Continuous Integration with GitHub Actions
Software Engineering for Scientists

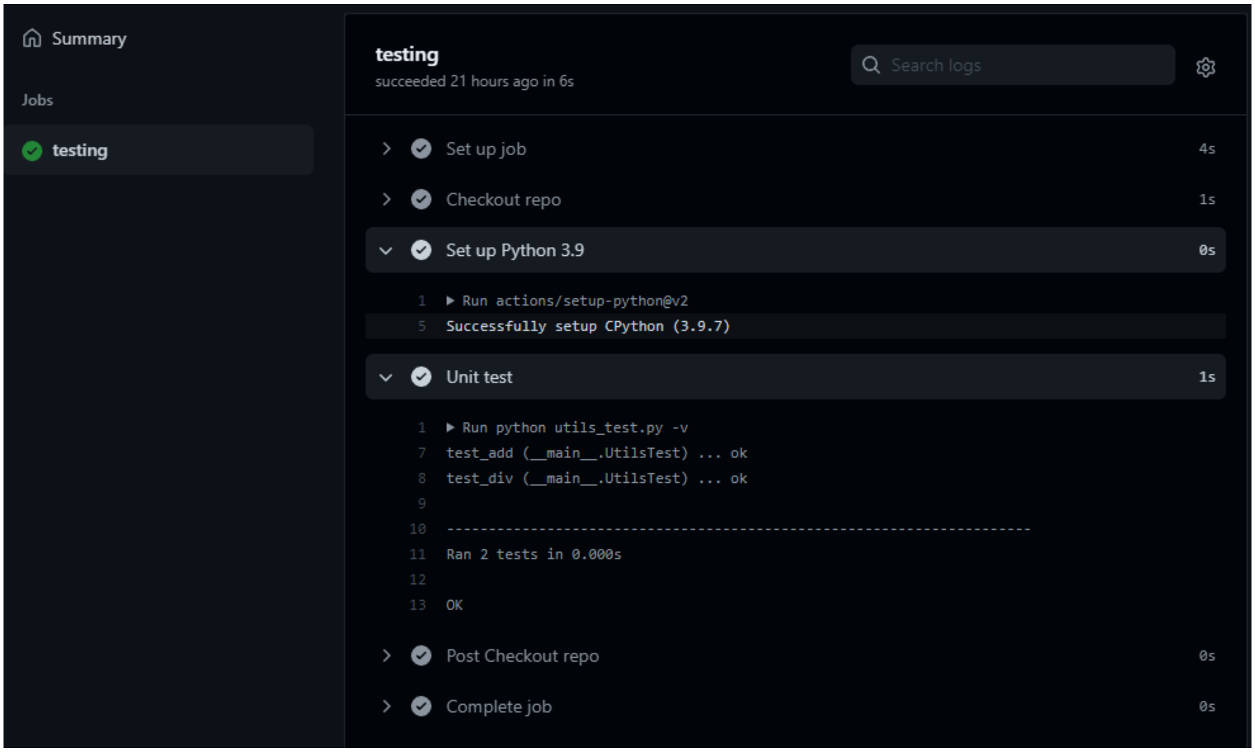
GitHub Actions is an interface for running continuous integration scripts (called "workflows") on your GitHub repositories. The following vocabulary describes the components of the workflow syntax:

- **Workflows**: An automated procedure that is added to the repository. A workflow is made up of one or more jobs and are triggered by an event.
- **Events**: A specific activity that triggers a workflow. These events (mostly) originate from GitHub, such as a push or pull-request.
- **Jobs**: A set of steps that is executed within the workflow. Jobs are run independent of one another. Jobs can be run in parallel (default) or in series. Each job has its own compute environment.
- **Steps**: The set of tasks that run within a job. A single step can be a single *Action* or a shell command. The steps in a single job share a common compute environment.
- **Actions**: *Actions* are standalone commands that are combined within steps to create a job. Actions are the smallest building block of a workflow. You can create your own actions or use those created by the GitHub community.

Now to present three different workflow examples and the results they produce when run on GitHub. The first demonstrates how to run a unit testing python script "`utils_test.py`" on an Ubuntu system, using Python version 3.9. Note the event that triggers the workflow is a git push to the main branch of the repo. In order to access the contents of the repo we use the `actions/checkout` Community Action, and to utilize the specific python version we use `actions/setup-python`.

```
name: Running unit tests

on:
  push:
    branch:
      - main

jobs:
  testing:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout repo
      uses: actions/checkout@v2
    - name: Set up Python 3.9
      uses: actions/setup-python@v2
      with:
        python-version: 3.9
    - name: Unit test
      run: python utils_test.py -v
```

Navigating to the “Actions” tab on the GitHub repo, we see the results of the above workflow. Here we see each of the action steps titled by the corresponding “name” keyword. The stdout from the unit test is displayed as if the script were run at the command line.

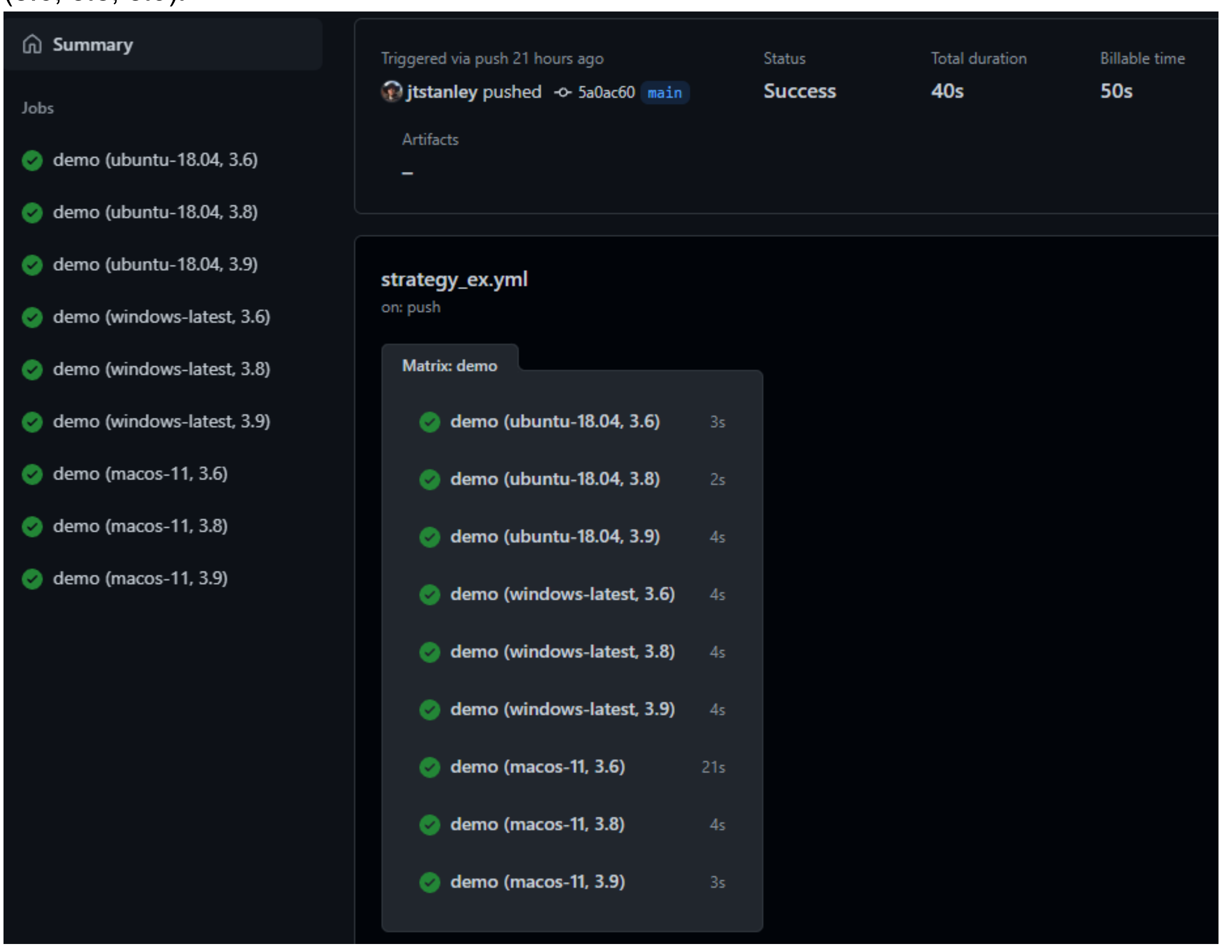


The second workflow demonstrates how to run the same job on an array (a.k.a. “matrix”) of different compute systems. This allows us to guarantee our software is robust to multiple platforms. Under the job name (“`demo`”) we use the `strategy` keyword to specify a list of operating systems (“`os`”) and a range of python versions (“`py`”). We can then run independent jobs for each pairwise combination of os and python version by using the “expressions” syntax for `runs-on` (referencing the os values via “`matrix.os`”) before the `steps` and `python-version` (referencing python via “`matrix.py`”) in the python setup action. The end of this workflow simply prints out the operating system and python version within the python environment (see the final `run` line) to confirm we are in fact running in the expected compute environment.

```
name: Strategy/matrix demo

on:
  push:
    branch: [main]

# This defines the scope of the job context

jobs:
  demo:
    strategy:
      matrix:
```

```
        os: [ubuntu-18.04, windows-latest, macos-11]
        py: [3.6, 3.8, 3.9]
    # This is an expression syntax
    runs-on: ${{ matrix.os }}
    steps:
    - name: Python setup
      uses: actions/setup-python@v2
      with:
        python-version: ${{ matrix.py }}
    - name: Print sys info
      shell: python
      run: |
        import sys
        import platform
        print(f"Operating system: {platform.system()}
({platform.release()})")
        print(f"Python version: {sys.version}")
```

In the results under the Actions tab on the GitHub repo, we see a separate job for each of the pairwise combinations of OS (ubuntu-18.04, windows-latest, and macos-11) and Python version (3.6, 3.8, 3.9).



Example of one of the details for one of the jobs:

Finally, here we see a workflow which installs all the packages within a conda environment YAML file (swefs.yml). The runners don't quite run conda in the same as CSEL or your personal computer. You can't create persistent environments to activate/deactivate. Instead, you can overwrite the base environment instead, with the contents of your desired conda environment. We do that with the following line:

- `conda env update –file swefs.yml –name base`

We then confirm the list of environments (should be just `base`) and then list the updated contents of that environment (`conda list –name base`). Finally, at the python command line, we import the `yaml` package and confirm it is the expected version.

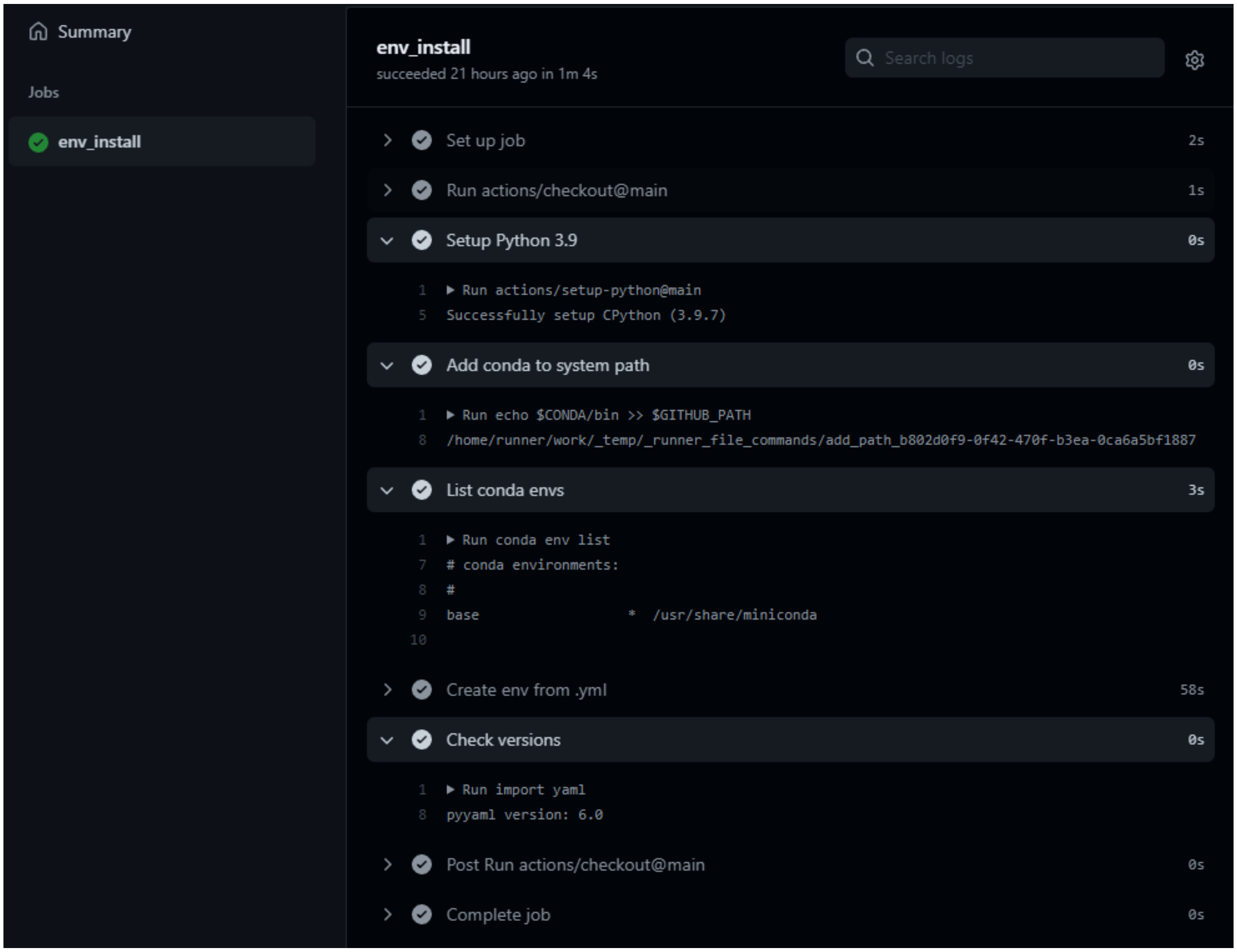
```
name: Test conda env

on:
  push:
    branches:
      - main

jobs:
  env_install:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@main
    - name: Setup Python 3.9
      uses: actions/setup-python@main
      with:
        python-version: 3.9
    - name: Add conda to system path
```

```
      run: |
        echo $CONDA/bin >> $GITHUB_PATH
        echo $GITHUB_PATH
    - name: List conda envs
      run: conda env list
    - name: Create env from .yml
      run: |
        conda env update --file swefs.yml --name base
        conda env list
        conda list --name base
    - name: Check versions
      shell: python
      run: |
        import yaml
        print(f"pyyaml version: {yaml.__version__}")
```

Conda environment YAML file, `swefs.yml`:

```
name: swefs
channels:
  - conda-forge
dependencies:
  - _libgcc_mutex=0.1=conda_forge
  - _openmp_mutex=4.5=1_gnu
  - ca-certificates=2021.10.8=ha878542_0
  - ld_impl_linux-64=2.36.1=hea4e1c9_2
  - libffi=3.3=h58526e2_2
  - libgcc-ng=11.1.0=hc902ee8_8
  - libgomp=11.1.0=hc902ee8_8
  - libstdcxx-ng=11.1.0=h56837e0_8
  - ncurses=6.2=h58526e2_4
  - openssl=1.1.1l=h7f98852_0
  - pip=21.2.4=pyhd8ed1ab_0
  - pycodestyle=2.7.0=pyhd8ed1ab_0
  - python=3.9.7=h49503c6_0_cpython
  - python_abi=3.9=2_cp39
  - pyyaml=6.0=py39h3811e60_0
  - readline=8.1=h46c0cb4_0
  - setuptools=58.0.4=py39hf3d152e_0
  - sqlite=3.36.0=h9cd32fc_1
  - tk=8.6.11=h27826a3_1
  - tzdata=2021a=he74cb21_1
  - wheel=0.37.0=pyhd8ed1ab_1
  - xz=5.2.5=h516909a_1
  - yaml=0.2.5=h516909a_0
  - zlib=1.2.11=h516909a_1010
prefix: /home/jovyan/.conda/envs/swefs
```

As expected, we see the updated base environment and confirm the yaml version, 6.0.


Summary
Jobs
env_install
env_install
succeeded 21 hours ago in 1m 4s
Search logs
Set up job
2s
Run actions/checkout@main
1s
Setup Python 3.9
0s
Run actions/setup-python@main
Successfully setup CPython (3.9.7)
Add conda to system path
0s
Run echo $CONDA/bin >> $GITHUB_PATH
/home/runner/work/_temp/_runner_file_commands/add_path_b802d0f9-0f42-470f-b3ea-0ca6a5bf1887
List conda envs
3s
Run conda env list
conda environments:
#
base * /usr/share/miniconda
Create env from .yml
58s
Check versions
0s
Run import yaml
pyyaml version: 6.0
Post Run actions/checkout@main
0s
Complete job
0s