




Frontender[1.0] GIT, NodeJS, NPM.

Работа с GIT через консоль с помощью команд. Блок 1

🔗 YouTube	https://youtu.be/ftAQBip0Q4M
🔗 Telegram	https://t.me/Dmitry_Kolotilshikov
# Номер урока	80

 Ссылка на официальный сайт NodeJS

<https://nodejs.org/en>

Node.js® is an open-source, cross-platform JavaScript runtime environment.

Download for Windows (x64)

18.16.0 LTS
Recommended For Most Users

20.2.0 Current
Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

For information about supported releases, see the [release schedule](#).

[Long Term Support](#)

Ссылка на локальный GIT client:

<https://git-scm.com/downloads>



Node.js - это среда выполнения **JavaScript**, построенная на движке **JavaScript V8** от **Google**, который также используется в браузере **Google Chrome**. **Node.js** позволяет запускать JavaScript-код на сервере, вне контекста браузера.

Основные особенности и возможности Node.js:

1. **Выполнение на стороне сервера:** Node.js позволяет разрабатывать серверные приложения на JavaScript. Он предоставляет среду, которая позволяет запускать JavaScript-код непосредственно на сервере, в отличие от традиционного JavaScript, который выполняется в браузере.
2. **Асинхронное и событийно-ориентированное программирование:** Одной из ключевых особенностей Node.js является его асинхронная и событийно-ориентированная модель программирования. Это означает, что Node.js может обрабатывать множество запросов одновременно без блокировки потока выполнения. Это особенно полезно для создания масштабируемых и высокопроизводительных приложений, таких как серверы веб-приложений или API.
3. **Модульная система:** Node.js предоставляет модульную систему, которая позволяет разделять код на отдельные модули. Модули могут импортироваться и экспортироваться, что способствует организации и повторному использованию кода. Большое сообщество разработчиков Node.js предоставляет множество модулей, которые можно использовать в ваших проектах.
4. **Управление зависимостями:** Node.js имеет встроенный пакетный менеджер `npm` (Node Package Manager), который позволяет управлять зависимостями проекта. С помощью `npm` вы можете легко устанавливать, обновлять и удалять пакеты JavaScript, а также управлять их версиями.
5. **Разнообразие применений:** Node.js широко используется для создания серверных приложений, веб-серверов, API, микросервисов, потоковых серверов, инструментов разработки и многого другого. Благодаря его гибкости и производительности, Node.js стал популярным выбором для разработчиков.

Node.js обладает активным сообществом разработчиков и имеет большую экосистему пакетов, что делает его мощным инструментом для разработки масштабируемых и эффективных серверных приложений на JavaScript.



npm (аббр. **node package manager**) – это стандартный менеджер пакетов, автоматически устанавливающийся вместе с **Node.js**. Он используется для скачивания пакетов из облачного сервера **npm**, либо для загрузки пакетов на эти сервера.

`npx` и `npm` - это два инструмента, связанных с пакетным менеджером `npm` (Node Package Manager), который широко используется в экосистеме Node.js. Вот их отличия:

1. `npm` - это инструмент командной строки, входящий в состав `Node.js`. Он предназначен для установки, управления и обновления пакетов Node.js и их зависимостей. С помощью `npm` вы можете установить пакеты локально в вашем проекте или глобально на вашей системе. Вы также можете использовать `npm` для управления версиями пакетов и запуска скриптов, определенных в файле `package.json`.
2. `npx` - это инструмент командной строки, который был введен в `npm` версии 5.2.0. Он позволяет временно устанавливать и запускать пакеты Node.js без необходимости устанавливать их глобально или локально. Когда вы запускаете `npx`, он автоматически ищет пакет в локальном каталоге `node_modules` и, если он не найден, временно устанавливает его, выполняет команду и затем удаляет установленный пакет. Это позволяет вам использовать инструменты командной строки, предоставляемые пакетами, без необходимости загромождать вашу систему установкой этих пакетов глобально.

Таким образом, основное отличие между `npm` и `npx` заключается в том, что `npm` предназначен для установки и управления пакетами, в то время как `npx` используется для временной установки и запуска пакетов без необходимости предварительной установки.



Ссылка на официальный сайт YARN и гайд по установке:

<https://classic.yarnpkg.com/lang/en/docs/install/>

(

```
npm install --global yarn)
```

Yarn - это пакетный менеджер, альтернативный `npm`, разработанный Facebook, Google, Exponent и Tilde. Он предназначен для управления зависимостями JavaScript и упрощает процесс установки, обновления и удаления пакетов.

Вот некоторые особенности и преимущества Yarn:

- Более быстрая и эффективная установка:** Yarn использует параллельную загрузку пакетов и кэширование, что позволяет установить зависимости быстрее, чем `npm`. Это особенно полезно при работе с большими проектами или при повторной установке пакетов.
- Предсказуемость версий:** Yarn использует файл `yarn.lock`, который явно фиксирует версии пакетов и их зависимостей. Это обеспечивает предсказуемость и консистентность установки пакетов на разных системах, что помогает избежать проблем с несовместимостью версий.
- Улучшенный контроль над зависимостями:** Yarn позволяет явно указывать версии зависимостей с помощью операторов (`^`, `~`, `>=`, и т.д.), что облегчает обновление и управление зависимостями проекта.
- Работа в автономном режиме:** Yarn может работать в автономном режиме, сохраняя все зависимости в локальном кэше. Это полезно, если у вас нет постоянного доступа к Интернету или если вам нужно повторно установить пакеты в другом окружении.
- Поддержка плагинов:** Yarn поддерживает плагины, которые добавляют дополнительные функциональные возможности, такие как управление семантическим версионированием или альтернативные источники пакетов.

Обратите внимание, что как `npm`, так и **Yarn** могут использоваться для управления зависимостями **JavaScript**, и выбор между ними зависит от ваших предпочтений и требований проекта.

Если YARN не работает в VS Code:

if `yarn` not working but `npm` doing ok in your vs code terminal. Then go to search of window and search for window powershell and run it using run as administrator and write code in number **2** and **3** and make the option **yes**

(Перевод):

Для винды: Если `yarn` не работает, но `npm` работает в терминале VS Code, то откройте **window powershell** от имени администратора и вписывайте команды из списка ниже **пункты 2 и 3**.

- Open Powershell and run as administrator**
- Get-ExecutionPolicy**
- Set-ExecutionPolicy Unrestrict**



Git команды (одни из популярных):

git init (Создает новый репозиторий Git или инициализирует существующую папку как репозиторий Git)

git remote add origin <repository_name> (Добавляет удаленный репозиторий с указанным именем и URL)

git remote -v (Показывает список удаленных репозиториях, связанных с текущим репозиторием)

git remote show <https://github.com/YOUR_ACCOUNT/YOUR_REP_NAME.git> (Показывает информацию о указанном удаленном репозитории, включая URL и ветки)

git clone <URL> (Клонирует удаленный репозиторий на локальную машину)

git config --global user.email "user email" (Устанавливают адрес электронной почты, связанный с коммитами)

git config --global user.name "user name"
(Устанавливают имя пользователя, связанный с коммитами)

НА МАКЕ ЕСЛИ ХОТИМ ЧТО-ТО ПОСМОТРЕТЬ ГЛОБАЛЬНО ДОБАВЛЯЕМ ПЕРЕД КОМАНДОЙ СЛОВО **SUDO** (sudo git ...)

ТАКЖЕ ЧТОБЫ УВИДЕТЬ СКРЫТЫЕ ФАЙЛЫ В ПАПКЕ НА МАКЕ, В САМОЙ ПАПКЕ ЖМЕМ **Command + Shift + .**

git config --list (Показывает список конфигураций)

git config --list | findstr user.name
(Показывает найденную строку в конфиге по запросу)

git branch (Показывает список локальных веток в репозитории)

git branch -a (Показывает список локальных и удаленных веток в репозитории)

git branch -M <branch_name> (изменяет название текущей ветки локально)

git branch -D <branch_name>
(удаляет ветку локально)

git branch -m <old_branch_name> <new_branch_name>
(изменяет название выбранной ветки на новое, локально)

git switch <branch_name>
(Переключает на ветку с указанным именем)

git switch -c <branch_name>
(Переключает на новую ветку с указанным именем)

git switch -
(Переключает обратно на предыдущую ветку)

git checkout <branch_name> (Переключает на ветку с указанным именем, тоже самое что и git switch <branch_name>)

git checkout -b <branch_name> (Переключает на новую ветку с указанным именем, тоже самое что и git switch -c <branch_name>)

git add <file name>
(Добавляет измененный/созданный файл в индекс (staging area) для последующего коммита)
git add .

(Добавляет все измененные/созданные файлы в индекс (staging area) для последующего коммита)

git commit -m "your commit name"

(Создает коммит с указанным сообщением, включая все файлы, добавленные в индекс)

git commit --amend -m "an updated commit message"

(Изменяет созданный коммит с указанным сообщением)

git commit --amend --no-edit

(Изменяет созданный коммит оставляя предыдущее сообщение)

git status (Показывает текущее состояние репозитория, включая неотслеживаемые файлы, измененные файлы и т.д.)

git push <branch_name>

(Отправляет изменения в удаленный репозиторий)

git push -u origin <branch_name>

(Отправляет изменения в удаленный репозиторий, это команда только когда локальная ветка отсутствует в удаленном репозитории)

git push origin --delete <branch_name>

(Удаляет ветку в удаленном репозитории)

git pull

(Получает последние изменения из удаленного репозитория и объединяет их с текущей веткой)

git log

(Показывает историю коммитов в репозитории)

git log --oneline

(Показывает историю коммитов в одну строку)

git log --all --decorate --oneline --graph

(Показывает историю коммитов красиво)

git cherry-pick <commit_hash>

(хэш коммита это просто уникальный id)

git cherry-pick --continue (Продолжает операцию cherry-pick)

git cherry-pick --abort (Отменяет операцию cherry-pick)

git merge <branch_name> (Объединяет указанную ветку с текущей веткой)

git merge --abort

(Отменяет объединение)

STASH:

git stash (Сохраняет текущие изменения в отдельной области (stash), чтобы временно переключиться на другую ветку без коммита)

git stash save "stash_message" (Сохраняет текущие изменения в отдельной области (stash), с установленным названием)

git stash save -u "stash_message" (флаг -u -include-untracked означает включать неотслеживаемые файлы в stash, команда также сохраняет текущие изменения в отдельной области (stash), с установленным названием)

git stash list (показывает список прятаний)

git stash apply <number> (применяет прятание по номеру или без номера, то последнее)

git stash pop <number> (применяет и удаляет прятание по номеру или без номера, то последнее)

git stash drop <number> (удаляет прятание по номеру или без номера, то последнее)

git stash show <number> (показывает какие изменения содержатся в прятание по номеру или последнем если без номера)

git stash clear (удаляет все прятания)

git stash branch <название> <stash_number> (создает новую ветку с последним прятаньем или по номеру прятания, и затем удаляет последнее или по номеру прятанье (как *git stash pop*))

RESET:

A - B - C - HEAD



git reset B(^ или ~)

git reset --soft B(^ или ~)

git reset --mixed B(^ или ~)

git reset --hard B(^ или ~)

если пушить изменения после **git reset** в репозиторий и если в этом репозитории присутствует отмененный коммит то нужно вызывать **git push -f (-f или --force)**. Т.е пушим форсированно

 означает "(первый) родительский элемент".  аналогичен, но он принимает число как аргумент и в основном означает "предок". Итак, например:

```
HEAD          = latest commit
HEAD^         = HEAD~1 = parent of latest commit
HEAD^^        = HEAD~2 = grandparent of latest commit
HEAD~100      = 100th ancestor of latest commit
```

REVERT:

A - B - C - HEAD

git revert --no-commit A

git revert --no-commit A (можно A B C)

(Тоже само что и **git restore --source A (можно A B C)**)

git revert --abort

git restore --source A .

(Тоже само что и: **git revert --no-commit A (можно A B C)**)

git blame <file> (Показывает автора и последний коммит, внесшие изменения в каждую строку указанного файла)

git blame -L <start>,<end> <file>

(Показывает автора и последний коммит, внесшие изменения в указанный диапазон строк файла)

DIFF (Показывает изменения в сравнении, эти команды часто заменяет ui интерфейс в IDE):

```
git diff
git diff HEAD
git diff <branch_name1> <branch_name2>
git diff <commit_hash> <commit_hash>
```



Основы работы с командной строкой (Терминалом) на Mac OS X:

<https://vertex-academy.com/tutorials/ru/komandy-komandnaya-stroka-terminal-v-macos/>