

# Engineering Open Source Projects

Anton Grishin (@alchemmunist)

Introduction to the EOSP course.



# Table of contents

---

`hello` Getting to know each other better.

---

`open-source` Diving into the World of OpenSource.

---

`course` Figure out what awaits us in the course.

---

`practice` Come up with a name for the project together.

---

`live-demo` Lay the foundation for the project together.

---

`api` Getting to know the GitHub API.

---

`github-flow` How are Open Source development processes organized.

# Let's get acquainted

hello

What do you think false here?

- I'm Anton. Study at 2nd course of SWE at Central University.
- Involved in the development of 70+ repositories and have sent ~1600 commits.
- I'm graduate of Yandex Lyceum golden certificate.
- I used Linux as main operation system for last 3 years.
- I can type 120 words per minute on qwerty keyboard.
- I used to be a professional volleyball player.
- I am using only terminal for developing.
- Author of blog: alchemmist.xyz

Tell us about yourself 

# What is Open Source

 open-source

Source code of project is open to see for everyone.

Open to copy? Open to appropriate? Open to sell?

# Postgres DBMS (MIT/BSD)

open-source

- Full access to the source code
- Use in commercial products
- Selling as part of your own product
- Closing your code on top of PostgreSQL
- Please, save the license and author name

The screenshot shows the GitHub interface for the 'postgres' organization. At the top, there's a navigation bar with 'Overview' (selected), 'Repositories' (5), 'Projects', 'Packages', 'People' (6), and a search bar.

**PostgreSQL** (Verified) - **1.8k followers** Worldwide <https://www.postgresql.org/>

**Pinned**

**postgres** (Public)

Mirror of the official PostgreSQL GIT repository. Note that this is just a \*mirror\* - we don't work with pull requests on github. To contribute, please see [https://wiki.postgresql.org/wiki/Submitting\\_a\\_Patch](https://wiki.postgresql.org/wiki/Submitting_a_Patch)

● C ⭐ 19.6k ⚡ 5.3k

**Repositories**

Find a repository... Type Language

**postgres** (Public)

Mirror of the official PostgreSQL GIT repository. Note that this is just a \*mirror\* - we don't work with pull requests on github. To contribute, please see [https://wiki.postgresql.org/wiki/Submitting\\_a\\_Patch](https://wiki.postgresql.org/wiki/Submitting_a_Patch)

● C ⭐ 19.554 ⚡ 5.329 ○ 0 ⚡ 0 Updated 15 hours ago

**pgweb** (Public)

Mirror of the code behind [www.postgresql.org](http://www.postgresql.org)

● HTML ⭐ 80 ⚡ 48 ○ 0 ⚡ 0 Updated 3 days ago

**pgcommitfest** (Public)

The source code for <https://commitfest.postgresql.org>

● Python ⭐ 12 ⚡ 19 ○ 13 ⚡ 1 Updated 3 days ago

## Postgres PRO (EULA)

[open-source](#)

- EULA — End User License Agreement
- Distributed for a money
- OpenCore model



Весь опыт  
разработчиков  
PostgreSQL для вас

# Angular JS (MIT)

[open-source](#)

- Fully open (as Postgres)
- Repo is exist, but deprecated
- No support
- No improves
- No bug fixes
- The reason is switched on TypeScript

This repository was archived by the owner on Apr 12, 2024. It is now read-only.

 angular.js [Public archive](#)

[Watch 3738](#) [Fork 27.3k](#) [Star 59k](#)

[master](#) [Go to file](#) [Code](#) [About](#)

	Brocco	chore: update post LT...
	.circleci	chore(build): supp...
	.github	chore(*): prep for ...
	benchmarks	chore(benchpress...
	css	style(css) separat...
	docs	chore: update pos...
	i18n	chore(i18n): fix U...
	images	docs(*): optimize i...
	lib	chore(ci): deploy t...
	logs	creating logs/ and...
	scripts	chore(functions): ...
	src	docs(\$parse): fix t...
	test	test(Angular): fix a...

AngularJS - HTML enhanced web apps!

[angularjs.org](#)

[Readme](#) [MIT license](#) [Code of conduct](#) [Contributing](#) [Security policy](#) [Activity](#) [Custom properties](#)

[59k stars](#) [3.7k watching](#) [27.3k forks](#) [Report repository](#)

**Releases**

[209 tags](#)

# Linux kernel (GPLv2)

open-source

- Full access to the source code
- Use in commercial products
- Forks must remain GPL if distributed: copyleft

The screenshot shows the GitHub repository for the Linux kernel. The repository is named "linux" and is public. It has 1 branch and 914 tags. The most recent commit is a merge from the "erofs-for-6.19-rc5-fixes" tag. The commit message is "Merge tag 'erofs-for-6.19-rc5-fixes' of git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-6.19-rc5". The commit was made by "torvalds" at b6151c4 · 7 hours ago. The commit details show merges into various kernel subsystems: Documentation, LICENSES, arch, block, certs, crypto, drivers, fs, include, init, io\_uring, ipc, kernel, lib, mm, and net. Each merge is accompanied by a detailed commit message describing the changes.

Merge Subsystem	Commit Message
Documentation	Merge tag 'soc-fixes-6.19' of git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-6.19-rc5
LICENSES	LICENSES: Add modern form of the LGPL-2.1 tags to the ...
arch	Merge tag 'arm64-fixes' of git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-6.19-rc5
block	Merge tag 'block-6.19-20260109' of git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-6.19-rc5
certs	sign-file,extract-cert: use pkcs11 provider for OPENSSL M...
crypto	crypto: seqiv - Do not use req->iv after crypto_aead_encr...
drivers	Merge tag 'block-6.19-20260109' of git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-6.19-rc5
fs	Merge tag 'erofs-for-6.19-rc5-fixes' of git://git.kernel.org/...
include	Merge tag 'acpi-6.19-rc5' of git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-6.19-rc5
init	Merge tag 'mm-nonmm-stable-2025-12-06-11-14' of git://git.kernel.org/...
io_uring	Merge tag 'io_uring-6.19-20260109' of git://git.kernel.org/...
ipc	Merge tag 'mm-nonmm-stable-2025-12-06-11-14' of git://git.kernel.org/...
kernel	Merge tag 'pm-6.19-rc5' of git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-6.19-rc5
lib	idr: fix idr_alloc() returning an ID out of range
mm	mm/ksm: fix pte_unmap_unlock of wrong address in bre...
net	Merge tag 'ceph-for-6.19-rc5' of https://github.com/ceph...

# OpenSource is the foundation of modern IT open-source

A space where technology is emerging.



*«99% of Fortune 500 companies currently use open source software. <...> Over 56 million developers are contributing to open source projects. <...> Due to ever-rising workloads, the Linux operating systems market is expected to grow at the rate of 7% a year, reaching \$9.7 billion by 2024.»*

**Pranay Ahlawat, Boston Consulting Group**

Article «Why You Need an Open Source Software Strategy», April 2021

# The idea and goal of our project

course

Course is totally practice-driven.

- Build a practical system to evaluate developer contributions based on GitHub activity
- Learn modular software design: library → CLI → Telegram bot
- Practice real-world Open Source workflows: issues, pull requests, reviews and so on
- Focus on clean, maintainable, and testable code
- Experience CI/CD pipelines, releases, and deployment automation
- Document, setting and organize projects properly
- Develop skills to build slide deck and pitch project publicly

# Two main problem cases

course

## Profile analytics

- HR wants quick insight into a developer's activity without digging into GitHub manually
- Analyze the **entire GitHub profile**: all repositories, contributions, and activity history
- Understand which languages and technologies a developer uses
- Track contributions across repositories: commits, pull requests, issues
- Generate a concise profile summary for recruitment decisions

## Leader board of team

- Team leads want visibility into team productivity
- Analyze **contributions within a single repository** to compare team members fairly
- Track per-developer metrics: code quality, review participation, issue resolution
- Identify who is actively contributing and who may need support or guidance
- Provide fair, data-driven insights to improve collaboration and team performance

# Three setups, three projects

course

## Python library

- Calculate developer contribution metrics from GitHub data
- Provide reusable, modular functions for metrics computation
- Include comprehensive unit tests and follow TDD approach
- Serve as the core foundation for CLI and bot integrations
- Support easy extension and maintainability

## CLI

- Provide command-line access to library metrics
- Support multiple commands, flags, and options
- Enable fetching, displaying, and exporting data conveniently
- Handle errors gracefully and show meaningful messages
- Integrate with CI/CD for automated releases

## Telegram bot

- Provide easy access to metrics via Telegram interface
- Interact with users, handle commands and queries
- Securely manage secrets and API tokens
- Fetch data from library and format it for user-friendly display
- Support notifications, updates, and automated alerts

# How should this course be perceived? course

Motivation and mindset.



*«It's a bit sad to think of all the high school kids turning their backs on building treehouses and sitting in class dutifully learning about Darwin or Newton to pass some exam, when the work that made Darwin and Newton famous was actually closer in spirit to building treehouses than studying for exams.»*

**Paul Graham**

Essay «A Project of One's Own», June 2021

It's time to come up with a name!

practice

Go to Figma board!

-----

# Let's make first step

[live-demo](#)

Creating GitHub organization and repo.

# Introduction to GitHub API.

api

```
gh api /octocat
```

# GitHub API is just HTTP api

Any tool that can send HTTP requests can work with GitHub API.

- `gh` — convenient wrapper around the API
- `curl` — raw HTTP from terminal
- `Python` — programmatic access for automation and logic

# Official gh cli api

The cli utility for using all github functionality from terminal.

Install into your shell:

```
# Mac:  
brew install gh  
  
# Windows:  
winget install --id GitHub.cli  
  
# Arch:  
sudo pacman -S github-cli
```

And try something, for example:

```
gh api /users/alchemmist
```

As result:

```
1  {  
2      "login": "alchemmist",  
3      "avatar_url": "https://avatars.githubusercontent.com/u/104511335?v=4",  
4      "html_url": "https://github.com/alchemmist",  
5      "followers_url": "https://api.github.com/users/alchemmist/followers",  
6      "subscriptions_url": "https://api.github.com/users/alchemmist/subscriptions",  
7      "repos_url": "https://api.github.com/users/alchemmist/repos",  
8      "type": "User",  
9      "name": "Anton Grishin",  
10     "blog": "alchemmist.xyz?utm_source=github",  
11     "location": "Russia, Moscow",  
12     "email": "anton.ingrish@gmail.com",  
13     "followers": 18,  
14     "created_at": "2022-04-27T14:12:26Z",  
15     "updated_at": "2026-01-09T06:23:55Z",  
16     ...  
17 }
```

## More in documentation

api

---

# Using curl for GitHub API

api

Raw HTTP requests from terminal.

Send GET authenticated request:

```
curl -L \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <TOKEN>" \
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/repos/alchemmist/eosp/stats/contributors
```

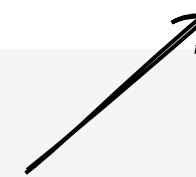
According to the documentation

- w — Start of the week, given as a Unix timestamp.
- a — Number of additions
- d — Number of deletions
- c — Number of commits

As result:

```
1  [
2    {
3      "total": 37,
4      "weeks": [
5        {
6          "w": 1765670400,
7          "a": 6477,
8          "d": 0,
9          "c": 1},
10         {"w": 1768089600,
11          "a": 0,
12          "d": 0,
13          "c": 0}, ...
14     ],
15     "author": {
16       "login": "alchemmist",
17       "id": 104511335,
18       "node_id": "U_kgDOBjq3Zw", ...
19     }
20   ]
```

Sun Dec 14 2025



## More in documentation

api

---

# Using GitHub API with Python

api

Efficient, parallel, production-ready requests.

Example with `httpx`:

```
import asyncio, httpx

async def fetch_prs(username):
    url = f"https://api.github.com/search/issues?\n    q=author:{username}+type:pr+created:>2025-01-01"
    async with httpx.AsyncClient() as client:
        resp = await client.get(
            url,
            headers={
                "Authorization": "Bearer <TOKEN>"
            },
        )
        data = resp.json()
        for pr in data["items"]:
            print(f"{pr['title']}\n\t-> {pr['html_url']}")

asyncio.run(fetch_prs("alchemmist"))
```

As result:

```
Add alchemmist.xyz individual blog
-> https://github.com/kilimchoi/engineering-blogs/pull/1201
Add alchemmist.xyz personal blog
-> https://github.com/learn-anything/blogs/pull/21
Add alchemmist.xyz blog
-> https://github.com/logancyang/awesome-personal-websites/pull/1
Add alchemmist.xyz blog
-> https://github.com/jkup/awesome-personal-blogs/pull/173
Add @alchemmist_blog to personal blogs section
-> https://github.com/goq/telegram-list/pull/992
Add @alchemmist_blog to personal blogs section
-> https://github.com/alchemist/telegram-list/pull/1
Add a "quiet" exit (#104)
-> https://github.com/cqfn/aibolit/pull/818
```

\*GraphQL - a query language for APIs that lets you request exactly the data you need in a single query, without extra fields.

# GitHub API Limitations

api

Understanding these limits is essential when building scalable, reliable systems for collecting GitHub data.

- Rate limits: **5000 requests per hour** for authenticated users
- Rate limits: **60 requests per hour** for unauthenticated users
- Pagination: most endpoints return max **100 items per page**, need to handle paging
- Private data requires proper authentication and scopes
- GraphQL vs REST: some data easier via GraphQL, but query complexity may hit limits
- API responses may be delayed or cached; real-time metrics may require retries
- Some endpoints change over time; library must handle API versioning

# Git vs GitHub Flow

github-flow

- Git: a version control system, local, focused on branching and commits
- GitHub: an online platform for Git with social and collaboration tools
- GitHub Flow: a lightweight workflow that allows fast development and integration via pull requests
- Main goal: a safe, transparent, and repeatable development process within a team

# Why GitHub Flow?

- All changes go through **pull requests** → code review, CI/CD checks
- Encourages small, incremental updates rather than long-lived branches
- Clear separation: `main` branch is always deployable
- Integration with issues and project boards → planning and tracking in one place
- Transparency and collaboration: team members can comment, review, approve, or reject changes

# Key Entities in GitHub Flow

- **Issue** – describes a bug, feature, task, or question; starting point for development
- **Branch** – isolated workspace for a specific feature or fix
- **Commit** – individual changes tracked in Git history
- **Pull Request (PR)** – proposes changes from a branch into `main`; facilitates review and discussion
- **Code Review** – teammates review PRs to ensure quality and maintainability
- **CI/CD checks** – automated tests, linting, build, and deployment pipelines
- **Merge** – approved PR is merged into `main`, usually triggers deployment

# GitHub Flow in Action

Visual representation of the workflow:

1. Create an **issue**
2. Create a **branch** for the issue
3. Commit changes locally and push to GitHub
4. Open a **pull request** referencing the issue
5. Conduct **code review** and CI/CD validation
6. Merge into **main** after approval
7. Deploy automatically (if CI/CD is configured)

## Issue → Merge: Step by Step

- **Step 1:** Issue is created describing bug, feature, or task
- **Step 2:** Developer creates a branch named after issue (e.g., `issue-42-fix-login`)
- **Step 3:** Commit small, meaningful changes to the branch
- **Step 4:** Push branch to GitHub and open a **pull request** linked to the issue
- **Step 5:** Team reviews code, CI/CD runs tests automatically
- **Step 6:** Address feedback, push updates to the PR
- **Step 7:** PR approved → merge into `main`
- **Step 8:** `main` is deployable, issue closed, cycle completed

# Best Practices in GitHub Flow

- Keep branches **short-lived** → frequent integration reduces conflicts
- Write **descriptive commits** → history becomes meaningful
- Reference issues in PRs → link work to context
- Use **templates** for PRs and issues → standardize workflow
- Automate as much as possible → CI/CD, tests, linters, code quality checks
- Encourage **review culture** → better code, knowledge sharing, accountability