Testing with GitHub Actions

Olaf Alders

CPAN: OALDERS

GitHub: @oalders

Twitter: @olafalders

Today's plan

- Olaf will cover GitHub Workflows and Actions
- Mark will show us how to do many of the same things with CircleCI

What are GitHub Actions and Workflows

- Both are components you can use as part of your Continuous Integration (CI) process

What is CI (Continuous Integration)?

- The practice of automating the integration of code changes into a software project.
- The CI process is comprised of automatic tools that assert the new code's correctness before integration.

Definitions loosely borrowed from https://www.atlassian.com/continuous-delivery/continuous-integration

Practical Examples of CI (push)

When code is pushed to a branch

- Run a test suite when new code changes are pushed.
- Check for code correctness by employing linters.
- Check for untidy code using tidiers.

Practical Examples of CI (merge)

When code has been merged

- Run all of the tests we run on a regular code push.
- Create a new Docker image and push it to Docker Hub.
- Send a notification to Slack or IRC.

Other Events Which Can Trigger CI in GitHub

- Pushing a tag
- Closing an issue
- Creating a pull request
- Editing a wiki page
- Forking a repository
- Adding a label to an issue
- Too many to list

See https://help.github.com/en/actions/reference/events-that-trigger-workflows

GitHub Actions vs GitHub Workflows

- These appear to be used interchangeably at times.
- Let's simplify these terms for our purposes today.

What are GitHub Actions?

- A product which allows you to create custom software development life cycle (SDLC) workflows directly in your GitHub repository.
- Individual tasks that you can combine to
    - create jobs
    - customize your workflow
- You can create your own actions, or use and customize actions shared by the GitHub community.

What are GitHub Workflows?

- A configurable automated process made up of one or more jobs.
- One or more YAML files which live in a `.github/workflows` dir.
- Can have descriptive names, eg: `.github/workflows/run-shellcheck.yml`
- `.yml` or `.yaml` extension required.

Actions vs Workflows Revisited

- For today's purposes, to use some Perl terminology
  - actions (re-useable bits) => modules
  - workflows (created per-repository, may include actions) => scripts

The Specs

Each virtual machine has the same hardware resources available.

- 2-core CPU
- 7 GB of RAM memory
- 14 GB of SSD disk space
- 20 concurrent jobs (5 on macOS) in the free tier

https://help.github.com/en/actions/reference/virtual-environments-for-github-hosted-runners

Available Runners

`runs-on` currently has the following possibilities:

Virtual environment	YAML workflow label
Windows Server 2019	`windows-latest` or `windows-2019`
Ubuntu 20.04	`ubuntu-20.04`
Ubuntu 18.04	`ubuntu-latest` or `ubuntu-18.04`
Ubuntu 16.04	`ubuntu-16.04`
macOS Catalina 10.15	`macos-latest` or `macos-10.15`

- `self-hosted` is also an option

Need Perl?

Only the Windows environments come with Perl pre-installed (5.30.2)

What Does a Workflow Look Like?

Let's start with a simple example, one which uses a single trigger.

Lint Your Shell Scripts

In a file called `.github/workflows/shellcheck.yml`

```yaml
on: push

name: Lint Scripts

jobs:
  shellcheck-job:
    name: Shellcheck
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@master
      - name: Lint Bash Scripts
        uses: ludeeus/action-shellcheck@master
```

Lint Your Shell Scripts

Add more conditions:

```yaml
on:
  push:
    branches:
      - "*"
  pull_request:
    branches:
      - "*"
  schedule:
    - cron: "15 4 * * 0" # Every Sunday morning EST
```

Lint Your Shell Scripts

Our final file:

```yaml
on:
  push:
    branches:
      - "*"
  pull_request:
    branches:
      - "*"
  schedule:
    - cron: "15 4 * * 0" # Every Sunday morning EST

name: Lint Scripts

jobs:
  shellcheck-job:
    name: Shellcheck
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@master
      - name: Lint Bash Scripts
        uses: ludeeus/action-shellcheck@master
```

Your Most Useful Action

`checkout`

```yaml
# You will us this one all the time
- name: Checkout
  uses: actions/checkout@v2

# Check out an additional repository, nested inside the first
- name: Checkout tools repo
  uses: actions/checkout@v2
  with:
    repository: my-org/my-tools
    path: my-tools
```

This action is highly configurable. Be sure to check out the documentation.

Invoking Actions

- `actions/checkout@v1` (major release tag)
- `actions/checkout@v1.3` (patch release tag)
- `actions/checkout@master` (named branch)
- `actions/checkout@4d93e0a8b53294e211fae35952eb233ded535037` (SHA)

Some Awesome Actions

https://github.com/sdras/awesome-actions

- Big list. Not sure how curated it is.
- Good starting point.

Let's revisit our first example

```yaml
on: push

name: Lint Scripts

jobs:
  shellcheck-job:
    name: Shellcheck
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@master
      - name: Lint Bash Scripts
        uses: ludeeus/action-shellcheck@master
```

What if we want to test some of our Bash scripts? We could use `bats` for that.

Using `run` steps

We could look for an action that provides `bats`, but in this case we want to demonstrate how to run arbitrary code.

```
steps:
  - uses: actions/checkout@master
  - name: Lint Bash Scripts
    uses: ludeeus/action-shellcheck@master
  - name: Install bats via npm
    run: npm install -g bats # assumes npm is available
```

Recovering from failure

```
steps:
  - uses: actions/checkout@master
  - name: Lint Bash Scripts
    uses: ludeeus/action-shellcheck@master
  - name: Install bats via npm
    run: npm install -g bats # assumes npm is available
  - name: use backup plan
    run: >
      cd /tmp &&
      git clone https://github.com/bats-core/bats-core.git --depth 1 &&
      cd bats-core &&
      ./install.sh /usr/local
    if: ${{ failure() }}
```

You may check for `success()`, `failure()`, `cancelled()` and `always()`.

Finally, let's run our tests

```
steps:
  - uses: actions/checkout@master
  - name: Lint Bash Scripts
    uses: ludeeus/action-shellcheck@master
  - name: Install bats via npm
    run: npm install -g bats # assumes npm is available
  - name: use backup plan
    run: >
      cd /tmp &&
      git clone https://github.com/bats-core/bats-core.git --depth 1 &&
      cd bats-core &&
      ./install.sh /usr/local
    if: ${{ failure() }}
  - name: Run tests
    run: bats -t t/foo.bats
```

The entire config file

```
on: push
jobs:
  test-job:
    name: I test shell scripts
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@master
      - name: Lint Bash Scripts
        uses: ludeeus/action-shellcheck@master
      - name: Install Bats
        run: npm install -g bats # assumes npm is available
      - name: Use backup install plan
        run: >
          cd /tmp &&
          git clone https://github.com/bats-core/bats-core.git --depth 1 &&
          cd bats-core &&
          ./install.sh /usr/local
        if: ${{ failure() }}
      - name: Run tests
        run: bats --tap t
```

Using Docker Images

Q: What if want to ensure that we have `npm`?

A: Use a Docker image

Why not use a Docker images for `bats`? The last official Docker image with `bats` is 2 years old. We want something more recent.

```
container:
  image: node:14.4.0-buster-slim
steps:
  ...
```

The entire worfkow file

```yaml
on: push
jobs:
  test-job:
    name: I test shell scripts
    runs-on: ubuntu-latest
    container:
      image: node:14.4.0-buster-slim
    steps:
      - uses: actions/checkout@master
      - name: Lint Bash Scripts
        uses: ludeeus/action-shellcheck@master
      - name: Install Bats
        run: npm install -g bats # assumes npm is available
      - name: Use backup install plan
        run: >
          cd /tmp &&
          git clone https://github.com/bats-core/bats-core.git --depth 1 &&
          cd bats-core &&
          ./install.sh /usr/local
        if: ${{ failure() }}
      - name: Run tests
        run: bats --tap t
```

See https://github.com/oalders/ci-example-bats

Poking around in your Docker container

Problem: how do we know that the Docker image we choose has both `npm` and modern version of `bash`?

```
docker run -it --rm node:14.4.0-buster-slim /bin/bash

root@187063816c88:/# bash --version
GNU bash, version 5.0.3(1)-release (x86_64-pc-linux-gnu)

root@187063816c88:/# npm install -g bats
/usr/local/bin/bats -> /usr/local/lib/node_modules/bats/bin/bats
+ bats@1.1.0
added 1 package from 1 contributor in 1.358s
```

`docker run` flags:

- `-i` : Keep STDIN open even if not attached
- `-t` : Allocate a pseudo-tty
- `--rm` : Automatically remove the container when it exits

Running Your Workflows Locally

Was that too tedious? Maybe you just want to run your actual workflow without pushing to GitHub.

https://github.com/nektos/act

Getting started is as simple as installing `act` and then running the `act` command, without arguments from the top level of your repository.

"Think globally, `act` locally"

```
$ act
[bats.yml/I test shell scripts] 🚀  Start image=node:14.4.0-buster-slim
[bats.yml/I test shell scripts]   🐳  docker run image=node:14.4.0-buster-slim entrypoint=["/usr/bin/tail" "-f" "/dev/null"] cmd=[]
[bats.yml/I test shell scripts]   🐳  docker cp src=/Users/olaf/Documents/github/ci-example-bats/. dst=/github/workspace
[bats.yml/I test shell scripts] ⭐  Run actions/checkout@master
[bats.yml/I test shell scripts]   ✅  Success - actions/checkout@master
[bats.yml/I test shell scripts] ⭐  Run Lint Bash Scripts
[bats.yml/I test shell scripts]   ☁  git clone 'https://github.com/ludeeus/action-shellcheck' # ref=master
[bats.yml/I test shell scripts]   🐳  docker build -t act-ludeeus-action-shellcheck-master:latest /Users/olaf/.cache/act/ludeeus-action-shellcheck@master
[bats.yml/I test shell scripts]   🐳  docker run image=act-ludeeus-action-shellcheck-master:latest entrypoint=[] cmd=[]
| ./bin/date.sh
[bats.yml/I test shell scripts]   🚧  ::warning:: programs in PATH should not have a filename suffix
[bats.yml/I test shell scripts]   💬  ::debug:: Checking ./bin/date.sh
[bats.yml/I test shell scripts]   ✅  Success - Lint Bash Scripts
[bats.yml/I test shell scripts] ⭐  Run Install Bats
/usr/local/bin/bats -> /usr/local/lib/node_modules/bats/bin/bats
+ bats@1.1.0
| added 1 package from 1 contributor in 1.258s
[bats.yml/I test shell scripts]   ✅  Success - Install Bats
[bats.yml/I test shell scripts] ⭐  Run Run tests
| 1..1
| ok 1 date works!
[bats.yml/I test shell scripts]   ✅  Success - Run tests
```

Took ~4 seconds on my desktop machine.

Get a visualization of the jobs in your workflow:

```
$ act -l
  ┌──────────┐
  │ test-job │
```

`act` Caveats:

- Artifacts are not yet supported, so you may need to comment out steps which use `actions/upload-artifact`
- If later logic relies on downloading the artifacts, you'll need to work around that as well.
- The `checkout` action does not accept any args, so it's quite limited here.
- I've only gotten this working on build steps. So far that has been good enough for my needs.
- May not detect broken workflow configs.

Fine, but what about Perl?

Some Available Docker Images for Perl testing

- The official Perl builds: https://hub.docker.com/_/perl

```
container:
  image: perl:5.32
```

- The official slim Perl builds: https://hub.docker.com/_/perl
  - (based on `debian:buster-slim`)

```
container:
  image: perl:5.32-slim
```

- https://hub.docker.com/r/perldocker/perl-tester
  - A really big hammer

```
 container:
   image: perldocker/perl-tester:5.32
```

perldocker/perl-tester

Pros:

- Comes pre-loaded with Dist::Zilla and Dist::Zilla plugins, Minilla and many test and development modules
- Saves you the download and install time you'd need to do this on demand
- Has nightly builds
- Includes some helper shell scripts to reduce build and test boilerplate

Cons:

- This won't help so much with macOS and Windows
- You might miss some undeclared dependencies if you *only* test on these images

Hints for Maximum Speed

- Build and test with `perldocker/perl-tester` images where possible
- Use the `cache` action to cache Perl module installs in other cases.
  - Adding `~/perl5` to your cached folders may be all that you need.
  - Maybe cache other tools that you need installed on your images.
  - Caches can come back to haunt you, so be careful
- Install with `cpm` rather than `cpanm` where possible, since `cpm` can install in parallel.
- Maybe run code coverage in its own, parallel job
- Build once, test many times

Build Once...

```yaml
jobs:
  build-job:
    name: Build distribution
    runs-on: ubuntu-latest
    container:
      image: perldocker/perl-tester:5.32
    steps:
      - uses: actions/checkout@v2
      - name: Build and test with coverage
        run: auto-build-and-test-dist
        env:
          AUTHOR_TESTING:  1
          RELEASE_TESTING: 1
          CODECOV_TOKEN:   ${{secrets.CODECOV_TOKEN}}
      - uses: actions/upload-artifact@master
        with:
          name: build_dir
          path: build_dir
```

ci-perl-tester-helpers

`auto-build-and-test-dist` is a `bash` script which is availble on the `perl-tester` images.

- will DWIM for `Dist::Zilla` and `Minilla` distributions
- if the `CODECOV_TOKEN` or `COVERALLS_TOKEN` environment variable is detected, a coverage report will be generated and uploaded

See https://github.com/oalders/ci-perl-tester-helpers, https://coveralls.io/ and https://codecov.io/

Add a Coverage Job

```yaml
coverage-job:
  needs: build-job
  runs-on: ubuntu-latest
  container:
    image: perldocker/perl-tester:5.32
  steps:
    - uses: actions/checkout@v2 # codecov wants to be inside a Git repository
    - uses: actions/download-artifact@master
      with:
        name: build_dir
        path: .
    - name: Install deps and test
      run: cpan-install-dist-deps && test-dist
      env:
        CODECOV_TOKEN: ${{secrets.CODECOV_TOKEN}}
```

`cpan-install-dist-deps` and `test-dist` are also provided by

https://github.com/oalders/ci-perl-tester-helpers

Test Many Times, by Building a Matrix

```yaml
test-job:
  needs: build-job
  runs-on: ${{ matrix.os }}
  strategy:
    matrix:
      os: [ubuntu-latest, macos-latest, windows-latest]
      perl-version:
        - "5.10"
        - "5.12"
        - "5.14"
        - "5.32"
      exclude:
        - os: windows-latest
          perl-version: "5.10"
        - os: windows-latest
          perl-version: "5.12"
  name: Perl ${{ matrix.perl-version }} on ${{ matrix.os }}
  steps:
    ...
```

Add the Necessary Steps

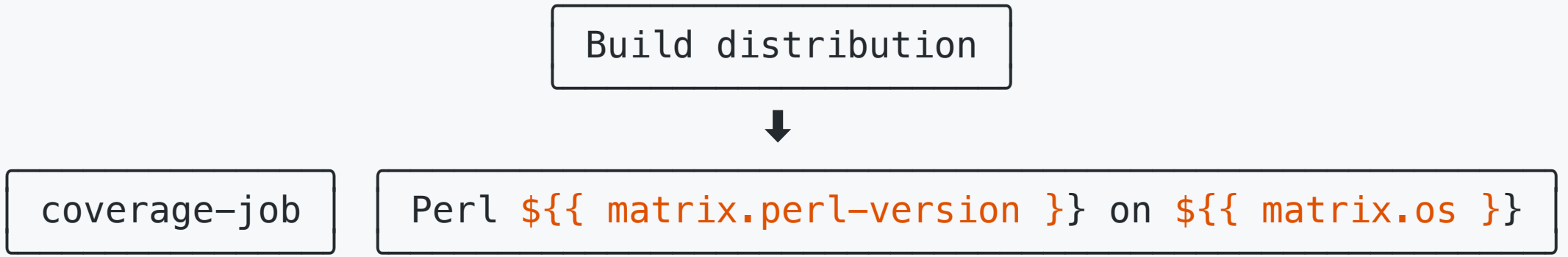
```
    steps:
      - name: Set up Perl
        uses: shogo82148/actions-setup-perl@v1
        with:
          perl-version: ${{ matrix.perl-version }}
          distribution: strawberry # this option only used on Windows
      - uses: actions/download-artifact@master
        with:
          name: build_dir
          path: .
      - name: Install deps with cpm
        uses: perl-actions/install-with-cpm@v1.3
        with:
          cpanfile: "cpanfile"
          args: "--with-suggests --with-recommends --with-test"
      - run: prove -l t xt
        env:
          AUTHOR_TESTING: 1
          RELEASE_TESTING: 1
```

The Entire Config File

It's too big to fit into a file, so please check out the example this was cribbed from at https://github.com/libwww-perl/libwww-perl

Visualize via `act`

```
$ act -l
```



Finding the appropriate Docker image

- Choose an existing Docker image which is appropriate to your needs
- Or
  - Build your own Docker images
  - Push them to Docker Hub
  - Use these as part of your build

Build Your Own Docker Images

```yaml
name: Publish to Docker
on:
  push:
    branches:
      - "master"
  schedule:
    - cron: "10 6 * * *"
jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
      matrix:
        perl-version:
          - "latest"
          - "5.32"
          - "5.30"
    steps:
      - uses: actions/checkout@master
      - name: Publish to Registry
        uses: elgohr/Publish-Docker-Github-Action@master
        with:
          name:       "${{ secrets.DOCKER_REPO }}"
          username:   "${{ secrets.DOCKER_USERNAME }}"
          password:   "${{ secrets.DOCKER_GITHUB_TOKEN }}"
          dockerfile:  Dockerfile
          buildargs:  "BASE=${{ matrix.perl-version }}"
          tags:       "${{ matrix.perl-version }}"
```

See https://github.com/Perl/docker-perl-tester

Spinning up a Docker Container as a Service

```
- name: Run libpostal as a service
  run: docker run -d -p 4400:4400 pelias/libpostal-service

- name: Check running containers
  run: docker ps -a

- name: Get dependencies
  run: go get -v -t -d ./...

- name: Build
  run: go build -v ./...

- name: Check libpostal service
  run: curl -s localhost:4400/parse?address=30+w+26th+st,+new+york,+ny
```

Uploading `cpanfile.snapshot`

```
- name: Maybe update cpanfile.snapshot
  run: carton
- name: Run Tests
  run: carton exec prove -lr --jobs 2 t
- uses: actions/upload-artifact@master
  with:
      name: "${{ matrix.perl-version }}"
      path: cpanfile.snapshot
```

See https://github.com/metacpan/metacpan-web

Manage Issue Labels in Repository (Define Labels)

Create a YAML file at `path/to/manifest/labels.yml`

```yaml
- name: bug
  description: Something isn't working
  color: d73a4a
- name: documentation
  description: Improvements or additions to documentation
  color: 0075ca
- name: duplicate
  description: This issue or pull request already exists
  color: cfd3d7
```

Manage Issue Labels in Repository (Update Labels)

```yaml
name: Sync labels
on:
  push:
    branches:
      - master
    paths:
      - path/to/manifest/labels.yml
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: micnncim/action-label-syncer@v1
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        with:
          manifest: path/to/manifest/labels.yml
```

See https://github.com/micnncim/action-label-syncer

See Also

- Dave Rolsky's "Continuous Integration for Perl with Azure Pipelines" tomorrow (Friday)
- `jonasbn/github-action-perl-dist-zilla`

Bonus Slides:

Installing Perl Modules Without an Available `cpm`

Don't want to use an action?

```
curl -sL https://git.io/cpm | perl - install -g Moo
```

perl-actions/install-with-cpanm

```yaml
- name: Install from cpanfile
  uses: perl-actions/install-with-cpanm@v1.1
  with:
    cpanfile: "cpanfile"
    sudo: false
```

```yaml
- name: Install modules by name
  uses: perl-actions/install-with-cpanm@v1.1
  with:
    install: |
      Simple::Accessor
      Test::Parallel
```

Works on Linux, macOS and Windows

perl-actions/install-with-cpm

```yaml
- name: Install from cpanfile
  uses: perl-actions/install-with-cpm@v1.1
  with:
    cpanfile: "cpanfile"
    sudo: false
```

```yaml
- name: Install modules by name
  uses: perl-actions/install-with-cpm@v1.1
  with:
    install: |
      Simple::Accessor
      Test::Parallel
```

Works on Linux, macOS and Windows

shogo82148/actions-setup-perl

```yaml
- name: Set Up Perl
  uses: shogo82148/actions-setup-perl@v1
  with:
    perl-version: ${{ matrix.perl-version }}
```

Windows:

```yaml
- name: Set Up Perl
  uses: shogo82148/actions-setup-perl@v1
  with:
    perl-version: ${{ matrix.perl-version }}
    distribution: strawberry
```

Windows notes:

- `distribution: strawberry` could save you some heartache
- You may have issues with CPAN module installs on Perls < 5.14

Works on Linux, macOS and Windows

actions/upload-artifact

```yaml
- uses: actions/upload-artifact@v2
  with:
    name: my-artifact
    path: path/to/artifact/ # or path/to/artifact
```

actions/download-artifact

```yaml
- uses: actions/download-artifact@v2
  with:
    name: my-artifact
    path: path/to/artifact
```

`actions/cache`

```yaml
- name: Cache Primes
  id: cache-primes
  uses: actions/cache@v2
  with:
    path: prime-numbers
    key: ${{ runner.os }}-primes
```

Cache `~/perl5` for Perl modules.

elgohr/Publish-Docker-Github-Action

```
      - name: Publish to Registry
        uses: elgohr/Publish-Docker-Github-Action@master
        with:
          name:       "${{ secrets.DOCKER_REPO }}"
          username:   "${{ secrets.DOCKER_USERNAME }}"
          password:   "${{ secrets.DOCKER_GITHUB_TOKEN }}"
          dockerfile: Dockerfile
          buildargs: "FOO=bar"
          tags:       "amazing"
```