



TUTORIAL -- A (VERY) QUICK INTRODUCTION TO GITHUB ACTIONS

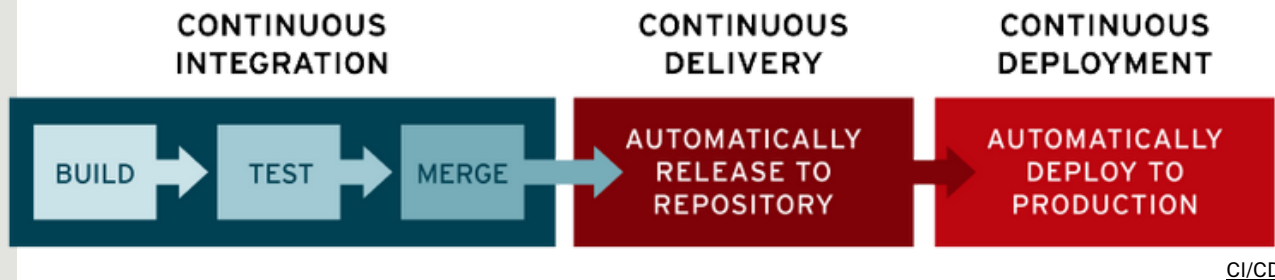
HPC TRAINING EVENTS

DR. OSCAR FABIAN MOJICA LADINO

oscar.ladino@fieb.org.br

WHAT ARE GITHUB ACTIONS?

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline.



Automatic

- Style Checking
- Testing
- Coverage Report Generation
- Documentation Building
- Package Building
- Publishing (to PiPy)

Why having a GitHub-native CI/CD tool is helpful?

The most basic answer is simplicity—if you're already hosting a project on GitHub, you have a built-in CI/CD tool that works right alongside your code.

Who can use GitHub Actions?

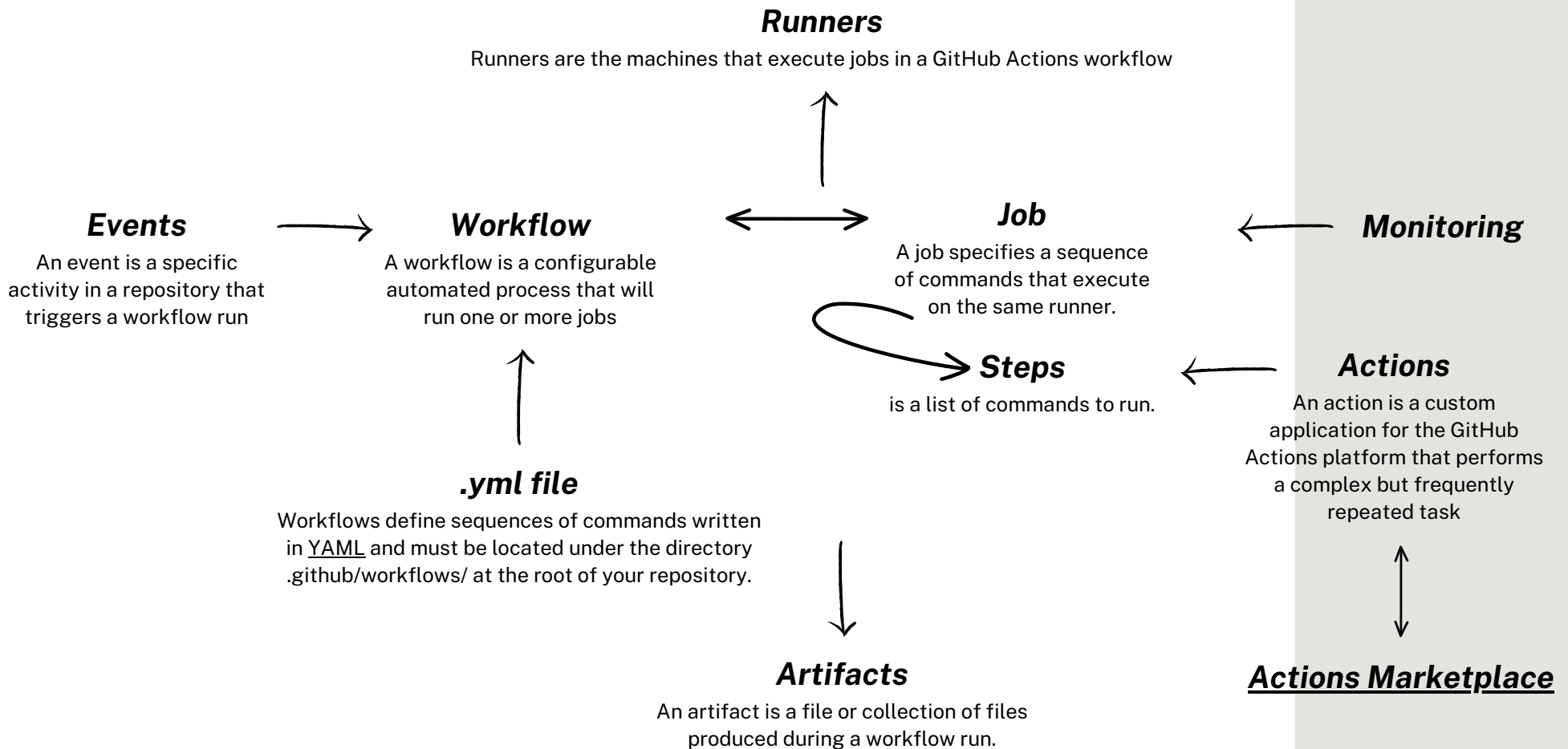
GitHub Actions is free and available for use on any public repository and self-hosted runner.

A GitHub Free account comes with 2,000 minutes a month that can be used on private repositories (developers have unlimited use of GitHub Actions on a public repository with a self-hosted runner)

WHAT MORE ARE GITHUB ACTIONS?

A platform to run any (not too complex) workflow in a virtual environment and integrate with GitHub.

Github Actions platform



RUNNERS: VIRTUAL COMPUTING ENVIRONMENT

- **Github-Hosted Runners (Free!)**
 - Ubuntu
 - MacOS
 - Windows Server
- **Large runners: Github Enterprise Cloud**
- **Self-Hosted Runners**

Virtual Machine	Processor (CPU)	Memory (RAM)	Storage (SSD)	OS (YAML workflow label)	Notes
Linux	2	7 GB	14 GB	ubuntu-latest, ubuntu-22.04, ubuntu-20.04	The ubuntu-latest label currently uses the Ubuntu 22.04 runner image.
Windows	2	7 GB	14 GB	windows-latest, windows-2022, windows-2019	The windows-latest label currently uses the Windows 2022 runner image.
macOS	3	14 GB	14 GB	macos-latest, macos-12, macos-11	The macos-latest workflow label currently uses the macOS 12 runner image.
macOS	4	14 GB	14 GB	macos-13 [Beta]	N/A

[GitHub-hosted runners](#)

HELLO WORLD

This example workflow prints “Hello world”, followed by “Step 1...”, “Step 2...”, “Step 3...”, and finally “Goodbye”.

```
name: hello-world-example
on:
  push:
jobs:
  say-hello:
    runs-on: ubuntu-latest
    steps:
      - name: Say Hello
        run: echo "Hello world!"
      - name: Do stuff
        run: |
          echo "Step 1..."
          echo "Step 2..."
          echo "Step 3..."
      - name: Say Goodbye
        run: echo "Goodbye!"
```

- We must specify what events trigger the workflow to run. In this case we run it every time someone pushes to the repo.
- A job specifies a sequence of commands. We named this job say-hello.
- jobs can run on different machines.
- steps is a list of commands to run.
- Finally, do stuff! Run a command using the operating system's shell.
- Use the pipe, |, to start a multiline string in yaml. This allows us to write easily readable multistep code

Another fairly simple example

TRIGGER EVENTS

- The **on** section defines what events trigger the workflow. For an exhaustive list of events and their types, [see here](#).
- Trigger on **push** to the remote repo
 - Optionally include **branches** to trigger the workflow only on branches that match the given patterns.
 - A ***** matches zero or more characters, but not the **/** character. E.g. matches refs/heads/foo/abc but not refs/heads/foo/abc/123.
 - A ****** matches zero or more of any character E.g. matches refs/heads/foo/abc and refs/heads/foo/abc/123.
 - **!** negates a match. E.g. excludes refs/heads/foo/abc/456. More on negative patterns [here](#).
 - Optionally include **tags** to trigger the workflow only on tags that match the given patterns
- Trigger on pull requests
 - Pull requests into develop branch
- Trigger on a schedule, specified with POSIX cron syntax

*	*	*	*	*
minute	hour	day	month	day
(0-59)	(0-23)	of month	(1-12)	of week
		(1-31)		(0-6)

```

name: event-triggers-example
on:
  push:
    branches:
      - 'develop'
      - 'foo/*'
      - 'foo/**'
      - '!foo/*/456'
    tags:
      - '*'
  pull_request:
    branches:
      - 'develop'
  schedule:
    - cron: '0 0 * * *'
jobs:
  say-hello:
    runs-on: ubuntu-latest
    steps:
      -
        name: Event
        run: echo "Triggered by: $GITHUB_EVENT_NAME"
      -
        name: Say Hello
        run: echo "Hello world!"

```

From [GitHub Actions by Example](#)

This is a default environment variable present in all Actions.

ACTIONS

○○○

```
name: actions-example
on:
  push:
jobs:
  use-actions:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-python@v3
        with:
          python-version: "3.10"

      - name: Install repo dependencies
        run: |
          python -m pip install --upgrade pip
          pip install pytest
          if [ -f requirements.txt ]; then pip install -r requirements.txt; fi

      - name: Test with pytest
        run: |
          pytest
```

- **Actions** reduce workflow steps by providing reusable “code” for common tasks. To run an action, you include the **uses** keyword pointing to a GitHub repo with the pattern {owner}/{repo}@{ref} or {owner}/{repo}/{path}@{ref} if it’s in a subdirectory. A ref can be a branch, tag, or SHA. GitHub officially supports many common actions. See the full list [here](#)
- The **checkout** action checks out your repo into the working directory at the ref which triggered the workflow (e.g. a branch that was pushed).
- The **setup-python** action installs a version of Python or PyPy and (by default) adding it to the PATH
 - Some actions also have required or optional arguments.

JOB ORDERING & PARALLEL JOBS

○○○

```
name: parallel-jobs
on:
  push:
jobs:
  job1:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Doing work parallel with job2"
  job2:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Doing work parallel with job1"
  job3:
    runs-on: ubuntu-latest
    needs: job1
    steps:
      - run: echo "job1 done, running job3"
  job4:
    runs-on: ubuntu-latest
    needs: [job2, job3]
    steps:
      - run: echo "job2 & job3 done, running job4"
  job5:
    runs-on: ubuntu-latest
    if: ${{ always() }}
    needs: job1
    steps:
      - run: echo "job1 completed with status ${{ needs.job1.result }}, running job5"
```

Jobs are run in parallel by default. You can force job ordering by using the **needs** keyword.

- *job1 must complete successfully before starting job3*
- *job2 and job3 must complete successfully before starting job4*
- *This job will run after job1, even if job1 fails (see [Context Variables](#))*

JOB MATRIX

You can run multiple jobs with different configurations by using a job matrix. Note that jobs defined by a matrix run in parallel by default.

- The **matrix** keyword is how you define a job matrix. Each user-defined key is a matrix parameter, here we've defined two: `os` and `python-version`. The list of possible values for each parameter are used in a cartesian product to create jobs. This section defines a 3 x 2 matrix of 6 jobs, each with a different combination of `os` and `python-version`.
- The **exclude** keyword prevents jobs with specific configurations from running.
- **include** allows you to add new jobs to the matrix. Note that the include rules always evaluated after the exclude rules.
- Use matrix parameters to configure jobs. Context expressions are used to insert the parameter value. This example uses the `os` parameter to set the operating system of the job.
- Setup python based on the matrix parameter `python-version`

```

name: job-matrix
on:
  push:
jobs:
  my-job:
    strategy:
      matrix:
        os: [ubuntu-18.04, ubuntu-20.04, ubuntu-latest]
        python-version: [3.8, 3.x]
      exclude:
        - os: ubuntu-18.04
          python-version: 3.8
      include:
        - os: macos-latest
          python-version: 3.x
    runs-on: ${ matrix.os }
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python ${ matrix.python-version }
        uses: actions/setup-python@v2
        with:
          python-version: ${ matrix.python-version }
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          python -m pip install pytest numpy
          pip install -e .[compression]
      - name: Run examples
        run: |
          python examples/use_classic.py
          python examples/use_modernised.py
      - name: Test with pytest
        run: py.test -vs tests

```

OUTPUTS

Data can be shared between **jobs** and **steps**.

- Create outputs for a step by writing to stdout in the format of `::set-output name=<name>::<value>`. A step can have multiple outputs.
- Steps that create outputs must have unique ids.
- Use the steps context variable and step id to get the value
- Create outputs for a job which will be available to other jobs that needs it (see Job Ordering). You can include output from steps that ran for the job.
- Use context expressions to grab outputs from a job included in needs (see Job Ordering)

```
name: outputs
on:
  push:
jobs:
  job1:
    runs-on: ubuntu-latest
    steps:
      - name: Do Work
        run: |
          echo '::set-output name=FAV_NUMBER::3'
          echo '::set-output name=FAV_COLOR::blue'
        id: abc
      - name: Read output
        run: |
          echo "${{steps.abc.outputs.FAV_NUMBER}}"
          echo "${{steps.abc.outputs.FAV_COLOR}}"
    outputs:
      fav-animal: tiger
      fav-number: ${{steps.abc.outputs.FAV_NUMBER}}
  job2:
    runs-on: ubuntu-latest
    needs: job1
    steps:
      - run: |
          echo "${{needs.job1.outputs.fav-animal}}"
          echo "${{needs.job1.outputs.fav-number}}"
```

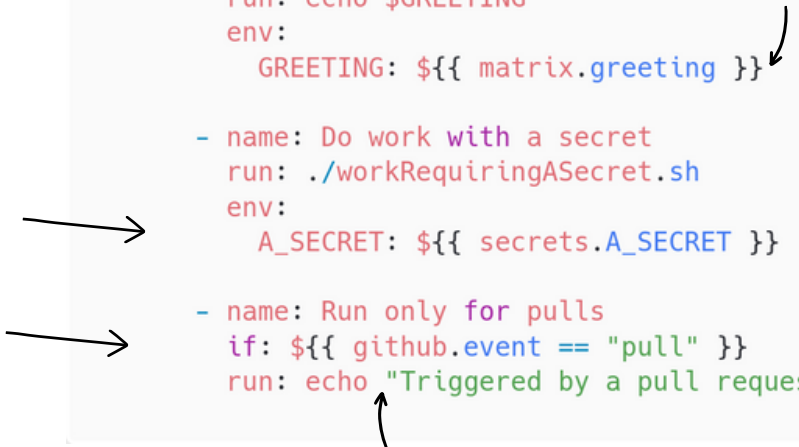
CONTEXT VARIABLES

Contexts are collections of variables that are accessible outside of the run commands. You can think of them as variables which can be templated into the workflow file itself. See the full list of contexts [here](#).

- Use the [matrix context](#) to print the job's greeting.
 - The matrix context contains the matrix properties defined in the workflow file that apply to the current job. In the example, we configured a matrix with the greeting key, the matrix context object includes the greeting properties with the values that are being used for the current job.
 - Property name: matrix.<property_name>
 - Type: String
 - Description: The value of a matrix property.
- Use a [secret from your repository](#) as an env var.
- Contexts can also be part of [expressions](#).

```
name: contexts-example
on:
  push:
  pull:
jobs:
  use-contexts:
    strategy:
      matrix:
        greeting: [Hello, Howdy, Hey]
    runs-on: ubuntu-latest
    steps:
      - name: Print greeting
        run: echo $GREETING
        env:
          GREETING: ${{ matrix.greeting }}
      - name: Do work with a secret
        run: ./workRequiringASecret.sh
        env:
          A_SECRET: ${{ secrets.A_SECRET }}
      - name: Run only for pulls
        if: ${{ github.event == "pull" }}
        run: echo "Triggered by a pull request"
```

Access contexts using the expression syntax `${{ <context> }}`



The github context contains information about the workflow run and the event that triggered the run.

CONTEXT EXPRESSIONS

As seen in last slide, you can inject variables into the workflow by using the `${{ <context variable> }}` syntax. Workflows support evaluating expressions as well such as comparisons and simple functions. For more information see the [docs](#).

- Use equality operators to get a boolean result; in this case we are only running the step when the greeting is “Hello”. Note the use of single quotes for the string literal. You can use `==`, `!=`, `<`, `<=`, `>`, `>=`, `&&`, `||`, and `(...)`. Learn more about operators [here](#)
- `toJSON()` converts things to a pretty-printed JSON string
- `fromJSON()` converts a string into a JSON object or value
- `success()` returns true if no previous steps have failed or have been canceled.
- `always()` returns true no matter if a step failed or the workflow was cancelled.
- `canceled()` returns true if the workflow was canceled.
- `failure()` returns true if any previous step failed.

```
name: expressions-example
on:
  push:
  pull_request:
jobs:
  use-expressions:
    strategy:
      matrix:
        greeting: [Hello, Howdy, Hey]
    runs-on: ubuntu-latest
    steps:
      - name: Print if 'Hello'
        if: ${{ matrix.greeting == 'Hello' }}
        run: echo "greeting is Hello"

      - name: Print if starts with 'He'
        if: ${{ startsWith(matrix.greeting, 'He') }}
        run: echo "greeting starts with He"

      - name: Print if ends with 'y'
        if: ${{ endsWith(matrix.greeting, 'y') }}
        run: echo "greeting ends with y"

      - name: Print if contains 'ow'
        if: ${{ contains(matrix.greeting, 'ow') }}
        run: echo "greeting contains ow"

      - name: Print formatted greeting
        run: |
          echo "${{ format('{0} says {1}', github.actor, matrix.greeting) }}"

      - name: To JSON
        run: echo 'Job context is ${{ toJSON(job) }}'

      - name: From JSON
        env: ${{ fromJSON('{"FAVORITE_FRUIT": "APPLE", "FAVORITE_COLOR": "BLUE"}') }}
        run: echo "I would like a ${FAVORITE_COLOR} ${FAVORITE_FRUIT}"

      - name: Success
        if: ${{ success() }}
        run: echo "Still running..."

      - name: Always
        if: ${{ always() }}
        run: echo "You will always see this"

      - name: Canceled
        if: ${{ cancelled() }}
        run: echo "You canceled the workflow"

      - name: Failure
        if: ${{ failure() }}
        run: echo "Something went wrong..."
```