы, которые не сохранены в репозитории: **git clean -df**

**Быстрый старт ч.2: залитие файлов на GitHub**

1. Идем на hithub и создаем там новый репозиторий.
2. Копируем и выполняем указанные там команды в нашем окне Git.

* **git remote add origin git@github.com:myuser/project.git** - создан удаленный реп
* **git push -u github master -** приставка -u для установки первичной связи с удаленным репо
* Смотрим результат: **git remote -v**

1. Перенос репозитория на другой комп:

**git clone https://github.com/myuser/project.git <myfolder>**

1. Для получения обновлений с удаленного репозитория: **git pull**

* Если локальные фалы были изменены, git pull выдаст ошибку. Что бы перезаписать локальные файлы:

**git fetch --all**

**git reset --hard https://github.com/en1ight/testing\_personal.git**

1. Если в удаленном реп лежат файлы более новой версии, git push выдаст ошибку. Что бы перезаписать файлы несмотря на конфликт версий, пишем **git push -f**
2. Иногда возникает необходимость отложить ваши текущие изменения и поработать над файлами, которые находятся в удаленном репозитории. Для этого отложите текущие изменения командой: **git stash pop**

После выполнения этой команды ваша локальная директория будет содержать файлы такие же, как и при последнем коммите. Вы можете загрузить новые файлы из удаленного репозитория командой **git pull** и после этого вернуть ваши изменения которые вы отложили командой: **git stash pop**

**Настройка**

git config --global user.name "Alexandr Bobrenko"

git config --global user.email [**oknerbob@gmail.com**](mailto:oknerbob@gmail.com)

Если для каких-то отдельных проектов вы хотите указать другое имя или электронную почту, можно выполнить эту же команду без параметра **--global** в каталоге с нужным проектом.

git config --global core.editor emacs // выбор редактора

**git config --list** //узнать конфигурацию

git config user.name //узнать конкретный ключ

**git help config**  //хелп по командам. В данном случае - **config**

**Работа с файлами**

После того, как нужный файл был изменен, проверка git status покажет, какие файлы были изменены:

git add index.html //проиндексирует файл (сохранит изменения локально)

git checkout index.html //отменит изменения в файле

Если изменения не хотим коммитить: **git reset HEAD index.html** //снять индексацию изменений

Предположим, что мы отредактировали три файла (**a.html, b.html, and c.html**). Теперь мы хотим закоммитить все изменения, при этом чтобы изменения в a.html и b.html были одним коммитом, в то время как изменения в c.html логически не связаны с первыми двумя файлами и должны идти отдельным коммитом. В теории, мы можем сделать так:

git add a.html

git add b.html

git commit -m "Changes for a and b"

**git add c.html** // Разделяя индексацию и коммит, мы имеем возможность

**git commit -m "Unrelated change to c"** //с легкостью настроить, что идет в какой коммит.

Если хотим закоммитить. После того, как были добавлены:

**git commit -m "changes in h1"** //сохраняем файлы в проект. -m означает коммент в строке

**git commit** //или так, открывается текстовый редактор

**История**

**git log**  // показать историю изменений

**git log --pretty=oneline** //та же история, только в одну строку

git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short

**--pretty="..."** — определяет формат вывода.

**%h** — укороченный хэш коммита

**%d** — дополнения коммита («головы» веток или теги)

**%ad** — дата коммита

**%s** — комментарий

**%an** — имя автора

**--graph** — отображает дерево коммитов в виде ASCII-графика

**--date=short** — сохраняет формат даты коротким и симпатичным

**q** - выйти из режима лога

**Алиасы**

Добавьте следующее в файл .gitconfig в вашем $HOME каталоге:

[alias]

co = checkout

ci = commit

st = status

br = branch

hist = log --pretty=format:\"%h %ad | %s%d [%an]\" --graph --date=short

type = cat-file -t

dump = cat-file -p

После этого, в консоли можно прописать **git hist** для получения истории по этому алиасу

**Получение старых версий**

**git hist**  //получаем историю, нас интересует хеш

**git checkout <hash>** //вставляем номер нужного хеша, первых 7 символов будет достаточно

**cat hello.html** //вывести в консоль содержимое файла

**git checkout master** //«master» — имя ветки по умолчанию. Переключая имена веток, вы попадаете на последнюю версию выбранной ветки.

**Создание имен для версий файлов (тегов)**

**git tag v1** //присваивает тег v1 текущей (открытой) версии файла

**git checkout v1^** // перейти к предшествующей версии файла (родителю)

**git checkout v1~1** //тоже самое, переход к предыдущей версии

**git checkout v1**  //перейти к файлу под названием v1

**git tag** //посмотреть, какие теги доступны

**git hist master --all** //Вы можете видеть теги (v1 и v1-beta) в логе вместе с именем ветки (master). Кроме того HEAD показывает коммит, на который вы переключились (на данный момент это v1-beta).

**Отмена локальных изменений (до индексации)**

Если изменения были внесены и сохранены локально (в редакторе), гит может их откатить:

git checkout master //проверяем, находимся ли мы на последней версии файла  
git checkout hello.html //тут происходит отмена изменений

git status //проверяем