DYALOG

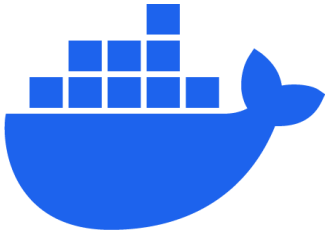
Elsinore 2023

Part 3: Testing Dyalog with Docker and GitHub Actions

Stefan Krüger

git

GitHub



Docker

Aims

- Learn how to run your unit tests from the shell.
- Learn how to use a Docker container to run your application's unit tests
- Learn how to deploy your Docker container as a GitHub Action to run your tests automatically on each commit

Aims

- Learn how to run your unit tests from the shell.
- Learn how to use a Docker container to run your application's unit tests
- Learn how to deploy your Docker container as a GitHub Action to run your tests automatically on each commit

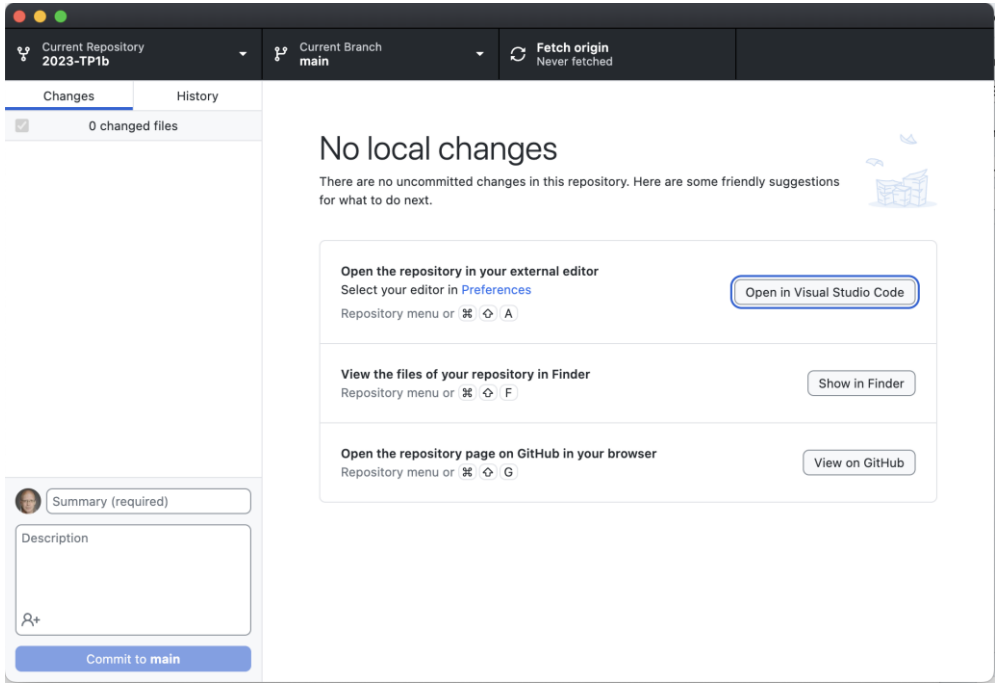
Aims

- Learn how to run your unit tests from the shell.
- Learn how to use a Docker container to run your application's unit tests
- Learn how to deploy your Docker container as a GitHub Action to run your tests automatically on each commit

Pre-requisites

- Docker installed
- `git` installed -- or GitHub Desktop
- A GitHub account
- Dyalog v18.2 + current(ish) `DTest`

GitHub Desktop



DEMO

Task: Fork 'n Clone

- To obtain a working copy, and not having to type along, fork and clone this repository

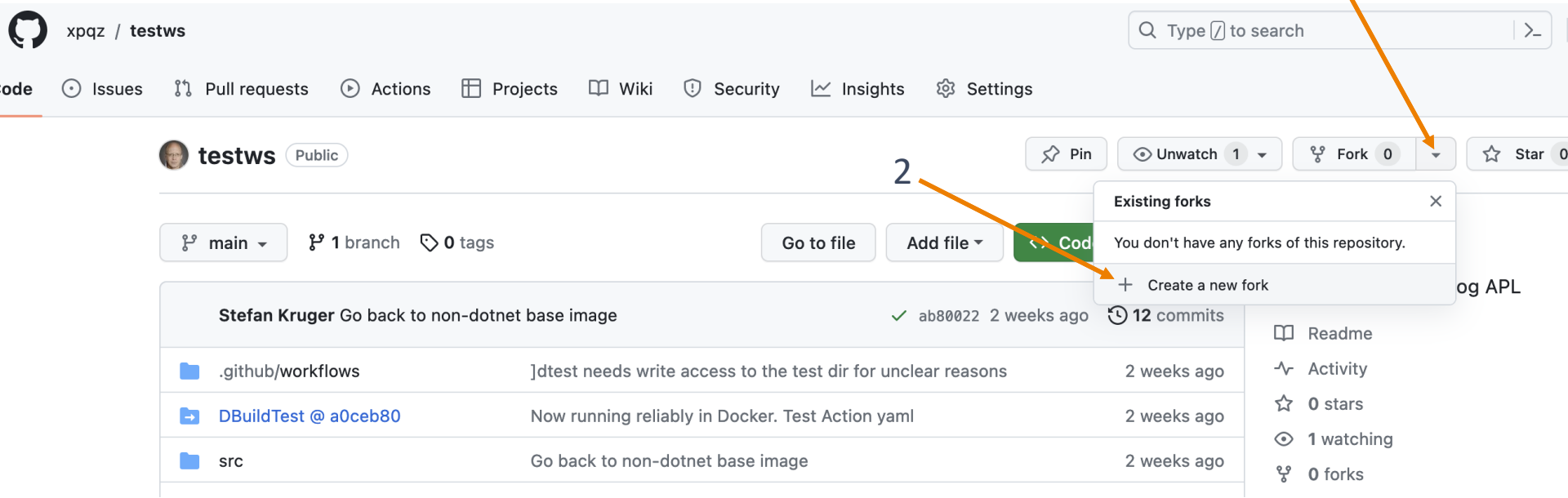
```
https://github.com/dyalog-training/2023-TP1b
```

Task: Fork 'n Clone

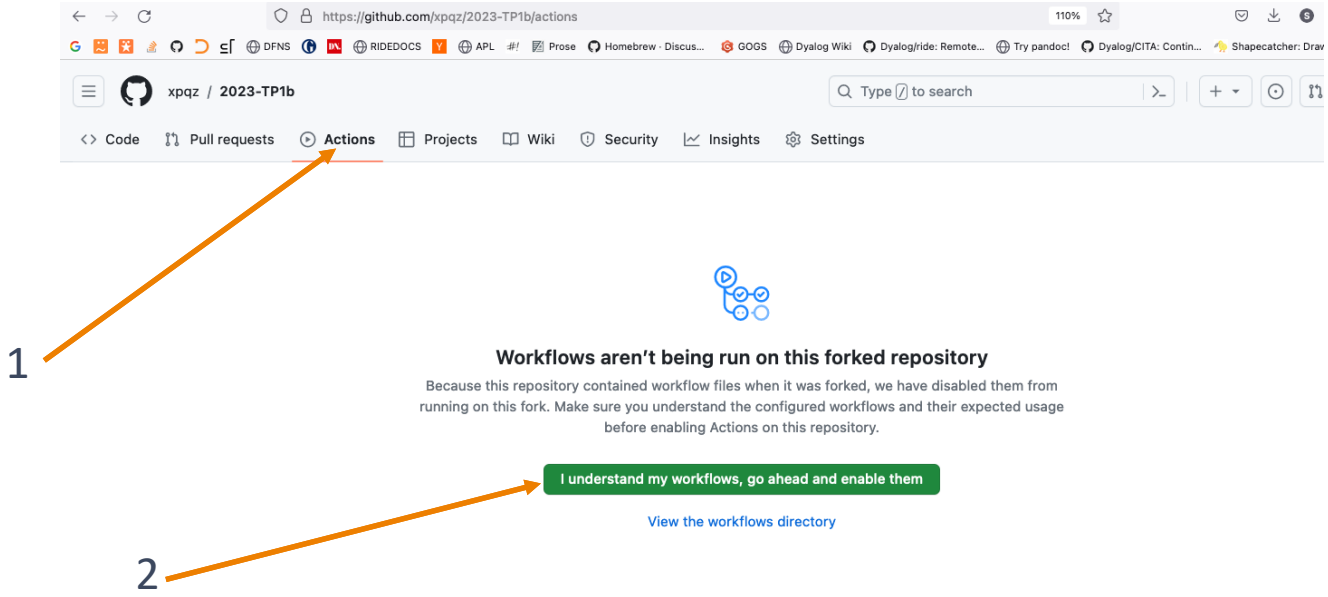
- To obtain a working copy, and not having to type along, fork and clone this repository

`https://is.gd/dytest`

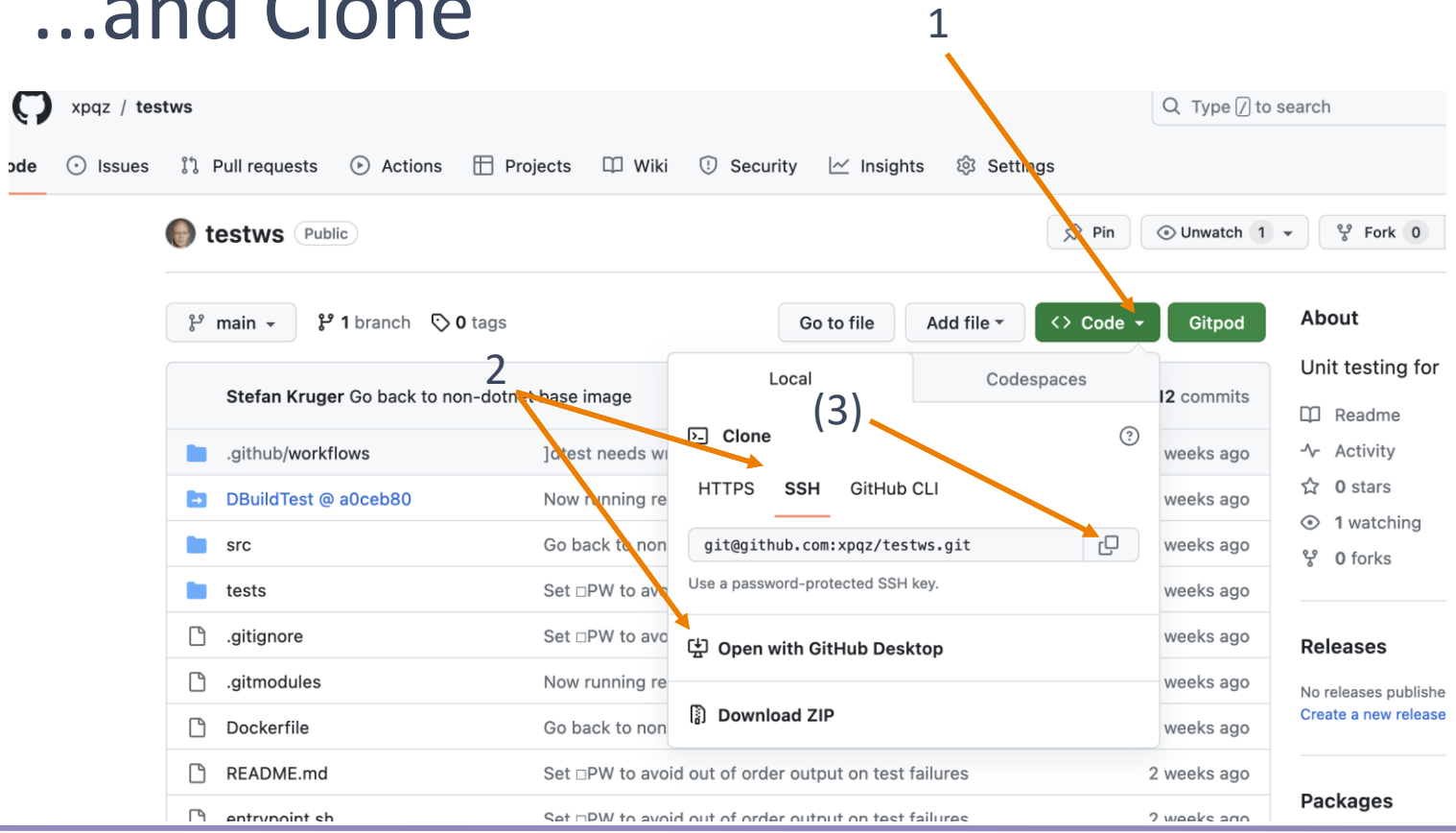
Fork...


1
2
xpqz / testws
Type / to search
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
testws
Public
Pin
Unwatch 1
Fork 0
Star 0
Existing forks
You don't have any forks of this repository.
Create a new fork
main
1 branch
0 tags
Go to file
Add file
Stefan Kruger Go back to non-dotnet base image
ab80022 2 weeks ago
12 commits
.github/workflows
]dtest needs write access to the test dir for unclear reasons
2 weeks ago
DBuildTest @ a0ceb80
Now running reliably in Docker. Test Action yaml
2 weeks ago
src
Go back to non-dotnet base image
2 weeks ago
Readme
Activity
0 stars
1 watching
0 forks

Enable workflows...



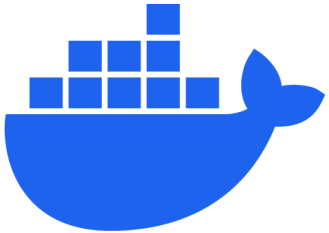
...and Clone


1
2
(3)
xpqz / testws
Type / to search
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
testws
Public
Pin
Unwatch 1
Fork 0
main
1 branch
0 tags
Go to file
Add file
Code
Gitpod
Local
Codespaces
Clone
HTTPS
SSH
GitHub CLI
git@github.com:xpqz/testws.git
Use a password-protected SSH key.
Open with GitHub Desktop
Download ZIP
Stefan Kruger Go back to non-dotnet base image
12 commits
.github/workflows
DBuildTest @ a0ceb80
src
tests
.gitignore
.gitmodules
Dockerfile
README.md
Set □PW to avoid out of order output on test failures
2 weeks ago
About
Unit testing for
Readme
Activity
0 stars
1 watching
0 forks
Releases
No releases publishe
Create a new release
Packages

Console jocks:

```
git clone --recursive git@github.com:{ACCT}/2023-TP1b.git
git clone --recursive https://github.com/{ACCT}/2023-TP1b.git
```

Actions



Docker

GitHub Actions: Continuous integration

- Code changes are automatically built, tested, and integrated into the existing codebase on a frequent basis
- GitHub has a light-weight built-in CI framework called "Actions".
- Combining Docker and Actions, we can test our Dyalog code automatically.

Continuous Integration?

- Code changes are automatically built, tested, and integrated into the existing codebase on a frequent basis
- GitHub has a light-weight built-in CI framework called "Actions".
- Combining Docker and Actions, we can test our Dyalog code automatically.

Continuous Integration?

- Code changes are automatically built, tested, and integrated into the existing codebase on a frequent basis
- GitHub has a light-weight built-in CI framework called "Actions".
- Combining Docker and Actions, we can test our Dyalog code automatically.

Docker?

- Containerisation: lightweight form of virtualisation.
- Containers share the host system's OS and isolate software dependencies.
- Efficient, easy to set up, and compatible across different computing environments.
- Useful for running tests in CI-environments.

Docker?

- Containerisation: lightweight form of virtualisation.
- Containers share the host system's OS and isolate software dependencies.
- Efficient, easy to set up, and compatible across different computing environments.
- Useful for running tests in CI-environments.

Docker?

- Containerisation: lightweight form of virtualisation.
- Containers share the host system's OS and isolate software dependencies.
- Efficient, easy to set up, and compatible across different computing environments.
- Useful for running tests in CI-environments.

Docker?

- Containerisation: lightweight form of virtualisation.
- Containers share the host system's OS and isolate software dependencies.
- Efficient, easy to set up, and compatible across different computing environments.
- Useful for running tests in CI-environments.

Project Layout

```
.
├── src
│   ├── mysum.aplf
│   └── run.aplf
└── tests
    ├── test_assert.aplf
    └── test_mysum.aplf
```

A DTest Function

```
test_mysum ← {
     '1+1 should equal 2'⊢2 Assert 1 #.mysum 1:
     ''
}
```

Task: Run `]dtest` manually

```
]link.create # /{path}/2023-TP1b/src
]dtest /{path}/2023-TP1b/tests
```

Running tests from the command line

- Utilise the `LOAD` parameter
- On start-up, Dyalog will `]link` the folder given
- If it finds a function called `Run`, it will run it

Task: Run tests from shell


```
% dyalog -b -s LOAD=src
Linked: # ↔ /Users/stefan/work/testws/src
All tests passed
```

Task: Make a test fail!


```
% dyalog -b -s LOAD=src
Linked: # ↔ /Users/stefan/work/testws/src
    *** Errors logged
        test_assert: ... = "Assertion failed: 1+1 should equal 2"
         left arg = "3", ⎕DR=83, rho=
        right arg = "2", ⎕DR=83, rho=
 Time spent:  0.0s
 -order="2 1"
```

The `Run` function

- `Run` locates the tests, and executes `]dtest`, and then □OFFs appropriately
- The GitHub Action expects a command to return 0 on success, and non-zero otherwise: □OFF 0 means success.

Run

Called with ⊂DIR by dyalog when encountering LOAD

```
Run dir;testdir;results

testdir ← 'tests',⍕⊃1⎕NPARTS'[/\\]$'⎕R''⊃dir
:If ~(⎕NEXISTS⊃1) testdir,'/test_*'
    ⎕←'No tests found'
    ⎕OFF 0
:EndIf
⎕PW ← 32767
results ← ⎕SE.UCMD'DTest ',testdir, ' -quiet'
:If 0=≢results
    ⎕←'All tests passed'
    ⎕OFF 0
:Else
    ⎕←results
    ⎕OFF 11
:EndIf
```

Find our tests dir

Abandon play if we can't find any tests

Call]dtest

]dtest returns empty vector in quiet mode

Return value 0 for success

FAIL!

Docker

```
[1]  FROM dyalog/dyalog

[2]  ARG DYALOG_RELEASE=18.2

[3]  USER root

[4]  RUN mkdir -p /home/dyalog/MyUCMDs

[5]  RUN chmod 777 /home/dyalog/MyUCMDs && chown dyalog:dyalog /home/dyalog/MyUCMDs

[6]  RUN mkdir /src /tests

[7]  RUN chown dyalog:dyalog /src /tests

[8]  COPY entrypoint.sh /entrypoint
[9]  RUN chmod +x /entrypoint

[10] RUN sed -i "s/{{DYALOG_RELEASE}}/${DYALOG_RELEASE}/" /entrypoint

[11] USER dyalog

[12] ENV LOAD "/src"

[13] ENTRYPOINT ["/entrypoint"]
```

```
[1]  FROM dyalog/dyalog

[2]  ARG DYALOG_RELEASE=18.2

[3]  USER root

[4]  RUN mkdir -p /home/dyalog/MyUCMDs

[5]  RUN chmod 777 /home/dyalog/MyUCMDs && chown dyalog:dyalog /home/dyalog/MyUCMDs

[6]  RUN mkdir /src /tests

[7]  RUN chown dyalog:dyalog /src /tests

[8]  COPY entrypoint.sh /entrypoint
[9]  RUN chmod +x /entrypoint

[10] RUN sed -i "s/{{DYALOG_RELEASE}}/${DYALOG_RELEASE}/" /entrypoint

[11] USER dyalog

[12] ENV LOAD "/src"

[13] ENTRYPOINT ["/entrypoint"]
```

```
[1]  FROM dyalog/dyalog

[2]  ARG DYALOG_RELEASE=18.2

[3]  USER root

[4]  RUN mkdir -p /home/dyalog/MyUCMDs

[5]  RUN chmod 777 /home/dyalog/MyUCMDs && chown dyalog:dyalog /home/dyalog/MyUCMDs

[6]  RUN mkdir /src /tests

[7]  RUN chown dyalog:dyalog /src /tests

[8]  COPY entrypoint.sh /entrypoint
[9]  RUN chmod +x /entrypoint

[10] RUN sed -i "s/{{DYALOG_RELEASE}}/${DYALOG_RELEASE}/" /entrypoint

[11] USER dyalog

[12] ENV LOAD "/src"

[13] ENTRYPOINT ["/entrypoint"]
```

```
[1]  FROM dyalog/dyalog

[2]  ARG DYALOG_RELEASE=18.2

[3]  USER root

[4]  RUN mkdir -p /home/dyalog/MyUCMDs

[5]  RUN chmod 777 /home/dyalog/MyUCMDs && chown dyalog:dyalog /home/dyalog/MyUCMDs

[6]  RUN mkdir /src /tests

[7]  RUN chown dyalog:dyalog /src /tests

[8]  COPY entrypoint.sh /entrypoint
[9]  RUN chmod +x /entrypoint

[10] RUN sed -i "s/{{DYALOG_RELEASE}}/${DYALOG_RELEASE}/" /entrypoint

[11] USER dyalog

[12] ENV LOAD "/src"

[13] ENTRYPOINT ["/entrypoint"]
```

```
[1]  FROM dyalog/dyalog

[2]  ARG DYALOG_RELEASE=18.2

[3]  USER root

[4]  RUN mkdir -p /home/dyalog/MyUCMDs

[5]  RUN chmod 777 /home/dyalog/MyUCMDs && chown dyalog:dyalog /home/dyalog/MyUCMDs

[6]  RUN mkdir /src /tests

[7]  RUN chown dyalog:dyalog /src /tests

[8]  COPY entrypoint.sh /entrypoint
[9]  RUN chmod +x /entrypoint

[10] RUN sed -i "s/{{DYALOG_RELEASE}}/${DYALOG_RELEASE}/" /entrypoint

[11] USER dyalog

[12] ENV LOAD "/src"

[13] ENTRYPOINT ["/entrypoint"]
```

```
#!/bin/bash

export DYALOG=/opt/mdyalog/{{DYALOG_RELEASE}}/64/unicode/
export LD_LIBRARY_PATH="${DYALOG}:${LD_LIBRARY_PATH}"
export WSPATH=$WSPATH:${DYALOG}/ws
export TERM=dumb
export APL_TEXTINAPLCORE=${APL_TEXTINAPLCORE-1}
export TRACE_ON_ERROR=0
export SESSION_FILE="${SESSION_FILE-$DYALOG/default.dse}"

$DYALOG/dyalog -b -s
```

Task 3: Build container locally

```
docker build -t dytest .
```

Task 4: Run it (on macOS or Linux)

```
docker run --rm \

  {FILE SYSTEM MOUNTS}

  dytest
```

Remove container on exit

The messy bits in the middle

Container name

Task 4: Run it (on macOS or Linux)

```
docker run --rm \
  -v
"$(pwd)/DBuildTest/DyalogBuild.dyalog:/home/dyalog
MyUCMDs/DyalogBuild.dyalog" \
  -v "$(pwd)/src:/src" \
  -v "$(pwd)/tests:/tests" \
  dytest
```

Task 4: Run it (on macOS or Linux)

```
docker run --rm \
  -v
"$(pwd)/DBuildTest/DyalogBuild.dyalog:/home/dyalog
MyUCMDs/DyalogBuild.dyalog" \
    -v "$(pwd)/src:/src" \
    -v "$(pwd)/tests:/tests" \
    dytest
```

Task 4: Run it (on macOS or Linux)

```
docker run --rm \
  -v
"$(pwd)/DBuildTest/DyalogBuild.dyalog:/home/dyalog
MyUCMDs/DyalogBuild.dyalog" \
  -v "$(pwd)/src:/src" \
  -v "$(pwd)/tests:/tests" \
  dytest
```

Task 4: Run it (Windows PowerShell)

```
docker run --rm `
 -v
"${PWD}/DBuildTest/DyalogBuild.dyalog:/home/dyalog
MyUCMDs/DyalogBuild.dyalog" `
  -v "${PWD}/src:/src" `
  -v "${PWD}/tests:/tests" `
  dytest
```

```
% docker run --rm \
  -v "$(pwd)/DBuildTest/ {∘∘∘} /DyalogBuild.dyalog" \
  -v "$(pwd)/src:/src" \
  -v "$(pwd)/tests:/tests" \
  dytest
Link Warning: ⎕SE.Link.Create: .NET or .NetCore not availabl
- watch defaults to 'ns'
Linked: # → /src

Rebuilding user command cache... done
All tests passed
```

GitHub Actions

```yaml
name: Run Dyalog APL Unit Tests

on:
  push:
    branches:
      - main

jobs:
  run-dyalog:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2
      with:
        submodules: 'recursive'
        fetch-depth: 0
```

Which events trigger the Action?

...pushes to the main branch

What kind of O/S should the Action runner use?

Check out our repository

Use GitHub's "checkout" action

...including submodules

Build container

```
    - name: Build custom Docker image
      run: docker build -t dytest .

    # ]dtest requires write access to /tests
    - name: Set permissions for /tests
      run: chmod 777 tests

    - name: Run unit tests
      run: |
        docker run --rm \
          -v "${{ github.workspace
}}/DBuildTest/DyalogBuild.dyalog:/home/dyalog/MyUCMDs/DyalogBuild.dyalog"
\
          -v "${{ github.workspace }}/src:/src" \
          -v "${{ github.workspace }}/tests:/tests" \
          dytest
```

Tweak permissions

Run container

Task 5: Action!

Git cheat-sheet

```
git add src/mysum.aplf

git commit -m 'Make a test fail'

git push origin main
```

Summary

- We executed our tests from the shell using LOAD
- We ran our tests in a Docker container
- We deployed our Docker container using a GitHub Action
- Now our every commit triggers a full test run.

Summary

- We executed our tests from the shell using LOAD
- We ran our tests in a Docker container
- We deployed our Docker container using a GitHub Action
- Now our every commit triggers a full test run.

Summary

- We executed our tests from the shell using LOAD
- We ran our tests in a Docker container
- We deployed our Docker container using a GitHub Action
- Now our every commit triggers a full test run.

Summary

- We executed our tests from the shell using LOAD
- We ran our tests in a Docker container
- We deployed our Docker container using a GitHub Action
- Now our every commit triggers a full test run.