

## ЯЗЫК РАЗМЕТКИ

**Язык разметки** — это специальный компьютерный язык для описания оформления и строения документа.

**WYSIWYG ("What You See Is What You Get")** — свойство прикладных программ или веб-интерфейсов, в которых содержание отображается в процессе редактирования и выглядит максимально похожим на конечный результат.

Примеры WYSIWYG-сервисов:

- MS Word, который позволяет создавать текстовые документы в графическом интерфейсе.
- Любые конструкторы сайтов. Они позволяют создавать структуру сайтов, даже если вы не владеете HTML.

Процесс создания страниц из составных элементов называется **компьютерной вёрсткой**. При этом могут использоваться как WYSIWYG-редакторы, так и редакторы, требующие знания языков разметки.

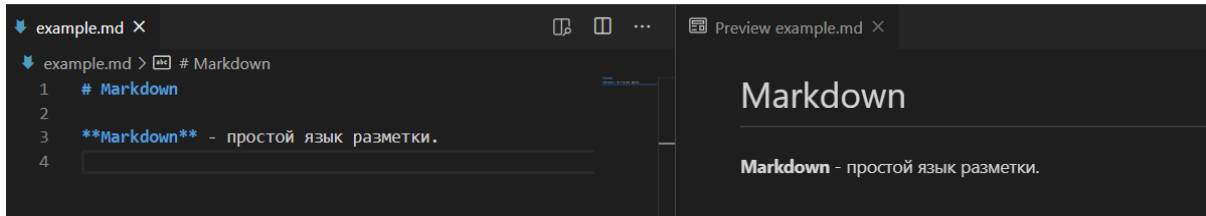
Если разметки очень много, как, например, при разработке сайта, может быть выделена должность отдельного специалиста — **верстальщика**.

## ЯЗЫК РАЗМЕТКИ MARKDOWN

**Markdown** — простой и понятный язык разметки для оформления документации, который является упрощённой версией HTML. Благодаря своей простоте он используется во множестве сервисов — как специальных (для разработчиков), так и направленных на пользователей.

Файлы с разметкой *Markdown* имеют расширение **.md**.

Пример ведения md-разметки в VS Code:



## СИНТАКСИС MARKDOWN: ШРИФТ

Наклонное и **полужирное** начертание в *Markdown* задаются при помощи символов `*` и `_`:

- один символ — для наклонного текста (`_italic_`, `*italic*`);
- два символа — для полужирного текста (`_strong_`, `**strong**`);
- три — для наклонного и жирного одновременно (`***жирный и наклонный***`).

## СИНТАКСИС MARKDOWN: ЗАГОЛОВКИ

Заголовки отмечаются символом `#` (от одного до шести штук) в начале строки. Размер шрифта каждого следующего уровня меньше предыдущего:

```
# Заголовок h1
## Заголовок h2
### Заголовок h3
#### Заголовок h4
##### Заголовок h5
##### Заголовок h6
```

Для центрирования текста в *Markdown* можно воспользоваться тегом `<center>`:

```
# <center> Заголовок h1 </center>
```

Полезно выделять структуру текста с помощью горизонтальных линий. Её можно добавить с помощью символов "---":

```
## Заголовок h2
---
## Заголовок h2
```

## СИНТАКСИС MARKDOWN: СПИСКИ

Для разметки нумерованных списков можно использовать символы \*, - или + — результат будет один:

```
+ элемент 1
- элемент 2
* элемент ...
```

Вложенные пункты создаются двумя или более пробелами перед маркером пункта:

```
* элемент
  * вложенный элемент 2.1
  * вложенный элемент 2.2
```

Для разметки нумерованных списков ставится положительное число с точкой:

```
1. элемент 1
2. элемент 2
  2.1. элемент 3
  2.2. элемент 3
3. элемент 4
```

Нумерация списка начинается с того числа, которое стоит первым на уровне. В отличие от нумерованных списков, в нумерованных перед элементами вложенного списка ставится три пробела.

## СИНТАКСИС MARKDOWN: ССЫЛКИ И ИЗОБРАЖЕНИЯ

Ссылки создаются комбинацией квадратных и круглых скобок: в квадратных скобках указывается отображаемый текст ссылки, а в круглых — URL-адрес или

путь до файла, на который вы ссылаетесь. Ссылки могут быть обычными (без подсказок) или с подсказками, которые всплывают при наведении курсором на ссылку:

- без подсказки — `[текст ссылки](http://example.com/link);`
- с подсказкой — `[текст ссылки](http://example.com/link "Подсказка").`

Для отображения изображений перед квадратными скобками ставится восклицательный знак:

```

```

Другой вариант вставки изображений — использовать тег `<img>` со специальными атрибутами:

```
<img src=https://i.imgur.com/3uj9teq.png width=500px height=30%>
```

## СИНТАКСИС MARKDOWN: ПРОГРАММНЫЙ КОД И ЦИТАТЫ

Для выделения программного кода используется обратный апостроф:

- одинарный парный — для вставки строки кода в текст;
- двойной парный — для вставки небольшого участка кода, содержащего одинарный апостроф, в текст;
- тройной парный — для вставки блока программного кода.

```
`print('Hello world!')`
```

При использовании тройного апострофа можно указать язык программирования, чтобы автоматически подсветить участки кода, как в редакторе:

```
```python  
lst = [10, 34, 21, 21, 3]  
summa = sum(lst)  
```
```

Для оформления цитат используется знак «больше» (`>`):

```
> Цитируемый текст
```

## СИНТАКСИС MARKDOWN: ФОРМУЛЫ

Формулы в *Markdown* отмечаются символом `$`. Если обрмить формулу с обеих сторон одним символом `$`, то её можно встроить в текст, а если двумя — формула автоматически центрируется.

### Например:

Пусть задано выражение:

`$$a = b + c, $$`,

где `$a=0$`

Для греческих букв есть специальные команды, которые выражаются через символ `\`.

### Например:

`$$\alpha$` —  $\alpha$ ;

`$$\gamma$` —  $\gamma$ ;

`$$\sigma$` —  $\sigma$

Степени и индексы набираются с помощью символов `^` и `_` соответственно. Если символов, которые нужно поместить в степень или индекс, несколько, то они выделяются фигурными скобками.

### Например:

`$a^2$` —  $a^2$ ;

`$b_{ij}$` —  $b_{ij}$ ;

`$w^{ij}_n$` —  $w_n^{ij}$

Для того чтобы создать «двухэтажную» дробь, можно воспользоваться оператором `\frac` с двумя параметрами, которые передаются в фигурных скобках (числитель и знаменатель).

### Например:

`$$\frac{1+x}{n}$` —  $\frac{1+x}{n}$

## СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ

**Система управления (контроля) версиями (англ. Version Control System)** — это программное обеспечение, которое позволяет управлять состояниями изменяющейся информации. Благодаря таким системам несколько людей могут работать с файлами, сохранять их версии, перемещаться между версиями и откатывать изменения.

**Репозиторий** — это хранилище каких-либо данных. В случае с системой контроля версий, репозиторий — это хранилище, содержащее программный код и другие атрибуты (например, данные) IT-проекта.

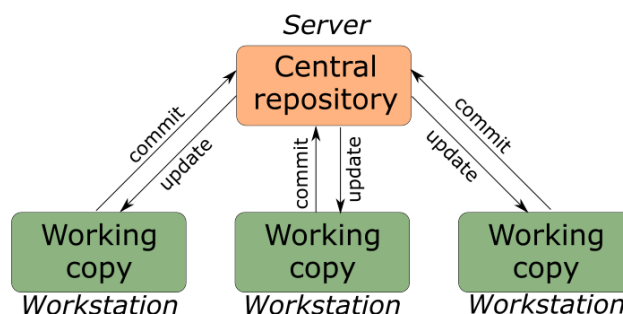
## ТИПЫ СИСТЕМ КОНТРОЛЯ ВЕРСИЙ

### → Локальные

Локальная система хранит все файлы на одном конкретном устройстве (например, на ПК) и контролирует их изменения на нём.

### → Централизованные

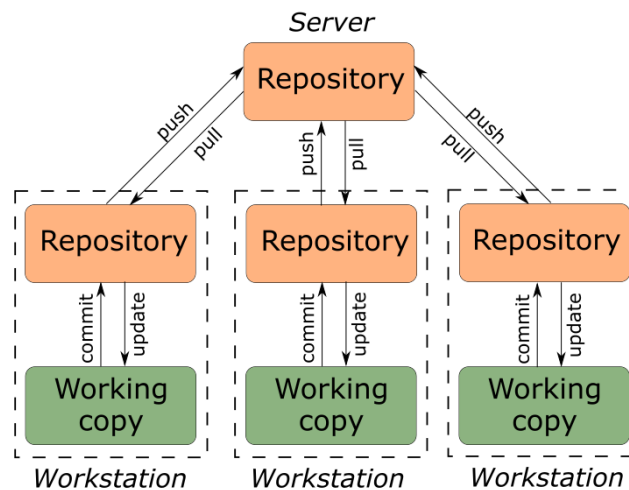
В централизованной системе весь проект существует в единственном экземпляре и хранится на главном удалённом сервере. Доступ к серверу осуществляется через специальное клиентское приложение, в котором разработчики напрямую обновляют состояние проекта.



### → Распределённые

В распределённой системе контроля версий есть один удалённый репозиторий (как правило, в [облаке](#)) и множество локальных. Удалённый репозиторий можно считать центральным, но только условно.

При этом локальные и удалённые репозитории синхронизированы, то есть разработчики работают с локальной копией общего репозитория, причём все копии проекта, включая удалённую, являются равнозначными. После внесения изменений в локальную копию проекта они отправляются на удалённый репозиторий.



## СИСТЕМА КОНТРОЛЯ ВЕРСИЙ GIT

[Git](#) — это распределённая система управления версиями с предусмотренным механизмом ветвления.

## GITHUB — ХОСТИНГ IT-ПРОЕКТОВ

[GitHub](#) — наиболее популярный хостинг для IT-проектов. На нём хранятся миллионы удалённых репозиториях как небольших команд, так и крупных корпораций.

Отличительная особенность *GitHub* — лёгкое создание форков.

**Форки** — это собственные проекты, созданные на основе сторонних проектов.

Среди возможностей этого IT-хостинга, кроме системы контроля версий, есть ведение документации (*wiki*) проекта, трекинг задач (*issues*), приём пожертвований.

## GIT: ОСНОВНЫЕ ТЕРМИНЫ

Основная терминология системы Git:

**Репозиторий** — папка проекта, отслеживаемого Git, содержащая историю изменений. Все файлы истории хранятся в специальной папке **.git/** внутри папки проекта.

**Рабочая директория** — файловая система проекта (те файлы, с которыми вы работаете).

**Индекс** — файл, в котором содержатся изменения, подготовленные для добавления в коммит. Вы можете добавлять и убирать файлы из индекса.

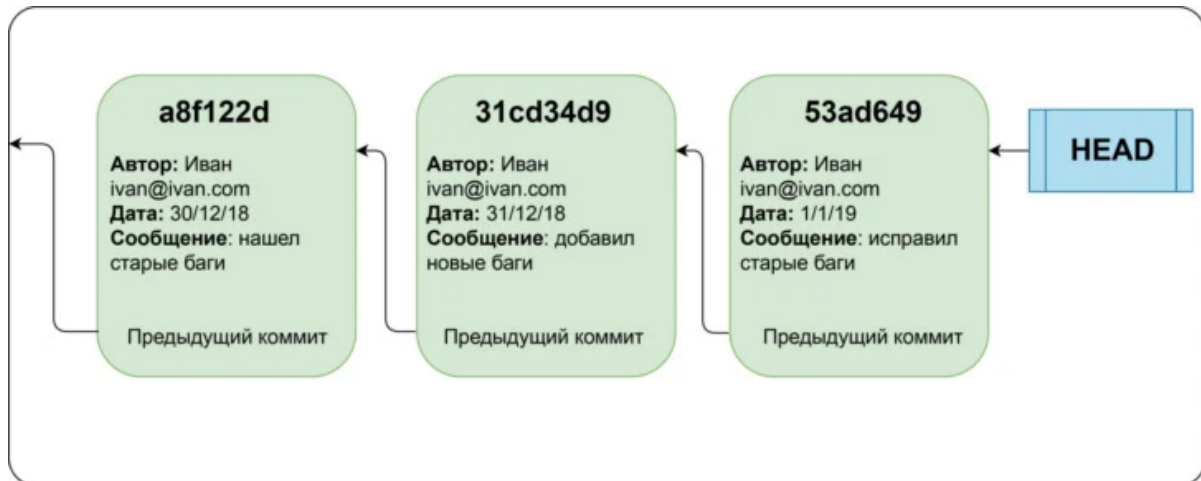
**Коммит** — это операция сохранения набора изменений, сделанного в рабочей директории с момента предыдущего коммита. Коммит неизменен, его нельзя отредактировать — можно только отменить. Коммит хранит изменённые файлы, имя автора коммита и время, в которое был сделан коммит. Кроме того, каждый коммит имеет уникальный идентификатор (хеш), который позволяет в любое время к нему откатиться.

**Указатель HEAD** — это указатель (то есть ссылка на один из коммитов), главное назначение которого — определять, в каком состоянии находится рабочая директория. На какой коммит указывает *HEAD*, в таком состоянии файлы и находятся в рабочей области.

**Ветка** — это последовательность коммитов. Технически же, ветка — это ссылка на последний коммит в этой ветке. Преимущество веток — в их

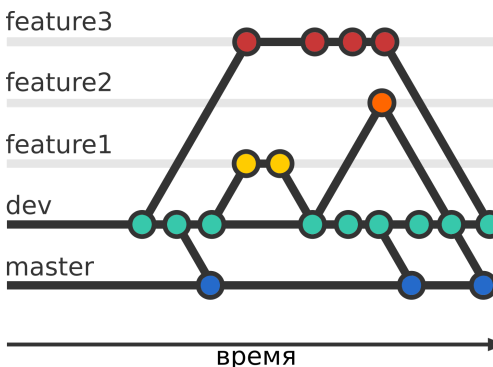
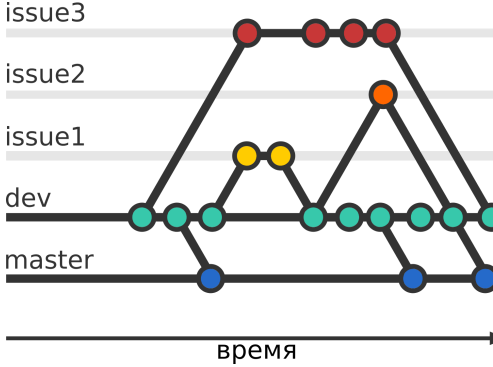
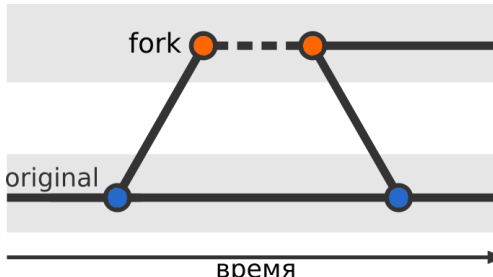
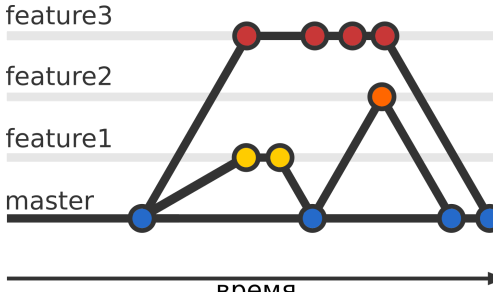


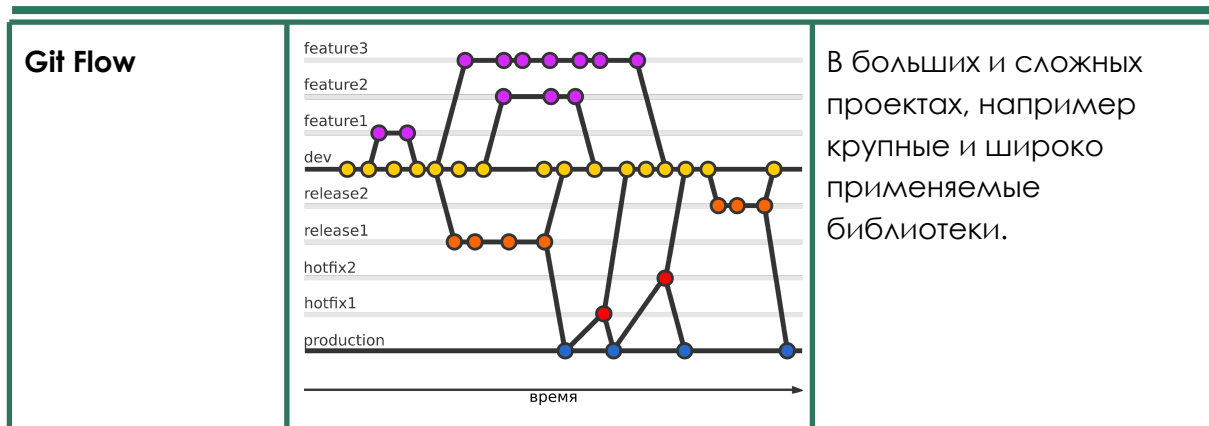
независимости: вы можете вносить изменения в файлы на одной ветке, например, пробовать новую функцию, и они никак не скажутся на файлах в другой ветке. Изначально в репозитории одна основная ветка (по умолчанию master), но пользователь *Git* может создавать их самостоятельно.



## МЕТОДОЛОГИИ ВЕТВЛЕНИЯ

| Методология                      | Схема  | Где используется  |
|----------------------------------|--|---|
| <b>Central Workflow</b>          | <p>The diagram shows a single horizontal line representing the 'master' branch. A series of blue dots connected by a line represent commits. An arrow below the line points to the right and is labeled 'время' (time).</p>  | Подходит для одиночного проекта. Не подходит для командной разработки.                                      |
| <b>Developer Branch Workflow</b> | <p>The diagram shows three horizontal lines representing different branches: 'anton' (top), 'master' (middle), and 'boris' (bottom). Yellow dots on the 'anton' branch and red dots on the 'boris' branch represent commits. Blue dots on the 'master' branch represent production-ready commits. Arrows show frequent branching and merging between the branches. An arrow below the lines points to the right and is labeled 'время' (time).</p> | Больше подойдёт для небольшого проекта с ограниченным количеством требований и разработчиков (меньше пяти). |

|                                |  |   |
|--------------------------------|--|---|
| <b>Feature Branch Workflow</b> |  <p>The diagram illustrates the Feature Branch Workflow. It shows a horizontal timeline labeled 'время' (time) at the bottom. The vertical axis lists branches: master, dev, feature1, feature2, and feature3. The master branch (blue dots) has two commits. The dev branch (green dots) has several commits, including a merge from feature1. Feature1 (yellow dots) branches off from dev, has two commits, and merges back into dev. Feature2 (orange dots) branches off from dev, has one commit, and merges back into dev. Feature3 (red dots) branches off from dev, has three commits, and merges back into dev. The master branch is updated with a new commit after the final merge from dev.</p> | <p>Подходит командам, которые используют специальные методы управления проектами, например <a href="#">Agile-методологию</a>.</p> |
| <b>Issue Branch Workflow</b>   |  <p>The diagram illustrates the Issue Branch Workflow. It shows a horizontal timeline labeled 'время' (time) at the bottom. The vertical axis lists branches: master, dev, issue1, issue2, and issue3. The master branch (blue dots) has two commits. The dev branch (green dots) has several commits, including a merge from issue1. Issue1 (yellow dots) branches off from dev, has two commits, and merges back into dev. Issue2 (orange dots) branches off from dev, has one commit, and merges back into dev. Issue3 (red dots) branches off from dev, has three commits, and merges back into dev. The master branch is updated with a new commit after the final merge from dev.</p>                | <p>Подходит командам, работающим по специальным методологиям управления проектом.</p>   |
| <b>Forking Workflow</b>        |  <p>The diagram illustrates the Forking Workflow. It shows a horizontal timeline labeled 'время' (time) at the bottom. The vertical axis lists branches: original and fork. The original branch (blue dots) has two commits. The fork branch (orange dots) branches off from the original, has two commits (indicated by a dashed line), and merges back into the original. The original branch is then updated with a new commit.</p>  | <p>Чаще всего используется в проектах с открытым исходным кодом и публичными репозиториями.</p>                                   |
| <b>GitHub Flow</b>             |  <p>The diagram illustrates the GitHub Flow. It shows a horizontal timeline labeled 'время' (time) at the bottom. The vertical axis lists branches: master, feature1, feature2, and feature3. The master branch (blue dots) has two commits. Feature1 (yellow dots) branches off from master, has two commits, and merges back into master. Feature2 (orange dots) branches off from master, has one commit, and merges back into master. Feature3 (red dots) branches off from master, has three commits, and merges back into master. The master branch is updated with a new commit after the final merge from feature3.</p>   | <p>Подходит командам, работающий по гибким методологиям управления проектами.</p>   |



## КУЛЬТУРА КОММИТОВ

- Текст коммита формируется из трёх частей:
  - ◆ действие (добавление, исправление, рефакторинг и т. д.);
  - ◆ сущность (документация, новая модель, главная страница и т. д.);
  - ◆ подробности (задача №23, несуществующий пользователь, зависимости и т. д.) — необязательное поле.
- Полнота — не многословие. Старайтесь давать достаточную информацию об изменениях, но избегайте излишних подробностей.
- Используйте в коммитах английский язык. В русскоязычных командах допускаются коммиты на русском языке, но это не лучшая практика, так как ограничивает аудиторию проекта.
- Найдите свой стиль. Необязательно изобретать велосипед. Ознакомьтесь с различными практиками, соблюдайте требования команды.

## ФОРК

**Форк (fork)** — собственный проект, основанный на другом проекте, но при этом сохраняющий связь с ним.

### Важные возможности форка:

- сохраняет связь с проектом-родителем, по которой он может получить изменения из проекта-родителя;
- сохраняет связь с проектом-родителем, по которой он может передать изменения в проект-родитель. Этот принцип используется в методологии ветвления *Forking Workflow*.

На *GitHub* форк создаётся с помощью кнопки **Fork**, которая появляется в интерфейсе при просмотре чужих проектов.

