## Медиазапросы и GIT

## **@-правило** — это инструкция CSS. Каждая инструкция начинается с @ и содержит одно из ключевых слов. Они подсказывают парсеру, как CSS должен себя вести.

## @import, @-rules, @font-face, @media

**Типы медиа бывают следующими:**

|  |  |
| --- | --- |
| all | Для всех девайсов |
| print | Для принтеров |
| screen | Для компьютеров, планшетов, телефонов и т.д. |
| speech | Для программ чтения с экрана, которые «читают» страницу вслух |

В современных браузерах можно также опускать медиатип (в таком случае будет применён тип all).

Вы можете комбинировать разные параметры, например: @media (max-width: 600px) and (orientation: landscape) { ... }

**Медиа-запросы можно добавлять всеми теми же способами, что и обычные стили.** Например, чтобы вставить медиазапрос непосредственно в *HTML,* вы можете указать, что конкретный файл со стилями содержит стили только при выполнении условий медиа-запроса:

<link rel="stylesheet" media="screen and (max-width: 600px)" href="special-style.css">

Для комбинирования разных условий вы можете использовать:

* Оператор **and** как показано выше в примере: он требует, чтобы выполнялись все связанные этим оператором условия.
* Оператор **not**. Например, @media not screen означает для всех типов медиа, кроме screen.
* Оператор **only**. Этот оператор позволяет скрыть стили от старых браузеров, которые понимают только CSS2. Например, @media only screen.

Для управления разметкой в мобильных браузерах используется специальный meta-тег **viewport**.

<meta name="viewport" content="width=device-width, initial-scale=1.0">

где

**width=device-width** даёт команду браузеру подстраивать ширину страницы под ширину экрана устройства,

а **initial-scale=1.0** — это коэффициент масштабирования.

Медиа-запросы могут использоваться для проверки информации разного характера, что позволяет точнее определить вид устройства. Эти параметры включают в себя:

* ширина и высота*viewport*;
* ширина и высота устройства;
* ориентация (расположен телефон/планшет в горизонтальном или вертикальном положении);
* разрешение.

**@font-face**

Данное правило позволяет загружать на веб-страницу специальные шрифты. Правило указывает браузеру скачать шрифт из указанного источника и затем отобразить его, как определено в *CSS*.

**Пример**:

**@font-face** {

font-family: 'My Web Font';

src: url(…);

}

body {

**font-family**: 'My Web Font', 'Open Sans', **cursive**;

}

## @import

Это правило обычно указывается в начале файла и указывает на включение внешнего CSS-файла. При этом содержимое внешнего файла вставляется на строку, где указано правил.

@import 'https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css';

## @media

Это правило содержит условия для определения стилей для разных экранов. Условия могут содержать размеры экранов, что очень полезно для адаптивной верстки. С этим правилом вы обязательно столкнетесь при разработке адаптивных макетов.

**Пример:**

**@media** not|only mediatype and (media feature) {

CSS-Code;

}

**Содержимое директории .git**

Рассмотрим, что содержится в папках и файлах:

* + В **файле config** находятся настройки данного репозитория. Его содержимое представлено в текстовом формате.
  + **Файл HEAD** указывает на текущую ветку.
  + В **файле index** хранится содержимое индекса.
  + В **директории objects** находится, собственно, база данных объектов *Git*. Если открыть каталог *objects*, то в нём будут находиться каталоги, имена которых представлены двумя шестнадцатеричными числами, внутри которых будут файлы, имена которых представлены 38 шестнадцатеричными числами. Вместе имя каталога и файл образуют 40-разрядный хэш, взятый от имени файла и его содержимого.
  + В **директории refs** находятся ссылки на объекты коммитов в этой базе (ветки).
  + **Директория logs** хранит логи коммитов.
  + В **директории info** расположен файл с глобальными настройкам игнорирования файлов. Он позволяет исключить файлы, которые вы не хотите помещать в .gitignore. Позднее мы остановимся на назначении файла .gitignore.
  + В **директории hooks** располагаются клиентские и серверные триггеры. Желающие прочитать про них могут обратиться к руководству по *Git.*

**git init  
Эта команда создаст .git репозиторий в Вашем проекте. Репозиторий или “repo” это коллекция всех изменений, которые были совершены на протяжении всего времени после инициализации репозитория. Это первое что нужно сделать для нового проекта.**

**Подключить комп к гитхабу -**

**git remote add origin https://github.com/Egnat/hello-world.git**

### Копирует инфу с гитхаба (проект) - git pull origin master или название др. ветки

**Команда « touch » является стандартной командой, которая используется в операционной системе Linux для создания и изменения временных меток файла. echo > — непосредственная команда, создающая файл.**

## Создать папку — mkdir или md

Создаём пустую папку code внутри папки html :

Создаём несколько пустых вложенных папок — для этого записываем их через косую черту.

## Удалить папку — rmdir или rd

При этом удалить можно только пустую папку. Если попытаться удалить папку, в которой что-то есть, увидим ошибку.

## Показать дерево папок — tree

Если вы хотите посмотреть содержимое всего диска, введите tree в корне нужного диска. Получится красивая анимация, а если файлов много, то ещё и немного медитативная.

## Удаление файла — del или erase

Команда для удаления одного или нескольких файлов.

## Переименование файла — ren или rename

Последовательно вводим ren, старое и новое имя файла.

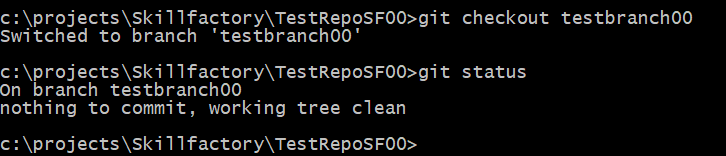
**Чтобы приступить к работе над изменениями для Чёрной пятницы, создадим специальную ветку:**

**git branch black-Friday**

**Создадим новую ветку в Git с помощью известной нам команды git branch, после которой следует имя создаваемой ветки. Назовем её следующим образом:**

**git branch testbranch00**

1. Для проверки состояния выполните команды git status и git log. Последняя показывает историю изменений: перечисляет коммиты, сделанные в репозитории, в обратном хронологическом порядке. Последние коммиты находятся вверху.
2. Команду git branch --list. Она выведет список веток, которые существуют в вашем локальном репозитории. Текущая ветка *master* будет выделена зелёным цветом.
3. Команду git branch --list -v . Вы увидите информацию о последних коммитах каждой из веток. Так как коммитов в ветку *testbranch00* ещё не было, то она указывает на коммит в *master*, от которого была создана.
4. Команда git checkout переключит нас на созданную ветку. Выполняя команду git checkout, укажите после неё имя ветки, на которую вы хотите переключиться.
5. Например, git checkout testbranch00. Вы получите сообщение о переключении на ветку *testbranch00*.
6. Выполните команду git status. Вы увидите сообщение о том, что вы находитесь на ветке *testbranch00*.



**Чтобы удалить ветку, достаточно выполнить команду:**

git branch -D black-Friday

**объединение веток** - git merge

**создать и сразу перейти на ветку –**

git checkout –b название ветки

запись изменений в файле

git commit –m “навание папки название ветки запись изменений”

копирует в комп с гитхаба - git clone

<C:\\Users\\SkillF> git clone ссылка моего реп-я https://github.com/Egnat/hello-world

1. **Если введем git log --help - он предоставит нам документацию по какой-то определенной команде (в данном случае это - log)**

## Команды одной строкой

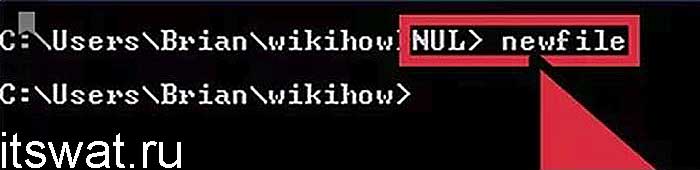
Очистить консоль — **cls**

Информация о системе — **systeminfo**

Информация о сетевых настройках — **ipconfig**

Список запущенных процессов — **tasklist**

Как создать новый текстовый файл через cmd? Для этого вводится команда NUL> с именем файла. Пример на изображении:



|  |  |
| --- | --- |
| **Название команды** | **Действие** |
| pwd | показать текущий каталог |
| ls | показать файлы в данной папке, кроме скрытых |
| ls -f | показать файлы в данной папке, включая скрытые |
| cd c: | перейти к диску (в примере диск *C*) |
| cd app | перейти в конкретный каталог (в примере папка *app*) |
| cd - | вернуться назад |
| cd .. | подняться на один уровень вверх |
| cd ../.. | подняться на два уровня вверх |
| mkdir | создать папку |