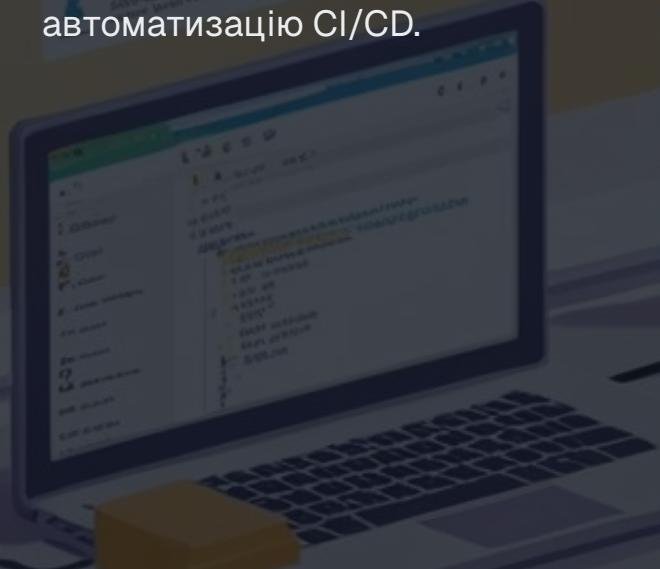


YAML для GitHub Actions

Повний практичний посібник з синтаксису YAML для опису workflow у GitHub Actions — від базової структури до складних тригерів і фільтрів. Ідеально для розробників і DevOps-інженерів, які починають або вдосконалюють автоматизацію CI/CD.



Вступ

Workflow в GitHub Actions описуються у форматі **YAML** (YAML Ain't Markup Language) — текстовому форматі для серіалізації даних, який завоював особливу популярність у світі DevOps та CI/CD. Завдяки своїй читабельності та природній підтримці структурованих даних, YAML став стандартом де-факто для опису конфігурацій автоматизованих процесів.

Коли ви працюєте з GitHub Actions, кожен workflow — це YAML-файл, що зберігається у директорії `.github/workflows/` вашого репозиторію. Цей файл визначає, **коли** запускати автоматизацію, **де** її виконувати і **які кроки** здійснювати. Розуміння основ YAML дозволяє вам впевнено читати, писати та підтримувати workflow без загадкових помилок і синтаксичних проблем.

Навіть якщо ви вже знайомі з іншими форматами конфігурацій, YAML має свої особливості — насамперед значущі відступи, які визначають вкладеність структур. Опанувавши ці правила, ви зможете ефективно автоматизувати збірку, тестування та деплой ваших проектів прямо з репозиторію GitHub.

Читабельність

Формат зрозумілий людині без додаткових інструментів

Стандарт DevOps

Використовується в Docker, K8s, Ansible та GitHub Actions

Простота підтримки

Коментарі і структура спрощують документування конфігів

Чому саме YAML

Серед усіх форматів для опису конфігурацій — JSON, XML, TOML, INI — YAML виділяється поєднанням людиночитабельності та виразності. Він дозволяє природно описувати складні ієрархічні структури без громіздкого синтаксису, що особливо важливо у великих workflow-файлах. Саме тому провідні DevOps-платформи обрали YAML як основний формат конфігурацій.



Читабельність

Відступи задають вкладеність — формат інтуїтивно зрозумілий для ока. Структура видна одразу без знання специфічного синтаксису.



Мінімалізм

Немає зайвих дужок як у JSON чи XML — менше синтаксичного шуму, більше змісту.



Типи даних

Підтримка рядків, чисел, дат, булевих значень, масивів і вкладених об'єктів без зайвих анотацій.



Структуровані дані

Ієрархія зрозуміла і людині, і парсеру. Вкладеність об'єктів, масиві і скалярні значення виражаються природно.



Мовно-незалежний

Однаковий формат у різних мовах програмування і інструментах — Docker, Kubernetes, Ansible, GitHub Actions.



Коментарі

Вбудовані коментарі через # — на відміну від JSON, що суттєво спрощує документування конфігурацій.



💡 YAML використовується у Docker Compose, Kubernetes, Ansible, Terraform та GitHub Actions — вивчивши його один раз, ви отримуєте навичку, що працює в усьому DevOps-екосистемі.

Базова структура workflow-файлу

Кожен workflow-файл GitHub Actions має передбачувану ієрархічну структуру. Знання цієї структури — відправна точка для будь-якої автоматизації. Файл зберігається за шляхом `.github/workflows/назва.yml` і складається з кількох обов'язкових і необов'язкових секцій.

Шаблон структури

```
name: <назва workflow>

on: <подія або список подій>

jobs:
  job_1:
    name: <назва першого job>
    runs-on: <тип машини>
    steps:
      - name: <крок 1>
        run: |
          <команди>
      - name: <крок 2>
        run: |
          <команди>
  job_2:
    name: <назва другого job>
    runs-on: <тип машини>
    steps:
      - name: <крок 1>
        run: |
          <команди>
```

Ключові компоненти

name

Відображається в інтерфейсі GitHub як назва workflow. Допомагає ідентифікувати запуски у вкладці Actions.

on

Визначає, **коли** запускати workflow — події push, pull_request, schedule та інші.

jobs

Один або кілька job; кожен має власний runs-on (тип runner) і список steps — кроків виконання.

Важливо пам'ятати: у YAML відступи мають значення. Використовуйте **пробіли** (не табуляцію) і будьте послідовні у кількості пробілів на кожному рівні вкладеності. Типова помилка початківців — змішування табів і пробілів, що призводить до синтаксичних помилок при парсингу.

Події для запуску: розклад (cron)

Один із найпотужніших тригерів GitHub Actions — запуск за розкладом через синтаксис **POSIX cron**. Це дозволяє автоматично виконувати workflow у визначений час: наприклад, щоночі запускати тести, щотижня генерувати звіти або щодня оновлювати залежності. Усі часові мітки у cron-розкладі використовують часовий пояс **UTC**.

Мінімальний інтервал між запусками — **кожні 5 хвилин**. Частіші запуски не підтримуються платформою. За замовчуванням workflow виконується на останньому коміті гілки за замовчуванням репозиторію (зазвичай `main`).

Приклад: двічі на добу

```
on:  
  schedule:  
    # * — спеціальний символ у YAML,  
    # тому рядок беремо в лапки  
    - cron: '30 5,17 * * *'
```

Цей cron-вираз запускає workflow щодня о **05:30 UTC** та о **17:30 UTC**.

Синтаксис cron

Поле	Значення
хвилини	0–59
и	
години	0–23 (UTC)
день місяця	1–31
місяць	1–12
день тижня	0–6 (нд–сб)
*	будь-яке значення
,	перелік значень

- ⚠ Символ `*` є спеціальним у YAML. Завжди беріть cron-вираз у одинарні лапки: `'30 5,17 * * *'`, щоб уникнути помилок парсингу.

Події: один тригер

Найпростіша форма визначення події — вказати один тригер безпосередньо після ключа `on`. Це мінімалістичний синтаксис, який чудово підходить для простих workflow, де потрібно реагувати лише на одну подію.

Синтаксис

```
on: push
```

Workflow запускається при будь-якому **push** у будь-яку гілку репозиторію.

Коли використовувати

Один тригер без додаткових фільтрів ідеально підходить для невеликих репозиторіїв або внутрішніх проектів, де будь-який **push** повинен запустити перевірку. Наприклад, для запуску юніт-тестів або лінтингу при кожному коміті в репозиторій.

- Швидкий зворотний зв'язок при кожному коміті
- Мінімальна конфігурація — менше помилок
- Підходить для проектів з однією гілкою розробки
- Легко розширити фільтрами пізніше

Важливо розуміти: без фільтрів цей тригер реагує на **будь-який push** у репозиторій, включно з `push` у feature-гілки, `tags` та захищенні гілки. Для великих проектів із командою розробників це може призводити до надмірного споживання хвилин Actions. У таких випадках рекомендується додавати фільтри гілок або шляхів, про які йтиметься в наступних розділах.

Події: кілька тригерів

Часто workflow має реагувати відразу на кілька різних подій. GitHub Actions підтримує компактний синтаксис у вигляді масиву YAML для переліку кількох тригерів в одному рядку. Це дозволяє уникнути дублювання workflow-файлів для схожої логіки.

Синтаксис масиву

```
on: [push, fork]
```

Запуск і при **push**, і при **fork** репозиторію — обидві події в одному рядку.

Популярні комбінації тригерів

→ **[push, pull_request]**

Запуск тестів при push і при відкритті PR — найпоширеніша комбінація для CI.

→ **[push, workflow_dispatch]**

Автоматичний запуск при push і ручний запуск через UI GitHub.

→ **[push, fork]**

Реакція на активність у репозиторії та при його форкуванні.

Коли потрібно задати додаткові параметри для кожної події (наприклад, фільтри гілок), масивний синтаксис вже не підходить — необхідно переходити до розширеного запису з вкладеними ключами. Але для простого переліку без налаштувань `on: [push, fork]` — найчистіше і найлаконічніше рішення.

Події: фільтр по гілках (push)

Запуск workflow при **будь-якому** push у репозиторій часто є надлишковим. GitHub Actions дозволяє точно налаштувати, при push у які гілки має запускатися workflow. Це особливо важливо для виробничих пайплайнів, де деплой або важливі перевірки мають виконуватись лише для певних гілок.

Фільтр за гілкою

```
on:  
  push:  
    branches:  
      - main
```

Workflow запускається лише при push у гілку `main`.

Розширені варіанти фільтрів

```
on:  
  push:  
    branches:  
      - main  
      - develop  
      - 'releases/**'  
      - 'feature/*'
```

Підтримуються **glob-паттерни**: `**` для будь-якої вкладеності, `*` для одного рівня шляху.

Фільтр `branches` підтримує glob-паттерни, що робить його надзвичайно гнучким. Наприклад, `releases/**` охоплює всі гілки, що починаються з `releases/`, а `feature/*` — усі `feature`-гілки без вкладених підрозділів. Це дозволяє описати складну логіку тригерів кількома рядками YAML без дублювання workflow-файлів.

- 💡 Паралельно з `branches` існує `branches-ignore` для виключення конкретних гілок — зручно, коли простіше перерахувати винятки, ніж дозволені гілки.

Події: запуск на Pull Request

Тригер `pull_request` — один із найважливіших у CI/CD процесах. Він дозволяє автоматично запускати перевірки, тести та аналіз коду при відкритті або оновленні Pull Request. Це є ключовим елементом code review процесу: жоден код не потрапляє в основну гілку без проходження автоматичних перевірок.

Базовий тригер PR

```
on:  
  pull_request:
```

Запускається при будь-якій дії з Pull Request у репозиторії.

Виключення гілок

```
branches-ignore:  
  - 'mona/octocat'  
  - 'releases/**-alpha'
```

Практичне застосування

Тригер `pull_request` без додаткових параметрів реагує на широкий спектр дій: відкриття PR, додавання комітів, синхронізацію та повторне відкриття. Це забезпечує, що кожна зміна в PR проходить перевірку.

Ключ `branches-ignore` дозволяє виключити конкретні гілки із тригерів. Наприклад:

- '`mona/octocat`' — точна назва гілки для ігнорування
- '`releases/**-alpha`' — glob-паттерн для всіх alpha-релізних гілок
- Зручно для тимчасових або службових гілок, що не потребують CI
- Зменшує споживання хвилин Actions для нерелевантних PR

Подiї: комбiнацiя branches та branches-ignore

Для тонкого налаштування тригерів GitHub Actions пiдтримує одночасне використання включень і виключень у межах одного тригера через символ !. Це особливо потужний пiдхiд, коли потрiбно охопити широкий набiр гiлок, але виключити певнi пiдмножини.

Синтаксис iз виключенням

```
on:  
  pull_request:  
    branches:  
      - 'releases/**'  
      - '!releases/**-alpha'
```

Символ ! перед назвою гiлки означає **виключення** з тригери.

Логiка роботи

✓ Запускається

releases/v1.0
releases/2024-stable
releases/hotfix

✗ Ігнорується

releases/v1.0-alpha
releases/beta-alpha
releases/test-alpha

Цей пiдхiд дозволяє описати складну логiку включення та виключення гiлок в однiй секцiї branches. Порядок рядкiв має значення: GitHub Actions обробляє паттерни послiдовно. Рядки з ! перекривають попереднi включення для гiлок, що вiдповiдають цьому паттерну. У прикладi вище: всi гiлки releases/** включено, але тi, що закiнчуються на -alpha — виключено. Це iдеально для сценарiю, коли alpha-версiї тестиються окремим workflow або зовсiм не потребують CI-перевiрок.

Події: фільтр за шляхами (paths-ignore)

Фільтрація за шляхами — потужний інструмент оптимізації CI/CD пайплайнів. Замість запуску workflow при **будь-якій** зміні у репозиторії, ви можете вказати, зміни яких файлів або директорій слід ігнорувати. Це суттєво економить хвилини Actions і прискорює зворотний зв'язок для розробників.

Приклад: ігнорувати docs/

```
on:  
push:  
  paths-ignore:  
    - 'docs/**'
```

Workflow **не** запускається, якщо змінились лише файли в директорії `docs/`.

Важлива логіка роботи

Тільки ігноровані файли змінились

→ Workflow **НЕ запускається**. Наприклад, оновлення README в `docs/` не тригерить CI.

Хоч один файл поза ігнорованими змінився

→ Workflow **запускається**. Зміна коду разом із документацією все одно тригерить пайплайн.

Альтернатива — ключ `paths` (без `ignore`), який запускає workflow **лише** при змінах у вказаних шляхах. Наприклад, `paths: ['src/**', 'tests/**']` запустить workflow тільки якщо змінились файли у цих директоріях. Комбінування `paths` і `branches` дає максимально точний контроль над тригерами і є найкращою практикою для великих монорепозиторіїв із кількома мікросервісами.

- 💡 Використовуйте `paths-ignore: ['docs/**', '*.md', '.gitignore']` щоб уникнути зайвих запусків при оновленні документації та конфігураційних файлів, що не впливають на код.

Події: типи подій (types)

Деякі події GitHub Actions мають кілька підтипів — наприклад, Issue можна відкрити, закрити, позначити міткою, призначити відповідального тощо. За замовчуванням тригер реагує на **всі типи** підподій, але для точного контролю можна вказати конкретні типи через ключ types.

Приклад: issues та pull_request

```
on:  
  issues:  
    types:  
      - closed  
  pull_request:  
    types:  
      - opened  
      - closed  
      - labeled
```

Типові сценарії застосування

1 Автоматичне прибирання при закритті Issue

Тригер issues: types: [closed] запускає workflow для архівації або очищення пов'язаних ресурсів.

2 Різна логіка для opened vs labeled

При відкритті PR — запуск повного тест-сюїту. При додаванні мітки — деплой у preview-середовище.

3 Умови в jobs/steps

Тип події доступний через github.event_name та github.event.action для умовної логіки в кроках.

Найпоширеніші типи для pull_request: opened, synchronize, closed, labeled, reopened. Для issues: opened, edited, closed, assigned. Використання types дозволяє створювати дуже точні тригери і уникати зайвих запусків workflow при нерелевантних подіях — що особливо важливо для оптимізації витрат на GitHub Actions у великих командах.

Висновок

YAML — це не просто формат файлу, це мова конфігурації, яка стоїть за автоматизацією сучасних DevOps-процесів. Опанувавши його основи в контексті GitHub Actions, ви отримуєте потужний інструментарій для побудови надійних, ефективних і легко підтримуваних CI/CD пайплайнів.

YAML як основа

Читабельний формат з підтримкою коментарів, типів даних і ієрархічних структур — ідеальний для опису workflow.

Ключові секції

`name`, `on` (події та фільтри), `jobs` з `runs-on` і `steps` — основа будь-якого workflow.

Точні тригери

Фільтри за гілками, шляхами та типами подій дозволяють уникати зайвих запусків і економити хвилини Actions.



Вивчити YAML

Налаштuvат и тригери

Оптимізуват и фільтрами

Наступний крок після освоєння YAML та тригерів — заглибитися в Jobs і Steps: вивчити, як задавати залежності між jobs, використовувати матриці для паралельного тестування на різних платформах, підключати готові Actions з Marketplace та керувати секретами і змінними середовища. GitHub Actions — це надзвичайно гнучка платформа, і YAML є вашим ключем до її можливостей.

- ☐ Практика — найкращий вчитель. Почніть з простого workflow для вашого наступного проекту і поступово додавайте фільтри та складніші тригери в міру потреби.