

# ВНЕДРЕНИЕ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ



**ИЛЬНАЗ ГИЛЬЗОВ**



# ИЛЬНАЗ ГИЛЬЯЗОВ

Технический директор в aims



[coursar@gmail.com](mailto:coursar@gmail.com)



# ПЛАН ЗАНЯТИЯ

1. [Контроль версий](#)
2. [Git](#)
3. [Работа в локальном репозитории](#)
4. [.gitignore](#)
5. [Пара слов о редакторах](#)
6. [Github](#)
7. [Markdown](#)



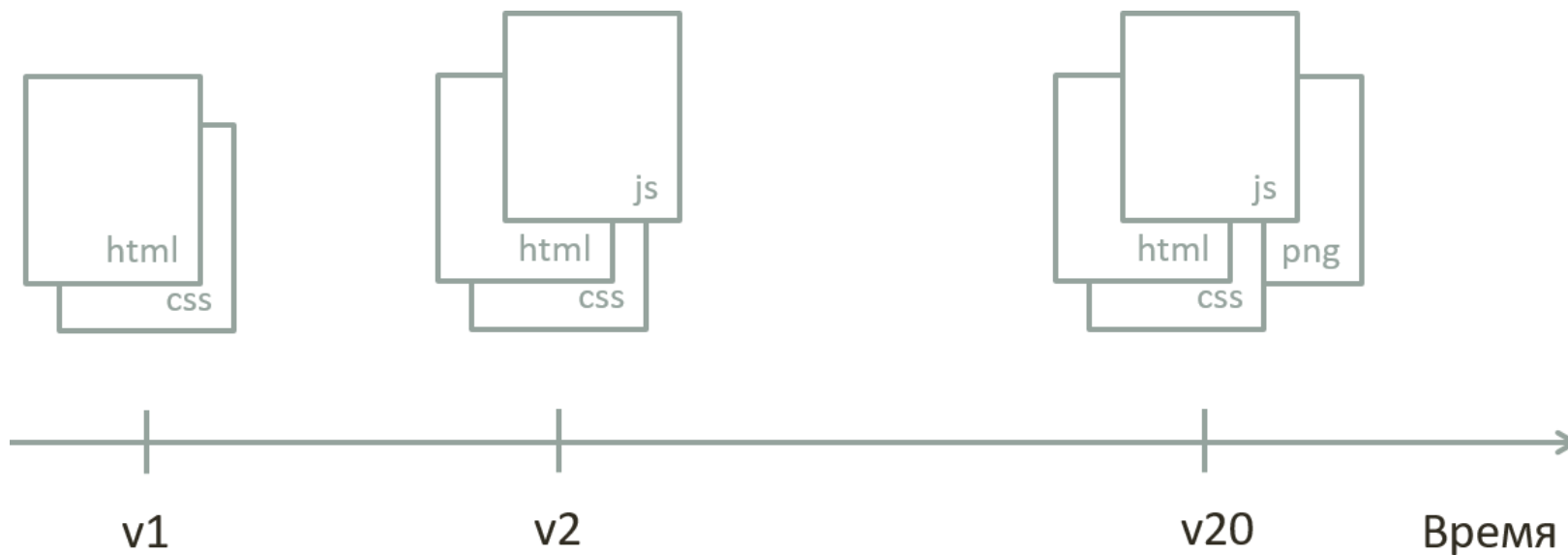
# КОНТРОЛЬ ВЕРСИЙ



# КОНТРОЛЬ ВЕРСИЙ

Представим следующую ситуацию: вы создаёте посадочную страницу. Всё хорошо, и со временем вы туда добавляете новые блоки, специальные акции (распродажи), формы обратной связи и т.д.

# КОНТРОЛЬ ВЕРСИЙ

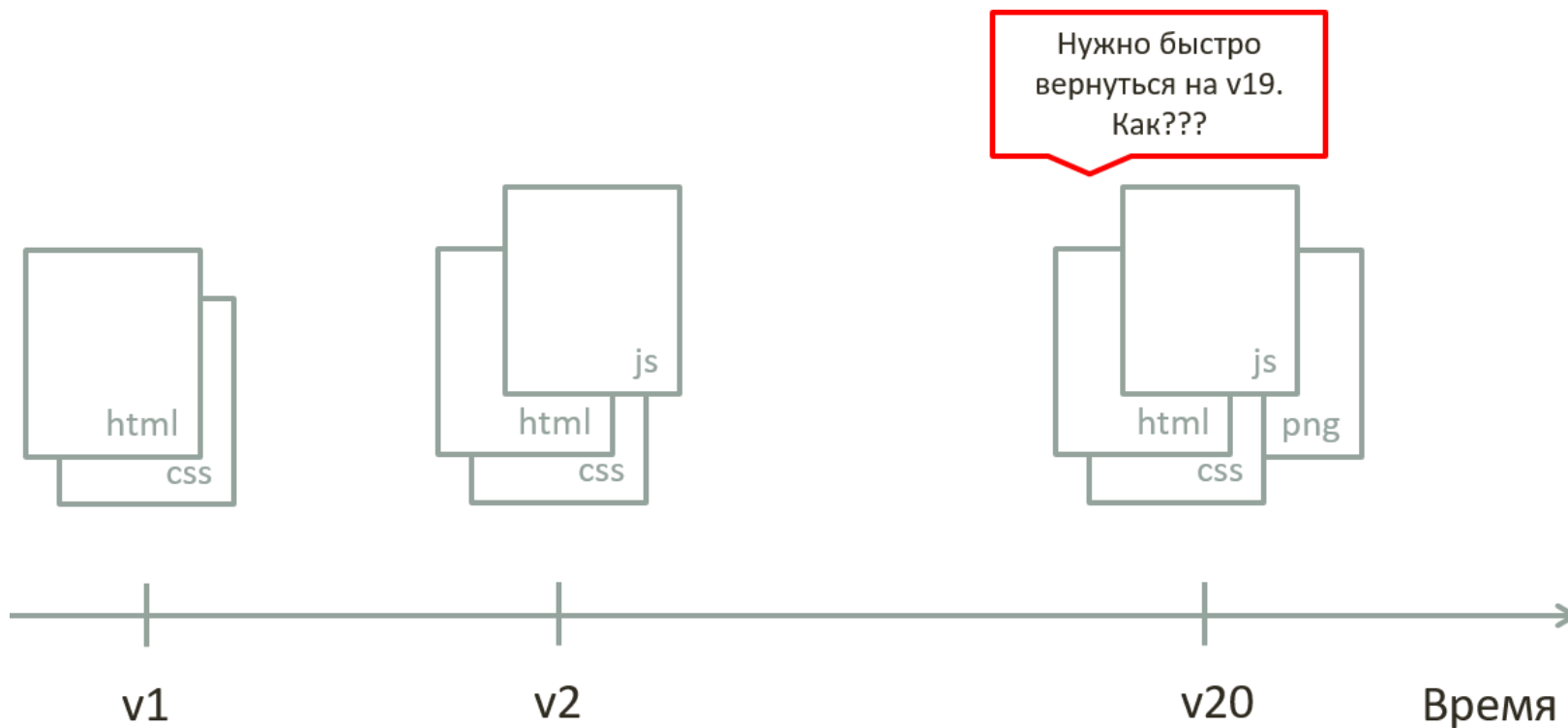




# КОНТРОЛЬ ВЕРСИЙ

И вдруг выясняется, что ваша версия содержит критическую ошибку, и нужно срочно вернуть всё так, как было в предыдущей версии. Что делать?

# КОНТРОЛЬ ВЕРСИЙ







# КОНТРОЛЬ ВЕРСИЙ

Можно, конечно, как студенты, хранить много версий файлов (или даже папок), например:

- /сайт\_v1
- /сайт\_v2
- ...
- /сайт\_v20



## ПРОБЛЕМЫ ТАКОГО ПОДХОДА:

- Случайно удалили каталог;
- Случайно удалили файл в каталоге;
- Случайно изменили файл в каталоге с предыдущей версией;
- Параллельно с коллегой меняете один и тот же файл.

Поэтому нужно использовать специальные системы, которые защищают нас от этих проблем и позволяют сделать процесс контроля версий более удобным.



# VCS

Системы контроля версий (VCS, от Version Control System) – системы, позволяющие удобно создавать новые версии файлов, отслеживать историю изменений и при необходимости возвращаться к предыдущим версиям, определять авторство изменений, смотреть различия и многое другое.



## КЛЮЧЕВЫЕ ФУНКЦИИ VCS

1. Создание новых версий файлов (фиксация изменений);
2. Откат изменений (возврат к предыдущим версиям);
3. Работа над параллельными изменениями;
4. Работа в команде;
5. Анализ истории и авторства изменений;
6. Резервная копия проекта.



# GIT

Распределённая система контроля версий, одна из самых популярных на текущий момент.

# УСТАНОВКА

- Windows: <https://git-scm.com/download/win>
- Mac: <https://git-scm.com/download/mac>
- Ubuntu: `apt-get install git`
- Fedora: `dnf install git`
- Другие версии Linux: <https://git-scm.com/download/linux>

# НАСТРОЙКА

Git хранит историю в привязке к имени пользователя и email.

Чтобы задать имя пользователя и email необходимо выполнить следующие команды:

```
git config --global user.name "Vasya Pupkin"  
git config --global user.email vasya@localhost
```




# РЕДАКТОР

Git использует по умолчанию редактор Vim (он достаточно тяжёл для начинающих пользователей).

Мы можем задать редактор попроще (Nano) с помощью команды:

```
git config --global core.editor nano
```



# РАБОТА В ЛОКАЛЬНОМ РЕПОЗИТОРИИ



# ЗАДАЧА

Решим следующую задачу: у нас есть проект веб-сайта, состоящий из нескольких файлов и каталогов. Задача состоит в том, чтобы перевести работу с этим сайтом в Git.



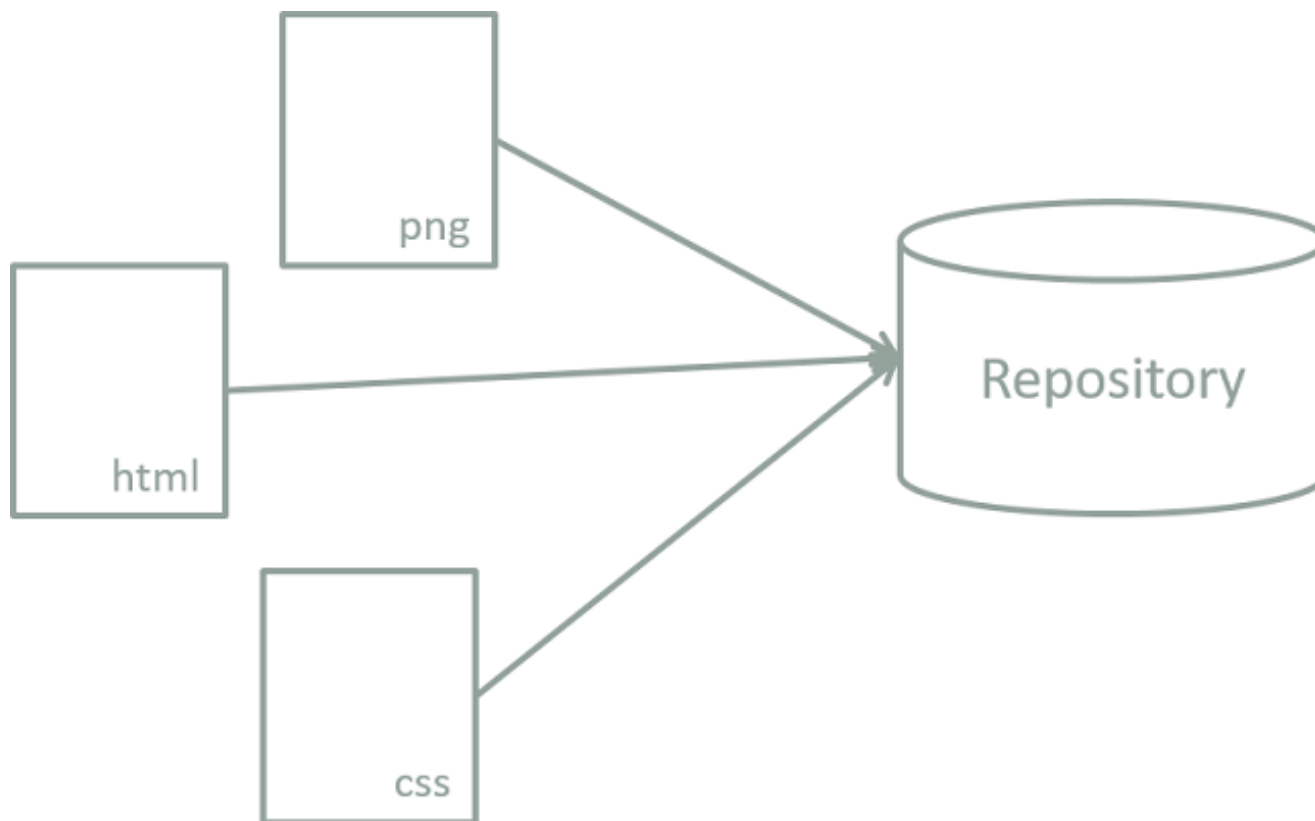
# ОБЩАЯ СХЕМА РАБОТЫ

1. Создание локального репозитория для проекта;
2. Добавление файлов (изменённых) в список отслеживания (stage area или index);
3. Фиксация изменений в файлах (commit).

Операции 2 и 3 выполняются в течение всего развития проекта.

# РЕПОЗИТОРИЙ

Репозиторий – это хранилище истории и данных (файлов, каталогов) вашего проекта.



# СОЗДАНИЕ РЕПОЗИТОРИЯ

Репозиторий создаётся в конкретном каталоге с помощью команды:

```
$ git init
```

```
Initialized empty Git repository in C:/project/.git/
```

## КАТАЛОГ `.git`

В результате выполнения команды `git init` появится подкаталог `.git` в котором и будут храниться служебные настройки git'a и сам репозиторий.

# ДОБАВЛЕНИЕ ФАЙЛОВ

Git будет следить только за теми файлами, которые вы добавили в «список отслеживаемых».

Сделать это можно с помощью команды:

```
$ git add index.html
```

где `index.html` – это имя добавляемого файла.



# МАСКИ ФАЙЛОВ

Есть ряд специальных символов, которые позволяют упростить процесс работы с файлами:

Специальный символ `*` означает любую последовательность символов.

- `git add *` добавит в список отслеживаемых файлов все файлы в текущем каталоге и подкаталогах;
- `git add *.js` добавит в список отслеживаемых файлов все файлы с расширением `.js` в текущем каталоге и подкаталогах.

# ТЕКУЩИЙ СТАТУС

Команда `git status` позволяет отследить текущий статус нашего репозитория:

```
$ git status
```

```
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   README.md
new file:   css/style.css
new file:   index.html
new file:   js/app.js
```

# ФИКСАЦИЯ ИЗМЕНЕНИЙ (КОММИТ)

Для фиксации изменений используется команда `git commit` (после этого откроется редактор для указания комментария). В редакторе nano нужно будет нажать клавиши Ctrl + O, Ctrl + X для записи и выхода из редактора (см. сокращения).

```
GNU nano 3.1 C:/project/.git/COMMIT_EDITMSG
|
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch
#
# Initial commit
#
# Changes to be committed:
#   new file:   README.md
#   new file:   css/style.css
#   new file:   index.html
#   new file:   js/app.js
#
```

# ФИКСАЦИЯ ИЗМЕНЕНИЙ (КОММИТ)

Комментарий является обязательным атрибутом коммита.

```
$ git commit
[master (root-commit) 4ad7e09] Первоначальная версия
4 files changed, 14 insertions(+)
create mode 10064 README.md
create mode 10064 css/style.css
create mode 10064 index.html
create mode 10064 js/app.js
```



# КОММЕНТАРИЙ

Комментарий коммита очень важен – в дальнейшем при просмотре истории, по тексту комментария вы сможете понять, какие вы изменения были внесены.

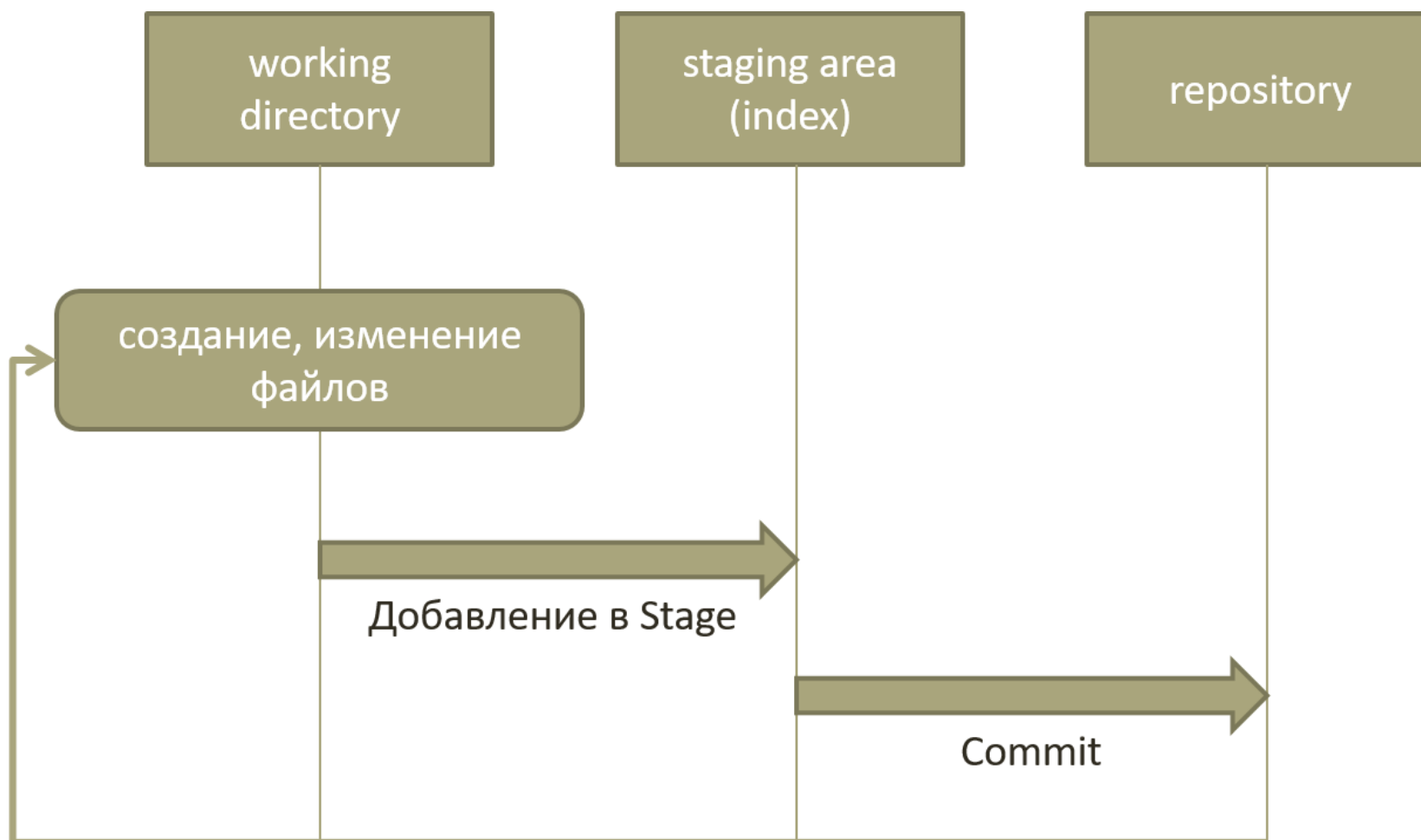
Поэтому старайтесь писать краткие, но отражающие суть изменений комментарии.

## ПОСЛЕ ФИКСАЦИИ

После коммита мы получаем «чистое» состояние, готовое к новому добавлению изменений и фиксации (`git status`)

```
$ git status  
On branch master  
nothing to commit, working tree clean
```

# НА ДИАГРАММЕ



# НОВАЯ ЗАДАЧА: GOOGLE FONTS

Нам пришла новая задача – добавить Google Fonts на наш сайт. Для этого изменим файл `index.html` (добавим шрифт Roboto).

Для этого в секцию `head` добавим тег `link`:

```
<link href="https://fonts.googleapis.com/css?family=Roboto&subset=cyrillic"
      rel="stylesheet">
```



# СМОТРИМ СТАТУС

Посмотрим статус после изменения:

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

# СМОТРИМ СТАТУС

Теперь согласно схеме, мы должны:

1. Добавить файл в `stage` с помощью команды `git add index.html`
2. Закоммитить изменения с помощью команды `git commit`.

```
$ git commit -a -m "Добавлены Google Fonts"
[master 075a656] Добавлены Google Fonts
1 file changed, 1 insertion(+)
```

## СОКРАЩЕНИЯ

Флаг `-a` (`--all`) говорит о том, что мы добавляем в `stage` все удалённые/изменённые файлы (но не новые, новые нужно добавлять отдельно).

Флаг `-m "Сообщение коммита"` (`--message="Сообщение коммита"`) позволяет не открывать редактор, а указывать сообщение прямо в командной строке.

## КОГДА ДЕЛАТЬ `git add`

**Важно запомнить следующий момент:** команда `git commit` фиксирует только те изменения, которые добавлены были в `staging area` через `git add`.

Поэтому если вы сделаете `git add` для файла, а затем измените его и сделаете `git commit`, то ваши последние изменения не зафиксируются, так как вы не сделали `git add`.

# СОКРАЩЕНИЯ

Поскольку операции добавления в `stage` файлов и их фиксации очень частые, то в Git для этого есть специальное сокращение, позволяющее сделать всё одной командой:

```
$ git commit -a -m "Добавление Google Fonts"
```

# ИСТОРИЯ

Посмотреть историю коммитов можно с помощью команды `git log`.

```
$ git log
commit 075a656ac42d7b892e12f309cdc547603ad53577 (HEAD -> master)
Author: Vasya Pupkin <vasya@localhost>
Date:   Fri Oct 26 19:42:48 2018 +0400
```

Добавлены Google Fonts

```
commit 4ad7e09ee8d76ce6c6f25e233cf186689b3b55b2
Author: Vasya Pupkin <vasya@localhost>
Date:   Fri Oct 26 19:26:05 2018 +0400
```

Первоначальная версия

# ИДЕНТИФИКАТОР КОММИТА

Перед коммитом файла Git вычисляет контрольную сумму, которая является идентификатором коммита.

```
$ git commit
[master (root-commit) 4ad7e09] Первоначальная версия
4 files changed, 14 insertions(+)
create mode 10064 README.md
create mode 10064 css/style.css
create mode 10064 index.html
create mode 10064 js/app.js

$ git log
commit 4ad7e09ee8d76ce6c6f25e233cf186689b3b55b2 (HEAD -> master)

Author: Vasya Pupkin <vasya@localhost>
Date:   Fri Oct 26 19:26:05 2018 +0400

    Первоначальная версия
```

# ИДЕНТИФИКАТОР КОММИТА

Посмотреть полную информацию о коммите мы можем с помощью команды `git show <id-коммитта>`. ID не обязательно писать целиком, достаточно первых нескольких символов:

```
$ git show 4ad7e0
commit 4ad7e09ee8d76ce6c6f25e233cf186689b3b55b2
```

```
Author: Vasya Pupkin <vasya@localhost>
```

```
Date:   Fri Oct 26 19:26:05 2018 +0400
```

Первоначальная версия

```
diff --git a/README.md b/README.md
```

```
new file mode 100644
```

```
index 0000000..e69de29
```

```
...
```





# ЕСЛИ ЧТО-ТО ПОШЛО НЕ ТАК

Давайте посмотрим, что можно сделать, если что-то пошло не так.

# ИСПРАВЛЯЕМ ОШИБКИ

Если вы случайно добавили в `stage area` файл, который добавлять не нужно, то удалить его можно командой:

```
$ git rm --cached stuff.txt
```

где `stuff.txt` – имя файла.

## ИСПРАВЛЯЕМ ОШИБКИ

Если вы случайно зафиксировали коммит с ошибочным комментарием, то исправить комментарий последнего коммита можно с помощью команды:

```
$ git commit --amend -m "Roboto Font"
```

где "Roboto Font" – новое сообщение коммита.

# ИСПРАВЛЯЕМ ОШИБКИ

Если вы залили коммит с ошибкой, то можно создать «зеркальный» коммит, который отменит действие предыдущего:

```
$ git revert <commit-id>
```

где `<commit-id>` – идентификатор коммита.

Идентификатор коммита можно посмотреть с помощью команды `git log`.

# СПРАВКА

Вся справка по командам предоставляется самим Git:

- `git` – краткая справка по git;
- `git help <command>` – справка по конкретной команде git.



`.gitignore`

# ИГНОРИРОВАНИЕ ФАЙЛОВ

Часто бывает, что в проектах бывают какие-то файлы, которые не нужно хранить в репозитории, например:

- файлы проектов тестовых редакторов;
- файлы операционных систем;
- генерируемые данные (результаты сборки, компиляции и т.д.);
- файлы с учётными данными;
- и любые другие, которые вы не хотите хранить в репозитории.

**Помните, всё, что хранится в репозитории будет доступно всем.**

# ИГНОРИРОВАНИЕ ФАЙЛОВ

Для того, чтобы игнорировать подобные файлы в Git есть специальный настроечный файл, который называется `.gitignore` (точка в начале имени файла обязательна!).

В нём мы можем перечислить файлы и каталоги, которые Git должен игнорировать при работе.



## `.gitignore`

Это обычный текстовый файл, где на каждой строке обычно размещается одно правило игнорирования.

Обычно этот файл располагается в самом верхнем каталоге проекта и хранится в репозитории.

Строки, начинающиеся с символа `#` являются комментарием и не воспринимаются как правила.

## ПРИМЕР `.gitignore`

```
# будут игнорироваться все файлы и каталоги Thumbs.db
# вне зависимости от того, в каком каталоге они находятся
Thumbs.db

# будет игнорироваться каталог tmp
# вне зависимости от того, в каком каталоге он находится
# слэш в конце указывает, что это каталог
tmp/

# будет игнорироваться относительно файла .gitignore
# чаще всего относительно всего проекта
/tmp/

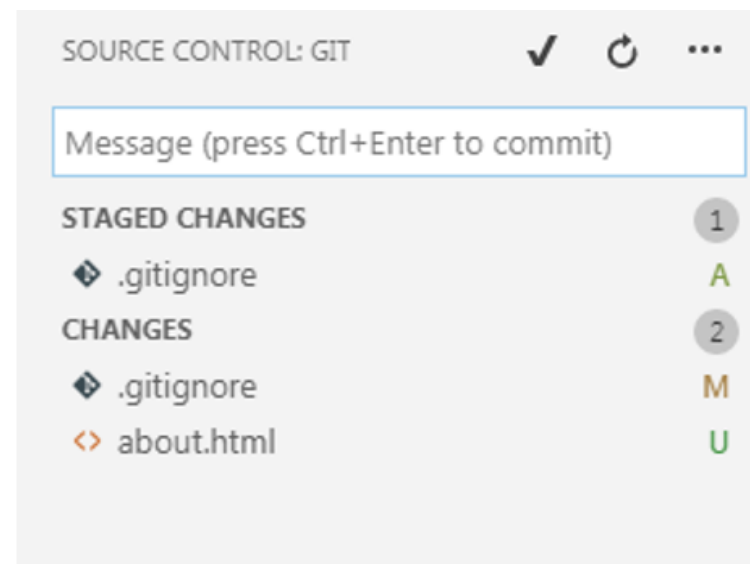
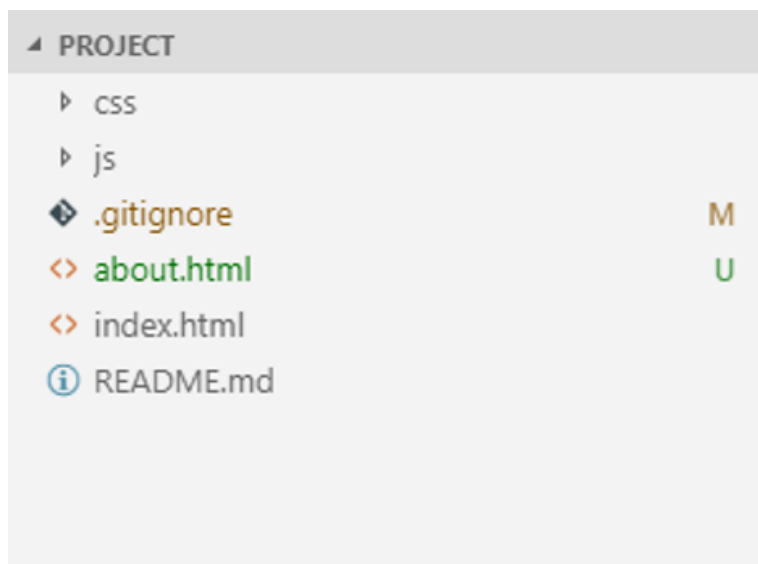
# будут игнорироваться все файлы и каталоги с расширением .txt
# вне зависимости от того, в каком каталоге они находятся
*.txt
```

---

# ПАРА СЛОВ О РЕДАКТОРАХ

# ПОДДЕРЖКА GIT

Большинство редакторов кода и сред разработки уже содержат либо встроенную поддержку Git, либо интегрируют её с помощью плагинов.



Несмотря на то, что визуальное представления часто удобнее, нужно уметь работать из консоли (т.к. некоторые вещи можно сделать только там).



**GITHUB**



## РЕЗЕРВНАЯ КОПИЯ

Достаточно опасно хранить всю историю работы с нашим проектом только на нашем жёстком диске (локально) – при возникновении какой-либо проблемы с ним или компьютером мы можем на достаточно долгое время потерять доступ к исходным кодам проекта (если не навсегда).

Конечно, есть сервисы вроде Dropbox, Google Drive, Яндекс.Диск и другие, но гораздо удобнее использовать специализированные решения, интегрированные с Git.



# РЕГИСТРАЦИЯ

Детально регистрация по шагам приведена в материалах к данной лекции.

Возникли ли у вас проблемы, которые не получилось решить?



# НАЧАЛО РАБОТЫ

В качестве базовых сценариев начала работы мы на данной лекции выделим два:

1. У вас есть локальный репозиторий с проектом, вы хотите к нему привязать репозиторий на GitHub;
2. У вас нет локального репозитория, вы хотите начать новый проект.

На самом деле, и в том, и в другом случае, нам понадобится репозиторий на GitHub, поэтому посмотрим как его создавать.



# СОЗДАНИЕ РЕПОЗИТОРИЯ НА GITHUB

Авторизируйтесь на GitHub и выберите «Start a Project»:

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

[Read the guide](#) [Start a project](#)

Welcome, Nat!  
Hear from Nat Friedman, our new CEO, on the future of GitHub.

Our new Terms of Service and Privacy Statement are in effect.

Repositories [New repository](#)

You don't have any repositories yet!

Browse activity [Discover repositories](#)

Discover interesting projects and people to populate your personal news feed.

Your news feed helps you keep up with recent activity on repositories you [watch](#) and people you [follow](#).

[Explore GitHub](#)

# СОЗДАНИЕ РЕПОЗИТОРИЯ НА GITHUB

На следующей странице самое главное – выбрать имя для вашего репозитория (1) и нажать на кнопку «Create Repository» (5):

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: netology-git / **Repository name: demo** ✓ (1)

Great repository names are short and memorable. Need inspiration? How about *ideal-guacamole*.

Description (optional): (2)

☒ Public  
Anyone can see this repository. You choose who can commit.

☐ Private  
You choose who can see and commit to this repository.

☐ Initialize this repository with a README (2)  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None (3) Add a license: None (4)

**Create repository** (5)

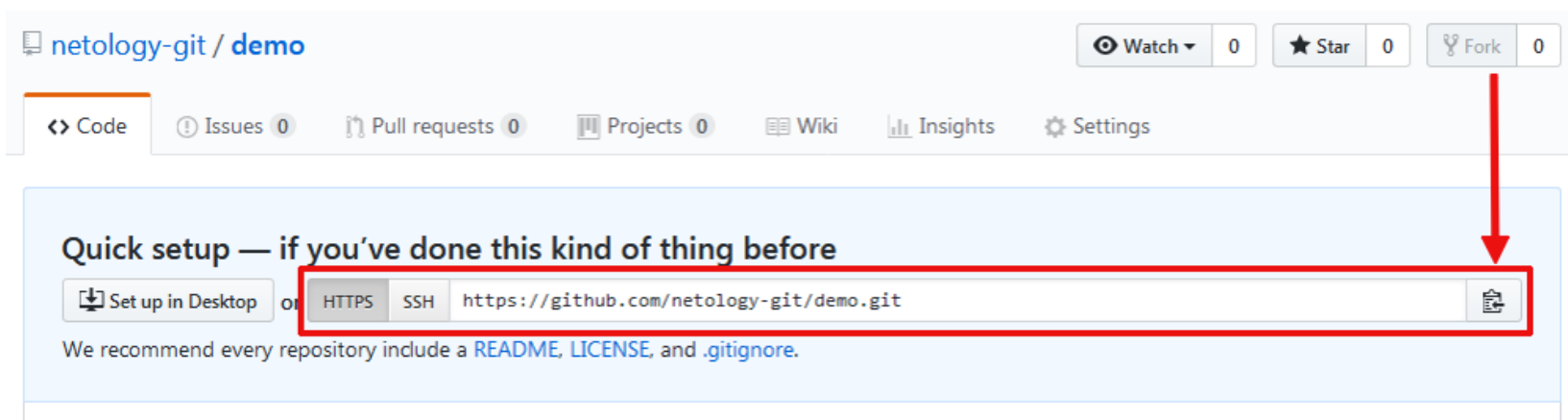
# СОЗДАНИЕ РЕПОЗИТОРИЯ НА GITHUB

1. **Name**
2. **Description** – имя репозитория;
3. **Include README** – автоматическое создание `README.md` (будем обсуждать чуть позже);
4. **Add .gitignore** – добавление преднастроенного `.gitignore`;
5. **Add a license** – добавление информации о лицензии.

В рамках этой лекции нам нужны только пункты 1 и 2, остальное всё оставьте пустым.

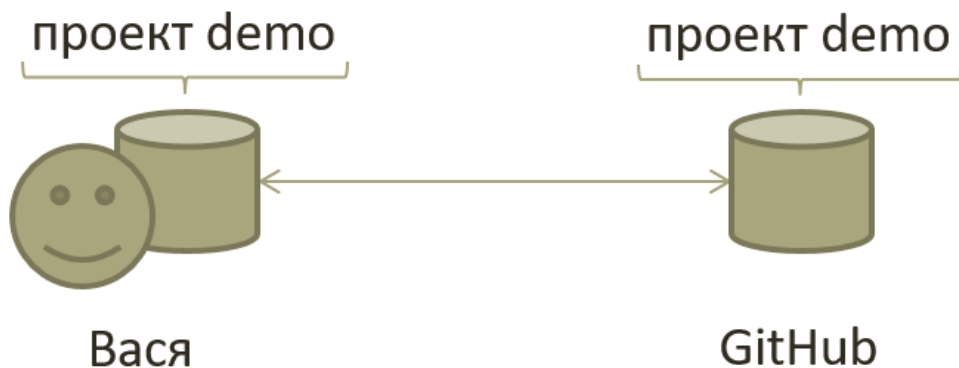
# URL РЕПОЗИТОРИЯ

Для дальнейших операций нам необходим URL, по которому располагается созданный нами репозиторий:



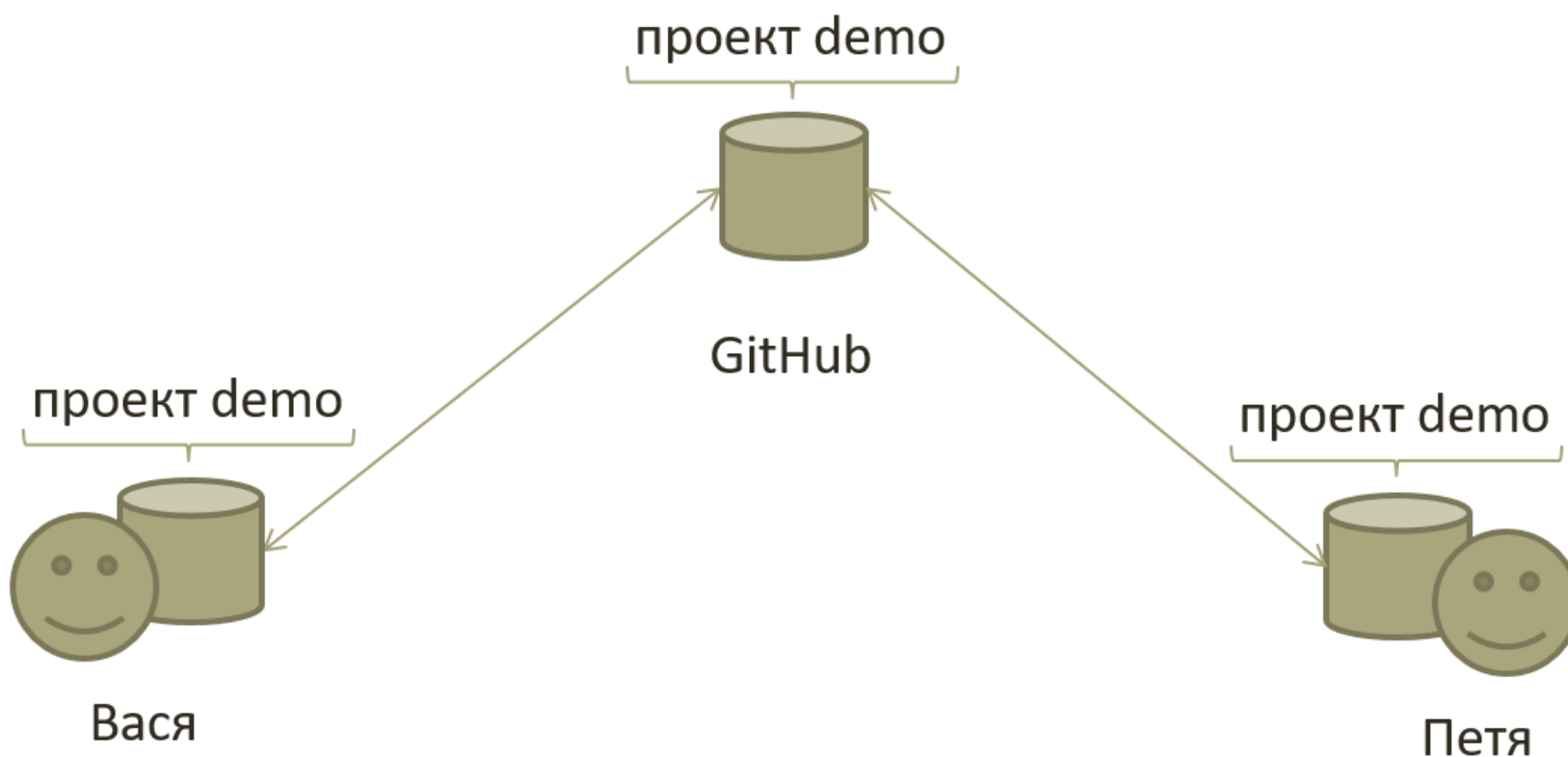
# УДАЛЁННЫЕ РЕПОЗИТОРИИ

Git позволяет нам привязать к нашему репозиторию удалённый репозиторий, так, чтобы мы могли туда отправлять изменения из своего репозитория и получать обновления с удалённого репозитория (если туда их ещё кто-то отправляет).



# УДАЛЁННЫЕ РЕПОЗИТОРИИ

Удалённые репозитории могут использоваться для совместной работы и привязать их можно много.



# КЛОНИРОВАНИЕ РЕПОЗИТОРИЯ

Операция `git clone` позволяет нам клонировать удалённый репозиторий, т.е. буквально взять почти всю информацию, хранящуюся в удалённом репозитории и создать копию на нашем компьютере – это очень здорово, потому что теперь даже если с удалённым репозиторием что-то случится, вся история разработки проекта со всеми версиями будет на нашем компьютере.

Чтобы его клонировать, нам необходим URL, по которому располагается этот репозиторий.

# КЛОНИРОВАНИЕ РЕПОЗИТОРИЯ

После того, как у нас есть URL мы можем в командной строке выполнить команду `git clone`:

```
$ git clone https://github.com/netology-git/demo.git
Cloning into 'demo'...
warning: You appear to have cloned an empty repository

$ ls
demo/
```

После успешной операции клонирования, создастся каталог `demo`, в котором будет наш репозиторий (уже привязанный к удалённому).



# КЛОНИРОВАНИЕ РЕПОЗИТОРИЯ

*Важно: не обязательно клонировать пустой репозиторий, вы можете клонировать любой репозиторий, зная его URL.*

URL вы можете найти на странице проекта:

The screenshot shows the GitHub interface for the repository 'netology-git / demo'. At the top, there are buttons for 'Watch', 'Star', and 'Fork', each with a count of 0. Below these are tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The 'Code' tab is selected. Below the tabs, there is a message: 'No description, website, or topics provided.' and a 'Manage topics' link. Below this, there is a summary bar showing '3 commits', '1 branch', '0 releases', and '0 contributors'. Below the summary bar, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', and 'Find file'. The 'Clone or download' button is highlighted with a red box and a red circle with the number 1. Below this, there is a dropdown menu for cloning the repository. The 'Clone with HTTPS' option is selected, and the URL 'https://github.com/netology-git/demo.git' is highlighted with a red box and a red circle with the number 2. The 'Use SSH' option is also visible.

netology-git / demo

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. Edit

Manage topics

3 commits 1 branch 0 releases 0 contributors

Branch: master New pull request

Create new file Upload files Find file

Clone or download

Clone with HTTPS Use SSH

Use Git or checkout with SVN using the web URL.

https://github.com/netology-git/demo.git

## git remote

Команда `git remote` позволяет нам управлять удалёнными репозиториями (добавлять, удалять, просматривать).

Например, мы можем в только что клонированном репозитории посмотреть remote:

```
$ git remote -v  
origin https://github.com/netology-git/demo.git (fetch)  
origin https://github.com/netology-git/demo.git (push)
```

Общепринято, что первый удалённый репозиторий называют `origin`.



## РАБОТА С РЕПОЗИТОРИЕМ

Дальше мы работаем с нашим репозиторием так же, как с обычным локальным ровно до тех пор, пока не захотим отправить изменения, которые мы внесли, на удалённый репозиторий (об этом поговорим чуть позже).

## ДОБАВЛЕНИЕ `remote`

Рассмотрим второй вариант, когда у нас уже есть локальный репозиторий с проектом, и мы хотим к нему подключить удалённый.

Для этого используем также команду `git remote`:

```
$ git remote add origin https://github.com/netology-git/demo.git
```

```
$ git remote -v
```

```
origin https://github.com/netology-git/demo.git (fetch)
```

```
origin https://github.com/netology-git/demo.git (push)
```

# ОТПРАВКА ИЗМЕНЕНИЙ

После того, как мы поработали локально, необходимо отправить наши изменения в удалённый репозиторий. Для этого используется команда `git push`, для первой отправки: `git push -u origin master`.

```
$ git push -u origin master
Username for 'https://github.com': netology-git
Enumerating objects: 10, done
Counting objects: 100% (10/10), done.
Compressing objects: 100% (7/7) done.
Writing objects: 100% (10/10), 1.13 KiB | 580.00 KiB/s, done.
Total 10 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/netology-git/demo/pull/new/master
remote:
To https://github.com/netology-git/demo.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

# ОТПРАВКА ИЗМЕНЕНИЙ

При отправке изменений вас попросят ввести логин и пароль.

При последующих отправках достаточно использовать команду `git push`.

*Важно: мы с вами ещё не проходили ветки и конфликты, поэтому не вносите параллельные изменения с разных компьютеров. До следующей лекции делайте одно клонирование с одного репозитория и все изменения отправляйте с одного локального репозитория.*

# ОТПРАВКА ИЗМЕНЕНИЙ

Посмотреть, что изменения отправились, можно на веб-страничке проекта:

The screenshot shows the GitHub interface for the repository 'netology-git / demo'. At the top, there are buttons for 'Watch', 'Star', and 'Fork', each with a count of 0. Below this is a navigation bar with links for 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Insights', and 'Settings'. The main content area starts with the text 'No description, website, or topics provided.' and an 'Edit' button. Below this is a 'Manage topics' link. A summary bar shows '3 commits', '1 branch', '0 releases', and '0 contributors'. Below the summary bar are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and a green 'Clone or download' button. The commit history section shows the latest commit by 'Vasya Pupkin' with the message 'Revert "Roboto Font"' and hash 'ffb1e3', dated '3 days ago'. Below the commit list are three files: 'css', 'js', and 'README.md', each with the description 'Первоначальная версия' and a date of '3 days ago'.

netology-git / demo

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. Edit

Manage topics

3 commits 1 branch 0 releases 0 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

Vasya Pupkin Revert "Roboto Font" Latest commit ffb1e3 3 days ago

|           |                       |            |
|-----------|-----------------------|------------|
| css       | Первоначальная версия | 3 days ago |
| js        | Первоначальная версия | 3 days ago |
| README.md | Первоначальная версия | 3 days ago |



# MARKDOWN





# MARKDOWN

Облегчённый язык разметки, который позволяет форматировать текст и затем преобразовывать его в другие форматы, например, HTML.

## README .md

Чаще всего, документ, оформленный с использованием языка Markdown, хранится в текстовом файле с расширением `.md`.

На сервисе GitHub принято описание, содержащееся в специальном файле с именем `README .md` выводить в качестве описания проекта.

# ЗАДАЧА: ОПИСАНИЕ ПРОЕКТА

Попробуем подготовить описание проекта, аналогичное скриншоту.

## Twitter Bootstrap



**Twitter Bootstrap** - популярный набор компонентов для фронтенд-разработки.

Построен на базе следующих технологий:

- HTML;
- CSS;
- JavaScript.

## Начало работы

Есть несколько вариантов подключения:

1. Пакетный менеджер `npm`;
2. CDN.

## Установка при помощи npm

Откройте консоль и выполните следующую команду: `npm install bootstrap`

## Установка помощи CDN

Если вы хотите подключить только CSS (без JavaScript-плагинов), добавьте в ваши html-файлы следующий код:

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
```

Для JavaScript необходимо добавить следующий код:

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965D" ></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js" integrity="sha384-q8i/X+965D" ></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js" integrity="sha384-q8i/X+965D" ></script>
```

## Использование

Все возможности по использованию описаны в [официальной документации](#).

# ЗАГОЛОВКИ

Заголовки – так же, как и в HTML поддерживаются несколько уровней заголовков: от 1 до 6

- 1 **# Заголовок первого уровня**
- 2 **## Заголовок второго уровня**
- 3 **### Заголовок третьего уровня**
- 4 **#### Заголовок четвертого уровня**
- 5 **##### Заголовок пятого уровня**
- 6 **##### Заголовок шестого уровня**

Потренироваться онлайн вы можете с помощью сервиса <https://hackmd.io>.

# ЗАГОЛОВКИ

- 1 # Twitter Bootstrap
- 2 ## Начало работы
- 3 ### Установка при помощи npm
- 4 ### Установка при помощи CDN
- 5 ## Использование

## Twitter Bootstrap

---

### Начало работы

---

### Установка при помощи npm

### Установка помощи CDN

### Использование

---

# ТЕКСТ: ОБЫЧНЫЙ И СТИЛИЗОВАННЫЙ

Обычный текст никакой специальной разметкой не оформляется.

Жирный, наклонный и перечёркнутый текст:

- 1 **\*\*Жирный текст\*\***
- 2 *\*Наклонный текст\**
- 3 ~~Перечеркнутый текст~~

**Жирный текст**

*Наклонный текст*

~~Перечеркнутый текст~~

# ТЕКСТ

## # Twitter Bootstrap

**\*\*Twitter Bootstrap\*\*** - популярный набор компонентов для фронтенд-разработки.  
Построен на базе следующих технологий:

### ## Начало работы

Есть несколько вариантов подключения:

#### ### Установка при помощи npm

Откройте консоль и выполните следующую команду: `npm install bootstrap`

#### ### Установка при помощи CDN

Если вы хотите подключить только CSS (без JavaScript-плагинов),  
добавьте в ваши html-файлы следующий код:

Для JavaScript необходимо добавить следующий код:

### ## Использование

Все возможности по использованию описаны в официальной документации.

---

# РЕЗУЛЬТАТ

## Twitter Bootstrap

---

**Twitter Bootstrap** - популярный набор компонентов для фронтенд-разработки. Построен на базе следующих технологий:

### Начало работы

---

Есть несколько вариантов подключения:

#### Установка при помощи npm

Откройте консоль и выполните следующую команду: `npm install bootstrap`

#### Установка при помощи CDN

Если вы хотите подключить только CSS (без JavaScript-плагинов), добавьте в ваши html-файлы следующий код:

Для JavaScript необходимо добавить следующий код:

### Использование

---

Все возможности по использованию описаны в официальной документации.

---



# СПИСКИ

Списки оформляются либо \* либо 1. :

- |   |                                  |
|---|----------------------------------|
| 1 | * Элемент списка                 |
| 2 | * Элемент списка                 |
| 3 | * Вложенный элемент списка       |
| 4 | * Вложенный элемент списка       |
| 5 | 1. Элемент упорядоченного списка |
| 6 | 1. Элемент упорядоченного списка |
| 7 | 1. Вложенный элемент списка      |
| 8 | 1. Вложенный элемент списка      |

- Элемент списка
- Элемент списка
  - Вложенный элемент списка
  - Вложенный элемент списка
- 1. Элемент упорядоченного списка
- 2. Элемент упорядоченного списка
  - 1. Вложенный элемент списка
  - 2. Вложенный элемент списка

# СПИСКИ

1 Построен на базе следующих технологий:

2 \* HTML

3 \* CSS

4 \* JavaScript

5

6 ## Начало работы

7 Есть несколько вариантов подключения:

8 1. Пакетный менеджер npm;

9 1. CDN.

Построен на базе следующих технологий:

- HTML;
- CSS;
- JavaScript.

## Начало работы

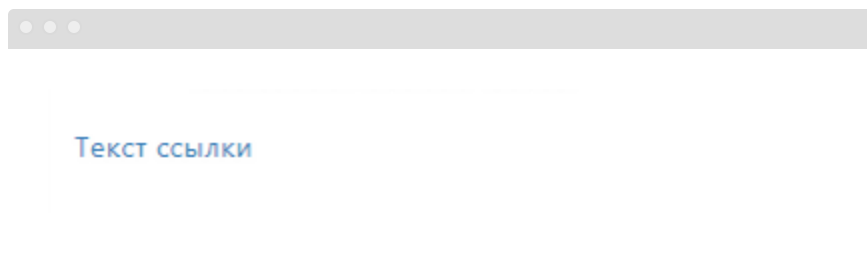
Есть несколько вариантов подключения:

1. Пакетный менеджер npm;
2. CDN.

# ГИПЕРССЫЛКИ

Гиперссылки оформляются в формате [Текст ссылки](url-адрес):

[Текст ссылки](http://localhost)



# ГИПЕРССЫЛКИ

- 1 **## Начало работы**
- 2 Есть несколько вариантов подключения:
- 3 1. Пакетный менеджер `[npm](https://npmjs.com)`;
- 4 1. CDN.

## Начало работы

Есть несколько вариантов подключения:

1. Пакетный менеджер `npm`;
2. CDN.

---

# ИЗОБРАЖЕНИЯ

Изображения оформляются так же, как и гиперссылки, но перед `[]` ставится `!`, т.е. `![логотип](url-изображения)`:

```
1 # Twitter Bootstrap
2
3 ![Bootstrap logo](https://i.imgur.com/qhtywl2.png)
4 **Twitter Bootstrap - популярный набор компонентов для фронтенд-разработки.
```

## Twitter Bootstrap



**Twitter Bootstrap** - популярный набор компонентов для фронтенд-разработки.

---

# КОД

Есть два варианта оформления кода:

- Inline: код заключается в backtick'и: ``строка кода``
- Block: код с подсветкой синтаксиса:

```
```javascript
console.log("");
```
```

```
1  ### Установка при помощи npm
2  Откройте консоль и выполните следующую команду: `npm install bootstrap`
3
4  ### Установка при помощи CDN
5  Если вы хотите подключить только CSS (без JavaScript-плагинов),
6  добавьте в ваши html-файлы следующий код:
7  ```html
8  <link rel="stylesheet"
9      href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
10     integrity="sha284-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkF0JwJ8ERdknLPM0"
11     crossorigin="anonymous">
12  ...
```

## Установка при помощи npm

Откройте консоль и выполните следующую команду: `npm install bootstrap`

## Установка помощи CDN

Если вы хотите подключить только CSS (без JavaScript-плагинов), добавьте в ваши html-файлы следующий код:

```
<link rel="stylesheet" href="https://stackpath.bootst
```



# МНОЖЕСТВО РЕАЛИЗАЦИЙ

Разные сервисы и библиотеки поддерживают различное подмножество языка Markdown (кто-то добавляет новые возможности, например, возможность описывать таблицы).

Поэтому внимательно читайте документацию о поддержке конкретных возможностей.



# МНОЖЕСТВО РЕАЛИЗАЦИЙ: GITHUB

Посмотреть все возможности оформления, которые предлагает сервис GitHub можно по ссылке: <https://guides.github.com/features/mastering-markdown/>

Среди них:

- Списки задач;
- Указание пользователей;
- Таблицы;
- Ссылки на issue;
- И т.д.



# ИТОГИ

Сегодня мы с вами разобрали достаточно много вопросов:

1. Предназначение системы контроля версий Git;
2. Работа с локальным репозиторием;
3. Привязка удалённого репозитория и GitHub;
4. Язык разметки Markdown.



**Задавайте вопросы и напишите отзыв о лекции!**

**ИЛЬНАЗ ГИЛЬЯЗОВ**

 [coursar@gmail.com](mailto:coursar@gmail.com)