

یک ورک فلوي ساده

GitHub Action



علیرضا ذوقفار

میزان سختی : ۴ از ۱۰

Github:

<https://github.com/azolfagharj>

Linkedin:

<https://www.linkedin.com/in/azolfagharj/>

Telegram:

@Azolfagharj

تو این مطلب میخواهم راه اندازی و تنظیم یک workflow ساده با Github Action رو توضیح بدم ، تو این آموزش روی مفاهیم پایه و اولیه تمرکز میکنم ، پس اگه حرفه ای هستید فکر نکنم برآتون مفید باشه 😊

قبلش لازمه يه سری مفاهیم اولیه ، نیازها و هدف این موضوع رو به صورت خلاصه بررسی کنیم

💡 چی شد که تصمیم گرفتم این پست رو بنویسم

چه وقت هایی که خودم برنامه نویسی میکنم چه وقت هایی که برای پروژه های مختلف workflow طراحی میکنم زیاد با این مورد مواجه میشم که خیلی از بچه های فنی یا اصلا از workflow استفاده نمیکنند یا شناخت کاملی نسبت که کارهایی که میتوانند با workflow انجام بدنند ندارد و این موضوع باعث میشه کارها رو به روش های سخت تری انجام بدند

یه شب وسط يه مهمونی 😊 داشتم يه workflow رو به رفیقم شهریار نشون میدادم و در موردش صحبت میکردیم که گفت اینی که تو نوشتی خیلی خفنه و متاسفانه وقتی workflow های موجود رو میبینم ، هیچ کدو مشون حتی از این موارد ساده هم استفاده نمیکنند ، خوب میشه اگه همین موارد ساده رو هم تو لینک دین پست کنی ، حتما برای خیلی ها مفیده

(اینکه وسط يه مهمونی ما داشتیم يه workflow رو بررسی می کردیم هم تو جای خودش قابل بحثه 😊)

⚡ چی لازم داریم؟

برای استفاده از این مطلب ، رسما به هیچ چیز نیاز نداری! کافیه کاری که میکنی به نوشتمن کدهای کامپیوترا مربوط باشه ، و برای چند دقیقه تمرکز اون مغز توانمندت رو روی این موضوع بذاری

🔗 همه چیز هست!

تمام کارهایی که تو این آموزش با هم انجام میدیم رو از قبل برات آماده کردم

میتونی تو لینک زیر بررسیش کنی

<https://github.com/azolfagharj/simple-github-action>

بریم سراغ مفاهیم:

گیت‌هاب اکشن و مفهوم ورک‌فلو

گیت‌هاب اکشن یک سیستم **CI/CD** داخلی گیت‌هابه که به ما اجازه میده کارهای خودکار مثل بیلد، تست و استقرار رو روی مخازن گیت‌هاب اجرا کنیم. این ابزار به کمک ورک‌فلوها، فرایندهای توسعه را ساده‌تر کرده و به ما کمک می‌کنه کارهای تکراری رو به صورت خودکار انجام بدیم.

ورک‌فلو (Workflow) چیه؟

یک ورک‌فلو مجموعه‌ای از دستوراته که به صورت خودکار روی یک مخزن اجرا می‌شه. این دستورات در یک فایل **YAML** تعریف شده و میتوانه تو شرایط مشخص مثل **push, pull request** یا یک زمانبندی خاص اجرا بشه.

چرا از ورک‌فلو استفاده کنیم؟

- ✓ خودکارسازی کارهای تکراری مثل بیلد، تست و استقرار
- ✓ اجرای تست‌ها قبل از ادغام کد و کاهش باگ‌ها
- ✓ انتشار خودکار پکیج‌ها و استقرار سریع‌تر
- ✓ اجرای اسکریپت‌ها روی سرور، Docker یا ماشین‌های مجازی

و کلی کار باحال دیگه که درنهایت فرایندها رو ساده، سریع و بی اشتباه تر میکنه

چه کارهایی رو میشه با ورک‌فلو انجام داد؟

- ✓ اجرای تست‌ها به صورت خودکار بعد از هر **push**
- ✓ بیلد و کامپایل پروژه تو محیط‌های مختلف مثل **Go, Python**
- ✓ انتشار خودکار پکیج‌ها تو سرویس‌هایی مثل **Docker Hub** یا **npm**
- ✓ استقرار خودکار روی سرور یا سرویس‌های ابری مثل **AWS**
- ✓ مدیریت خودکار مخزن، مثل برچسب گذاری ایشو ها

خب تا اینجا مفاهیم کلی رو متوجه شدیم

شما میتوانید workflow ها رو برای محیط های مختلف و پروژه های مختلف طراحی و اجرا کنید اما تو این پست من موضوع رو برای Github Action توضیح بدم تا در ساده ترین حالت برای دوستانی که ریپازیتوری های عمومی و خصوصی پروژه های شخصی کوچیک هم دارند قابل استفاده باشند

یه پروژه‌ی خیالی

برای درک بهتر موضوع بیايد همینجا یه پروژه‌ی خیالی تعریف کنیم که خیلی ساده است ،

فرض میکنیم ما داریم یه اسکریپت Bash مینویسیم و هیچ تیمی وجود نداره ، فقط خودمونیم و خودمون اما میخوایم هربار که تغییری رو پوش میکنیم ، یه نسخه از این تغییرات با توضیحاتش باقی بمونه و بتونیم بهش رجوع کنیم ، در کنارش ، یه ریلیز برای اون پوش ایجاد کنیم که بعضی فایل ها رو تولید کنه و تو ریلیز انجام بدیه یا حتی تغییرات رو روی سرور دیگه ای آپلود کنه

ایجاد یک فایل ورکفلو

- داخل ریپازیتوری یه دایرکتوری به نام `github` ایجاد میکنیم
- داخل دایرکتوری بالا یه دایرکتوری دیگه با نام `workflows` ایجاد میکنیم
- داخل دایرکتوری بالا یه فایل `Yaml` به نام دلخواه میسازیم ، مثلا `release.yml` تا اینجا باید چیزی شبیه ساختار زیر داشته باشیم

```
---- Simple project
|   ---- .github
|   |   ---- workflows
|   |   |   ---- release.yml
```

بخش های مختلف یک Workflow

همونطور که تا الان متوجه شدید workflow از ساختار **Yaml** استفاده میکنیم همینجا لازمه یادآوری کنم :
دوست خوب من حواست به **Indent** ها باشه که به خطای خوری 😊

نام **name** که با **workflow** مشخص میشے ✓

تو این بخش ما نام رو مشخص میکنیم و خیلی به سلیقه‌ی شما مربوط میشے اما سعی کنید اسمی بذارید که نفر بعدی که اون رو میخونه یا حتی خودتون بعدها گیج نشید و نفرین و لعنت نفرستید
مثال این بخش میتونه شکل زیر باشد

```
name: Release Automation
```

رویداد (Event) که با **on** مشخص میشے ✓

تو این بخش مشخص میکنیم چه رویدادی باعث اجرای این **Workflow** میشے ، در واقع تو این بخش **trigger** اجرای **Workflow** رو مشخص میکنیم . این رویداد به صورت خلاصه مشخص میکنه چه اتفاقی در کجا باعث اجرای **Workflow** میشے تو پروژه‌ی خیالیمون، ما میخوایم هر بار که یک **push** روی برنج اتفاق افتاد اجرا بشه پس بخش رویداد ما به شکل زیره **Main**

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

مجوزها که با **permissions** مشخص میشے ✓

تو این بخش مشخص میکنیم که این **workflow** چه مجوزهایی در مورد ریپازیتوری داره که شما دقیقاً میتوانید تو شن مشخص کنید ، دسترسی انجام چه کارهایی رو داره ، حدود ۱۳ آپشن مختلف داره که هر کدام میتونند یکی از سه حالت **read** ، **write** و **none** رو داشته باشند ، تو این پروژه ما احتیاجی به تمامش نداریم ، فقط میخوایم بتوانیم محتوای ریپازیتوری رو تغییر بدیم پس خیلی ساده

از دو تا از آپشن ها استفاده میکنیم تا workflow بتوانه بعضی بخش ها رو بخونه یا حتی تغییر بده ، تو قسمت های بعدی متوجه میشید چرا به این دسترسی نیاز داشتیم

```
permissions:  
  contents: write  
  id-token: write
```

وظایف که با jobs مشخص میشه ✓

این بخش ، هسته اصلی workflow هست و مشخص میکنه دقیقا چه وظایفی باید انجام بشه ، معمولاً دو بخش اصلی داره که محیط اجرا و قدم های اجرا رو مشخص میکنیم ، در ادامه بخش های مختلف رو برآتون توضیح میدم اما فعلاً حالت کلی بخش jobs رو نگاه کنید که به شکل زیره

```
jobs:  
  create-release:  
    runs-on: ubuntu-22.04  
    steps:  
      - name: step 1  
        uses: actions/checkout@v3  
  
      - name: step 2  
        id: step_2  
        run: |  
          command  
  
      - name: step 3  
        id: step_3  
        run: |  
          command
```

حالا بریم بخش های مختلف jobs رو با هم بررسی کنیم

محیط اجرا که با runs-on مشخص میشے

همونطور که احتمالا خودتون متوجه شدید اینجا مشخص میکنیم وظایف تو چه محیطی اجرا بشه ، محیط های اجرا متفاوت اما محدوده و لیست کاملش رو میتوانید با جستجو پیدا کنید اما من معمولا از 22 ubuntu استفاده میکنم چون استیبل و سریعه ، پس به صورت زیر میشه

runs-on: ubuntu-22.04

مراحل انجام Job ی بهش steps میگیم

میشه گفت اصل داستان همینجاست ، و جادوی اصلی تو step ها اتفاق میفته ، اینجاست که شما تک تک مراحل مورد نظرتون برای انجام رو نامگذاری و تعریف میکنید ، هر step آپشن های مختلفی داره که بعضی اشون ضروری هستند و بعضی دیگه نه ، مواردی مثل name که نامش رو مشخص میکنه id که یک شناسه ی یکتا بهش میده uses که مشخص میکنه از چه چیزی استفاده میکنید و run که اصل عملیات رو نشون میده

نگران نشید! 😊

ممکنه به خودتون بگید : ای بابا ، تا همین جاش کلی کار داشت ، من دیگه نمیکشم اما نگران نباشید ، شما قدم اصلی رو وقتی برداشتید که این مطلب به نظرتون جالب اوmd، اگه به اینجای آموزش رسیدید، پس این آموزش مخصوص خود شماست ، و به سادگی از پیش بر میاید، نفس عمیق بکشید تا مراحل بعدی رو با مثال پیش ببریم، و فقط لذت ببرید

برای راحتی بیشتر، من معمولاً از هر اسکریپت Bash که مینویسم، چند تا متغیر اضافه میکنم که بعداً ازشون تو workflow و حتی جاهای دیگه استفاده کنم مواردی مثل version و تاریخ آپدیت و....

بسته به زبان مورد نظر و سلیقه میتوانید ساختار خودتون رو داشته باشید

ابتداً پروژه‌ی خیالی ما با این شکله

```
#!/bin/bash
# Simple Bash script to show how github action work
AUTHOR="ALIREZA.ZOLFAGHAR"
VERSION="2.9.0"
VERSION_UPDATED="2025-02-21"
UPLOAD_TO_ALIREZAZ=true
```

طراحی step ها

همونطور که همه میدونیم ، بهترین نوع یادگیری و تمرین ، اونیه که در حال انجام کار انجام میشه پس به جای توضیحات در مورد آپشن‌ها و شرایطشون ، باید شیرجه بزنیم تو ش و در عمل step‌های مختلف پروژه‌ی خیالمون رو بنویسیم

دریافت محتوای ریپازیتوری تو محیط رانر

لازمه ما به تمام محتوای push شده تو محیط runner که همون 22 ubuntu هست دسترسی داشته باشیم ، پس اولین step رو اینجوری تنظیم میکنیم تا محتوای ریپو رو بیاریم داخل رانر

steps:

- name: Checkout repository
- uses: actions/checkout@v3

در واقع ما داریم برای این کار از یک action آماده به نام checkout استفاده میکنیم که کارش همینه و دیگه نیازی نیست خودمون از اول بنویسیم مثل

اکشن های آماده بی زیادی وجود داره که شما میتوانید سر فرصت تو شون بچرخید و باهاشون کارهای خودتون رو ساده تر انجام بدید ، برای دیدن لیست اکشن های آماده لینک زیر شروع خوبیه:

<https://github.com/marketplace?type=actions>

خواندن اطلاعات از Code

تو این پروژه خیالی لازمه ما سه تا چیز رو از code بخونیم : نسخه
ی برنامه ، تاریخ آپدیت ، و متغیر آپلود، این کار رو با دستورات
لینوکسی انجام میدیم

```
- name: Extract version number
  id: extract_version
  run: |
    VERSION=$( grep VERSION= simple-script.sh | head -n 1 | awk -F '=' '{print $2}' )
    echo "VERSION=$VERSION" >> $GITHUB_ENV
    UPDATE=$( grep VERSION_UPDATED= simple-script.sh | head -n 1 | awk -F '=' '{print $2}' )
    echo "UPDATE=$UPDATE" >> $GITHUB_ENV
    UPLOAD=$( grep UPLOAD_TO_ALIREZAZ simple-script.sh | head -n 1 | awk -F '=' '{print $2}' )
    echo "UPLOAD=$UPLOAD" >> $GITHUB_ENV
```

سه متغیر VERSION، UPDATE و UPLOAD تعريف کردیم و با دستورات ساده بی لینوکسی مثل grep و awk و مقدار این متغیر ها رو از کد اصلی اسکریپت که فایل simple-script.sh است باشه دریافت کردیم و آخرش این متغیر ها رو به GITHUB_ENV فرستادیم تا به عنوان متغیر محیطی گیت هاب شناخته بشند و در ادامه workflow ازشون برای اهدافمون استفاده کنیم

درباره اطلاعات کامیت

برای خوانایی بیشتر، من معمولاً عنوان و توضیحات Commit را هم از خود گیت هاب میگیرم تا به صفحه `release` اضافه کنم، اما این مورد ضروری نیست و شما میتوانید ازش بگذرید

```
- name: Get Commit Details
  id: commit_details
  run: |
    COMMIT_TITLE=$(git log -1 --pretty=%s)
    COMMIT_BODY=$(git log -1 --pretty=%b)
    echo "COMMIT_TITLE=$COMMIT_TITLE" >> $GITHUB_ENV
    echo "COMMIT_BODY=$COMMIT_BODY" >> $GITHUB_ENV
```

عنوان و توضیحات commit را از خود گیت هاب میگیریم و مثل دفعه‌ی قبل به متغیرهای محیطی گیت هاب میفرستیم تا بعداً بتونیم ازش استفاده کنیم

به این موضوع دقیق کنید که ممکنه کاراکترهای خاصی تو عنوان و توضیحات commit شما باشه که این بخش نتونه به درستی عمل کنه، پس یا باید از یه شیوه‌ی استانداردی برای عنوان و توضیحات کامیت استفاده کنید یا باید شیوه استخراج این دو مورد رو تغییر بدید و چیزی مناسب با عادت خودتون رو استفاده کنید

اما در حالت کلی، چیزی که اینجا برآتون نوشتم کار میکنه و در اکثر موارد لازم نیست تغییرش بدید مگر اینکه بخواید اطلاعات جذابتری رو هم از commit استخراج کنید و تو `release` ثبت کنید

ساخت release notes

ما میخوایم تمام این اطلاعاتی که تو مراحل قبلی دریافت کردیم رو
جایی به یادگار بگذاریم 😊 تا آیندگان ، وقتی به این پروژه رجوع
میکنند ، یه نسخه‌ی کامل از فایل‌های اون نسخه از پروژه داشته
باشند ، دوما شماره‌ی نسخه و توضیحات مربوط به commit این
نسخه رو ببینند، برای این هدف ، چی بهتر از صفحه‌ی
وجود داره؟

پس ما تمام این موارد رو تو صفحه‌ی ریلیز برای هر ورژن منتشر
میکنیم، و برای این توضیحات release note میسازیم ، که شکل
کلی step به این صورت میشه

```
- name: Create release notes
  id: generate_notes
  run: |
    echo "# Version: $VERSION" > release_notes.md
    echo "# Updated: $UPDATE" >> release_notes.md
    echo "# Upload : $UPLOAD " >> release_notes.md
    echo "If true, it will upload to External Server" >> release_notes.md
    echo "" >> release_notes.md
    echo "## Commit Details" >> release_notes.md
    echo "**Title:** $COMMIT_TITLE" >> release_notes.md
    echo "**Description:** $COMMIT_BODY" >> release_notes.md
    echo "" >> release_notes.md
    echo "## Installation" >> release_notes.md
    echo "curl -k https://github.com/${{ github.repository }}/releases/\`>
download/$VERSION/simple-script.sh | bash" >> release_notes.md
    echo "" >> release_notes.md
```

همونطور که مشخصه ما او مدیم با چند تا دستور **echo** محتوای
مورد نظرمون به فایلی با نام **release_notes.md** اضافه کردیم و
تو مرحله پایانه به گیت‌هاب میگیم از این فایل به عنوان توضیحات
ریلیز استفاده کنه ، تمام متغیرها رو تو مراحل قبل استخراج کردیم
و به متغیر محیطی گیت‌هاب دادیم تا همینجا ازش استفاده کنیم
 فقط ممکنه یکی از خطوط به نظر گیج کننده باشه :

```
curl -k https://github.com/${{ github.repository }}/releases/download/$VERSION/simple-script.sh
```

ما این خط رو اضافه کردیم که توضیحات هر نسخه یه آموزش ساده‌ی نصب مخصوص هر نسخه رو هم داشته باشه و به همین خاطر لینک دانلود رو به صورت داینامیک ساختیم گیت‌هاب خودش آدرس ریپو رو جایگزین متغیر استخراج کرده رو هم جایگزین **github.repository** میکنه و شماره‌ی نسخه ای که از اسکریپت استخراج کرده رو از **VERSION** میکنه تا لینک هر ورژن به صورت داینامیک و مخصوص همون ورژن ساخته بشه اگه برآتون سواله که سایر قسمت‌های لینک رو از کجا آوردیم ، باید بگم این آدرسی هست که گیت‌هاب به صورت پیشفرض به هر ریلیز شما میده ، کافیه `tag` یه داشته باشه ، که اینجا تگ ما همون `version` هست که از داخل `code` استخراج شده و باید دقیق کنید تو این مثال ، ما آدرس فایل پروژه رو مستقیم از توی ریپو ندادیم ، چون در اون صورت همیشه به آخرین نسخه‌ی فایل اشاره می‌کرد ، در حالی که ما میخواستیم به نسخه‌ی خاصی از فایل اشاره کنیم و به یادگار بگذاریم

ساخت یک فایل Zip از تمام ریپازیتوری ✓

پروژه‌ی خیالی ما، یه اسکریپت ساده‌ی `bash` یه فایل `pdf` داره که همین آموزش هست و از همه مهم‌تر یه فایل `release` که `release` هست و ما هر سه فایل رو به `release` اضافه میکنیم چون میخوایم بعداً به اونها لینک مستقیم داشته باشیم ، اما تو یه پروژه‌ی واقعی ما فایل و دایرکتوری‌های دیگه ای داریم که لینک مستقیم لازم ندارند اما نیاز داریم تا به عنوان نسخه پشتیبان از اونها نگهداری کنیم، پس برای اینکه بکاپ کامل باشه ، ما از تمام محتوای ریپازیتوری هم یک فایل `zip` می‌سازیم و به ریلیز اضافه می‌کنیم

```
- name: Create simple-github-action.zip
run: |
  zip -r simple-github-action.zip . -x ".git/*"
```

این فایل میتونه هر اسمی داشته باشه و با یک دستور لینوکسی ساده ساخته میشه ، اینجا من zip رو انتخاب کردم چون اطمینان داشتم تو محیط رانر به صورت پیشفرض نصب هست اما شما میتوانید هر پسوند دیگه ای مثل `tar | tar.gz | tar.zst` و ... انتخاب کنید ، فقط باید مطمئن بشید پیش نیاز هاش روی رانر نصبه و اگه نصب نبود از قبل با یه دستور apt نصبش کنید موضوع بعدی اون سوئیچ X هست که شاید برآتون سوال ایجاد کنه

باید بگم اون رو گذاشت تا بدونید محدودیتی برای دستورات نداریم و هر کاری تو یک لینوکس عادی انجام میدیم ، اینجا هم قابل انجامه ، با این آپشن من مشخص کردم یک نسخه از همه چیز داخل فایل zip قرار بگیره به غیر از محتوای پوشه `i`.

ساخت و انتشار release

تا اینجا ما هر اطلاعاتی می خواستیم رو از پروژمون استخراج کردیم، و حتی برآش release note ساختیم و دیگه همه چیز آماده است تا موشکی که ساختیم رو به هوا بفرستیم ، برای ساخت ریلیز میایم از یکی دیگه از اکشن های آماده `i` گیت هاب به نام **action-gh-release** استفاده میکنیم که از ما عنوان ریلیز، tag و فایل های اون رو میگیره و منتشرش میکنه فایل های که داریم اینها بودند

دو فایل از ریپازیتوری

- **simple-script.sh**
- **simple-github-action.pdf**

یک فایل zip که تو ریپازیتوری نیست و ما هر بار خودمون میسازیمیش:

- **simple-github-action.zip**

و یک فایل release notes که اون رو هم هر بار میسازیم تا توضیحات تو ش باشه

- **release_notes.md**

شکل این مرحله اینطوری میشه:

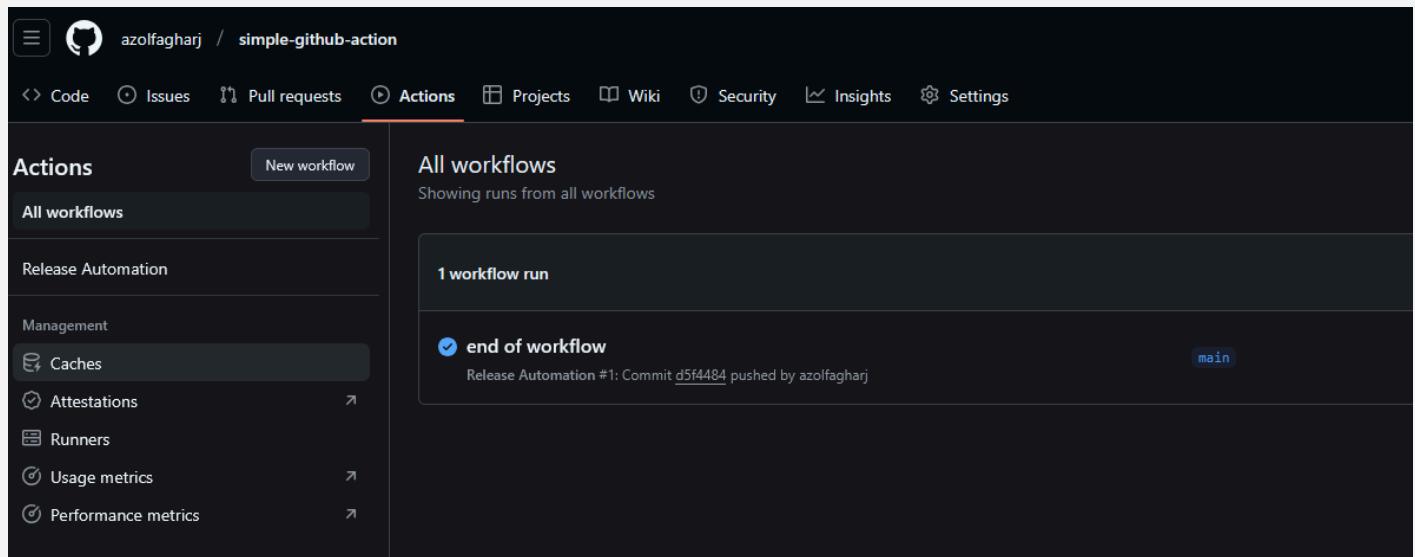
```
- name: Create release
  uses: softprops/action-gh-release@v1
  with:
    name: ${{ env.VERSION }}
    tag_name: ${{ env.VERSION }}
    body_path: ./release_notes.md
    files: |
      simple-script.sh
      simple-github-action.pdf
      simple-github-action.zip
  env:
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

- ما VERSION رو که تو مراحل قبل از code استخراج کردیم هم به جای نام release معرفی میکنیم و هم به جای tag فایل.md که تو مراحل قبل ساختیم رو هم به عنوان متن ریلیز معرفی میکنیم
- سه فایل دیگه رو هم به عنوان فایل هایی که باید تو release قرار داده بشند اعلام می کنیم

بخش مهم از این step env قسمت هست :
تو این workflow شما مثل یک کاربر دارید با گیت هاب تعامل میکنید ، تمام این کارها رو میتوانستید به صورت دستی انجام بدید و در نهایت به صفحه release از ریپازیتوری برید و عنوان و توضیحات رو بنویسید و فایل ها رو ضمیمه کنید و در نهایت منتشرش کنید ، حالا که قراره workflow به جای شما انجامش بده ، لازمه دسترسی انجامش رو داشته باشه ، پس با این تیکه از code شما به گیت هاب می فهمونید که خودکار دسترسی لازم رو اعطاء کنه

تمومه ! بريم تست کنيم

کار تمومه ، الان فقط کافيه يك push جديد روی اين رipiازيتوري بفرستيم و بشينيم نگاه کنيم که چطور Github Action به جاي ما بقيه ي کارها رو انجام مиде و لذت بريم اگه همه چيز درست باشه وقتی تو Github به صفحه ي رipiازيتوري بريid و سربرگ Action رو باز کنيد باید چيزی مثل تصویر صفحه ي بعد رو ببینيد



چرا راه دور بريم؟
ميتوسي تو لينك زير ببینيش

<https://github.com/azolfagharj/simple-github-action/actions/runs/13459803522/job/37612142096>

🏁 يه اجرای workflow با عنوان آخرین push شما ايجاد شده که تيک آبي داره ، يعني همه چيز عاليه و نتيجه ي کار تو صفحه ي release پروژه داره چشمك ميزنه <https://github.com/azolfagharj/simple-github-action/releases>

اگه يه ضربدر قرمز ديديد نگران نشيد!

آزمایش و رفع مشکلات هم بخشي از فرایند شماست ، پس اگه مشکلي وجود داشت کافيه همینجا رو workflow کليک کنيد تا جزئيات و خطاهای احتمالي ببینيد و رفع کنيد



هورا !!!

شما تونستید و الان یک Workflow کار راه بندار دارید
تا همینجا ش عالیه



اما اگه کلی ایده داری که دارند مغزت رو قلقلک می دند ، و احساس میکنی یه چیزهایی جا افتاده میتونی یه نگاه به آخرین صفحه بنداری :

- ★ امیدوارم این مطلب برات مفید بوده باشه
- ★ این تمام راه نبود ! اما قدم اول بود تا بتونی بخشنی از زحمت های تکراری هر روزت رو با خودکار سازی کم کنی.
- ★ از اینجا به بعد دیگه میتونی با توجه به نیازهات و خلاقیت فوق العادهات که میدونم کم هم نیست کارهای جالب تر و پیچیده تری بکنی

شاید به کار بیاد



This Repository:

<https://github.com/azolfaghari/simple-github-action>

Github Action Marketplace:

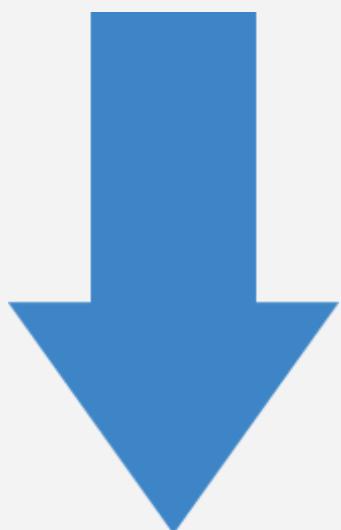
<https://github.com/marketplace?type=actions>

My GitHub:

<https://github.com/azolfaghari>

My Linkedin:

<https://www.linkedin.com/in/azolfaghari/>



ایول ! 🔥

میبینم که گربه‌ی شرودینگر زنده بود و او مدعی اینجا
دم شما گرم...

یا متوجه شدی یه چیزی کم بود
یا میخوای بیشتر بدونی

در هر دو حالت، نمیدارم دست خالی بری

چی کم بود ?

اون بالا ما یه متغیر boolean داشتیم به نام UPLOAD که داخل workflow حتی مقدارش رو هم گرفتیم اما ازش استفاده نکردیم

خب که چی ؟ 🤔

خیلی وقت‌ها تو پروژه ما code رو تغییر میدیم، همه چیز هم درست کار میکنه و مشکل و باگی وجود نداره، اما به دلیلی نمیخوایم فعلاً اون تغییرات تو محیط پروداکشن عملیاتی بشه اینجاست که اون متغیر به کارمون میاد

مثلثاً فرض کنید، تو همین پروژه‌ی کوچیک خودمون، من یه نسخه از پروژه روی یک سرور دیگه دارم و لینک ثابتی داره که با curl کردن اون لینک، بارها و بارها اسکریپت رو روی سرورهای دیگه دانلود و اجرا میکنم پس نمیخواهم تا زمانی که شرایط فراهم نشده اسکریپتی که به اون لینک وصله تغییر کنه

ولی در عین حال میخواham نسخه‌های جدید بدم و هر زمان لازم بود این workflow بتونه فایل روی اون سرور رو هم به روز کنه و مجبور نباشم این کار رو دستی انجام بدم پس هر زمان که نمیخواستم فایلی که روی سرور پروداکشن هست تغییر کنه code رو اینجوری می‌نویسم

```
UPLOAD_TO_ALIREZAZ=false
```

و هر زمان که خواستم فایل روی سرور پروداکشن تغییر کنه code رو اینجوری می‌نویسم

```
UPLOAD_TO_ALIREZAZ=true
```

چه طور فایل‌ها رو به سرور خارجی بفرستیم

یه نگاهی به فایل release.pro که تو ریپازیتوری پروژه برات گذاشتم بنداز، به آخر فایل برو

<https://github.com/azolfaghari/simple-github-action/blob/main/.github/workflows/release.pro>

همونطور که میبینی یه بخش اضافه به صورت زیر هست

```
- name: Install sshpass
  if: env.UPLOAD == 'true'
  run: sudo apt-get install -y sshpass

- name: Upload files to external server
  if: env.UPLOAD == 'true'
  run:
    export SSHPASS="${{ secrets.SSH_PASS }}"
    sshpass -e rsync -avz --delete -e "ssh -p ${{ secrets.PORT }} -o StrictHostKeyChecking=no" \
      simple-script.sh \
      simple-github-action.pdf \
      simple-github-action.zip \
      ${{ secrets.SSH_USER }}@${{ secrets.SERVER_IP }}:${{ secrets.REMOTE_PATH }}/
```

اینجا دو تا step اضافی هست که اول هر کدوم با if متغیر مربوط به Upload بررسی میشه و اگه true بود step رو انجام میده

نصب پکیج مورد نیاز

قصد ما اینه که فایل‌ها رو با دستور rsync روی یک سرور آپلود کنیم پس به password او نیاز داریم و میخوایم این رمز رو داخل دستور rsync بذاریم

برای این کار، روش استاندارد اینه که از sshpass استفاده کنیم پس خیلی ساده مثل یه اوبونتو لینوکس معمولی با استفاده از apt نصبش میکنیم

انتقال فایل‌ها

انتقال رو با استفاده از rsync انجام میدیم، یکی از پکیج‌های معروف و کار راه بنداز لینوکسیه که حتما باهاش کار کردیم و اگر هم کار نکردید کافیه یه جستجوی ساده بکنید تا متوجه بشید چه طور میشه باهاش فایل‌ها رو جابه‌جا کرد

این بخش از کار توضیحی نمیخواد اما موضوع شیوه‌ی استفاده از اطلاعات حساس، زمان rsync هست که برای توضیح میدم

مفهوم GitHub Secret

اگه به آخر دقت کرده باشی متوجه شدی که داره از پنج متغیر عجیب استفاده میکنه که ما تو workflow تعریفش نکردیم

- secrets.SSH_PASS
- secrets.PORT
- secrets.SSH_USER
- secrets.SERVER_IP
- secrets.REMOTE_PATH

حتما میتونید حدس بزنید تو هر کدام از متغیرها چه چیزی ذخیره شده، آدرس IP، پورت ssh، نام کاربری، رمز عبور سرور مقصد اینها همون اطلاعاتی هستند که وقتی به صورت دستی از دستور rsync استفاده میکنیم باید وارد کنیم تو هر ریپازیتوری گیت‌هاب اگه به بخش زیر برید

Settings > Secrets and Variables > Actions > Repository secrets

میتوانید برای ریپازیتوری از این متغیرها تعریف کنید که secrets صداشون میکنیم این بخش مخصوص متغیرهای مخفیانه ای هست که گیت‌هاب اکشن میتوانه از اونها استفاده کنه و به غیر از شما شخص دیگه ای نمیتوانه از مقدارشون آگاه بشه مثل هر متغیر دیگری، یک نام دارند و یک مقدار، متغیرها رو ایجا تعریف میکنیم و بعد تو workflow از اونها استفاده میکنیم تا از چشمان پاک و ناپاک اغيار مخفی بمونند 😊

تمام مولکولهای کائنات با تو یار باشند 🏆

خسته نباشی قهرمان، دمت گرم که تا اینجا او مددی اگه انتقاد و پیشنهاد یا سوالی داشتی، خودت میدونی کجا پیدام کنی.

میخوای یه پروژه‌ی جذاب رو تست کنی؟
یه نگاهی به پروژه‌ی زیر بنداز

<https://github.com/azolfagharj/DloadBox>