

Рекомендуемый порядок расположения блоков внутри job в .gitlab-ci.yml?

В GitLab CI/CD есть рекомендованный порядок, который улучшает читаемость и поддержку pipeline.

□□ Стандартный порядок блоков внутри job

```
job_name:
  extends: .some_template # 1□ Наследование (если есть)
  stage: build           # 2□ Определение этапа
  tags:                  # 3□ Теги для раннера (если нужны)
  - docker
  needs:                 # 4□ Зависимости от других job'ов
  - previous_job
  variables:             # 5□ Локальные переменные для job
  SOME_VAR: "value"
  before_script:        # 6□ Подготовка окружения перед `script`
  - echo "Setting up environment..."
  script:                # 7□ Основной код
  - echo "Executing job..."
  after_script:         # 8□ Завершающие действия (если нужны)
  - echo "Cleanup after job"
  artifacts:            # 9□ Артефакты (если нужны)
  paths:
  - my_output/
  expire_in: 1 hour
```

```
rules:          # [] Условия выполнения job
- if: '$CI_COMMIT_BRANCH == "main"'
allow_failure: false  # 1[]1[] Позволять ли падение job
retry: 2           # 1[]2[] Число повторных попыток
timeout: 30m       # 1[]3[] Лимит времени выполнения
```

[] Разбор структуры

Секция	Описание
<code>extends:</code>	Наследование от шаблонного <code>job</code> (если есть).
<code>stage:</code>	Назначение <code>job</code> к определенному этапу.
<code>tags:</code>	Указывает, на каких раннерах будет выполняться <code>job</code> .
<code>needs:</code>	Указывает зависимости от других <code>job</code> (если надо запустить <code>job</code> только после выполнения других).
<code>variables:</code>	Определяет переменные, видимые только внутри <code>job</code> .
<code>before_script:</code>	Выполняется один раз перед <code>script:</code> (установка окружения, логирование, авторизация и т. д.).
<code>script:</code>	Основной код <code>job</code> , выполняется после <code>before_script:</code> .
<code>after_script:</code>	Выполняется после <code>script:</code> , даже если <code>script:</code> упал. Подходит для логов, очистки файлов.
<code>artifacts:</code>	Определяет файлы, которые сохраняются после завершения <code>job</code> .
<code>rules:</code>	Условия выполнения <code>job</code> (например, только в <code>main</code> , только при изменениях в <code>src/</code>).
<code>allow_failure:</code>	Разрешает <code>job</code> завершаться с ошибкой (<code>true</code> → ошибки не остановят pipeline).
<code>retry:</code>	Число повторных попыток при падении <code>job</code> .
<code>timeout:</code>	Лимит времени выполнения <code>job</code> (например, <code>30m</code>).

[] Пример 1: Минимальная конфигурация `job`

Если у вас простой `job`, можно оставить только **ключевые секции**:

```
build_job:
  stage: build
  script:
    - echo "Building the project..."
```

Минимально необходимый `job`: `stage:` + `script:`.

▣▣ Пример 2: Расширенная конфигурация `job`

Если `job` сложный, используем **все ключевые блоки**:

```
deploy_job:
  extends: .deploy_template
  stage: deploy
  tags:
    - production-runner
  needs:
    - build_job
  variables:
    DEPLOY_ENV: "production"
  before_script:
    - echo "Подготовка к деплою..."
  script:
    - echo "Выполняем деплой..."
  after_script:
    - echo "Очистка после деплоя..."
  artifacts:
    paths:
      - deploy_logs/
    expire_in: 1 day
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
  allow_failure: false
```

retry: 2

timeout: 20m

Этот `job` должен выполняться ТОЛЬКО в `main`.

Если `job` упадет, он попытается перезапуститься дважды (`retry: 2`).

`before_script:` подготавливает окружение, а `after_script:` выполняет очистку.

Итог

Оптимальный порядок блоков `job:`

- `extends:` (если используется)
 - `stage:`
 - `tags:` (если есть)
 - `needs:` (если `job` зависит от других `job`'ов)
 - `variables:` (локальные переменные)
 - `before_script:` (подготовка окружения)
 - `script:` (основная логика `job`)
 - `after_script:` (очистка, логи)
 - `artifacts:` (если нужны)
 - `rules:` (условия выполнения)
 - `allow_failure:` (можно ли игнорировать ошибки)
 - `retry:` (количество повторных попыток)
 - `timeout:` (ограничение времени выполнения)
-

Revision #1

Created 31 January 2025 11:20:18 by Admin

Updated 31 January 2025 11:23:59 by Admin