CI Systems

Github Actions

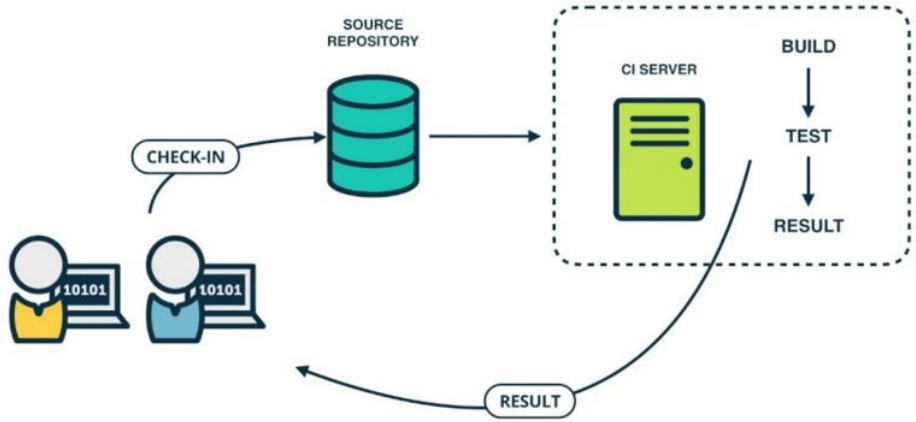
CI Systems

- What is Continuous Integration (CI)?
  - A software development practice where members of a team integrate their work frequently
  - Usually each member integrates at least daily - leading to multiple integrations per day.
  - Each integration is verified by an automated build and test
    - To detect integration errors as quickly as possible.
- Why continuous integration?
  - Easier to handle smaller integrations often, than one big at the end.
  - **All** tests are **always** run automatically for **all changes** to the codebase; not at the developer's discretion
  - Code quality feedback comes sooner rather than later
  - Reduces "It works on my computer"-related issues
  - Increases confidence in the product

CI Systems

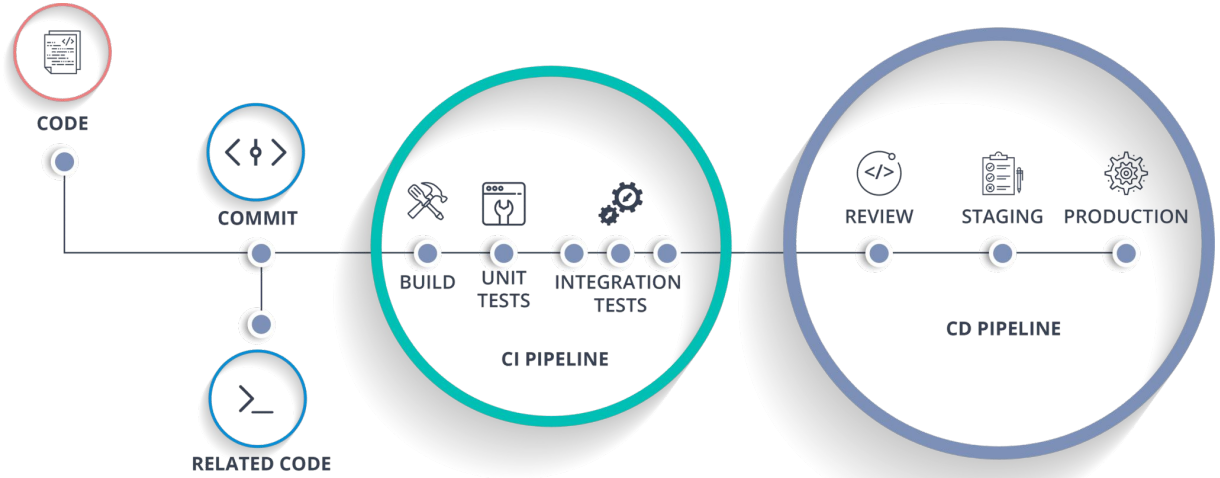
- How do we make it happen?
  - Maintain a single source repository
  - Automate the build
  - Make the build self-testing
  - Everyone commits to the **mainline** every day
  - Every commit should build the **mainline** on an integration machine (CI server)
  - Keep the build fast
  - Test in a clone of the production environment
  - Make results easily accessible
- A CI server is used to automate
  - Building, testing, reporting and etc.

Some CI system examples



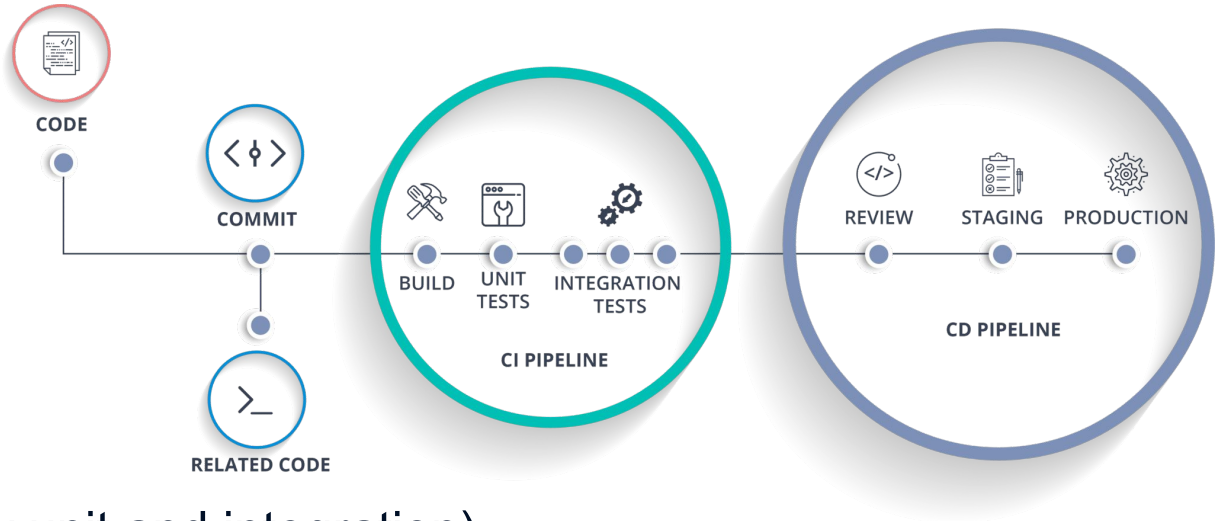
CI Systems

- Maintain a single source repository
  - Setting up a new environment should be problem free and scripted
  - Keep everything required to build the system in the repository!
    - Test scripts
    - Properties files
    - Install scripts
    - Third-party libraries
    - etc.
  - Exceptions
    - Core softwares like OS, git
    - Dependencies that are large or complicated to install



CI Systems

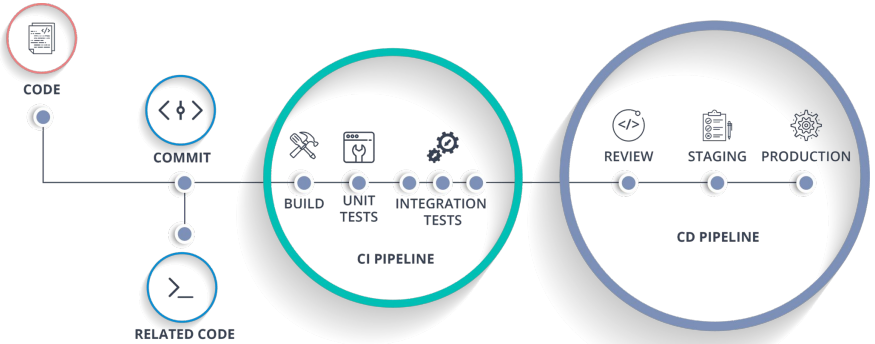
- Automate The Build
  - Build and setup everything required for the application to run and be tested
  - Common to use automated build systems:
    - Make (QMake, CMake)
    - Maven
    - Gradle
    - et.c.



- Make the Build self-testing
  - Include automated tests (mainly unit and integration)
  - Failing tests should fail the build!

CI Systems

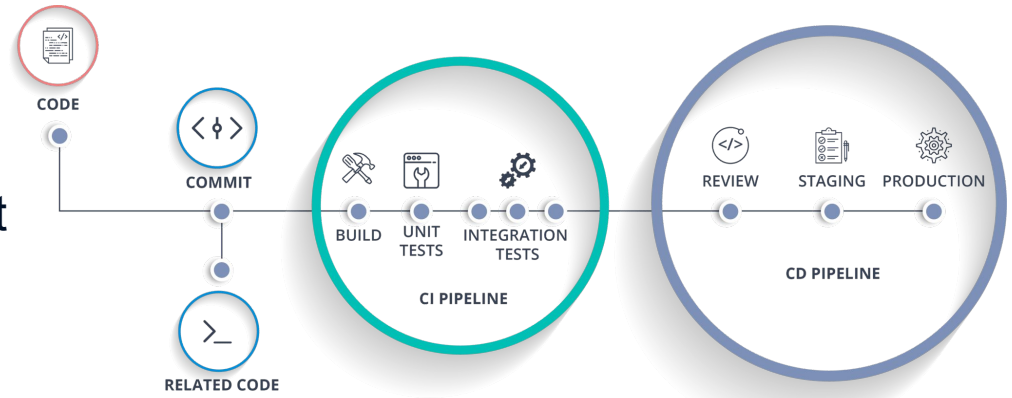
- Everyone Commits To the Mainline Every Day
  - Ideally, **mainline** is the main integration branch
    - Typically master or development branches
  - The **mainline** can be a feature-branch
    - And **everyone** means the developers who are working on the feature.
  - Break your work down to small but meaningful chunks (to a couple of hours)



- Every commit should build the mainline on a CI server
  - Automated build triggering
  - Use build agents in collaboration with the CI server
  - If you use feature branches (most likely), trigger for commits on them as well

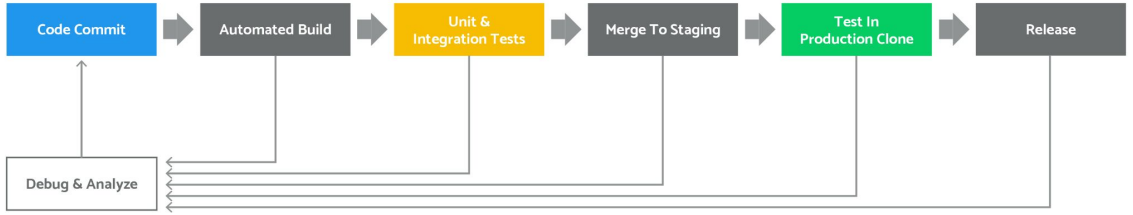
CI Systems

- Keep the build short and fast
  - Fast builds equals fast feedback
    - If it takes long time, people will ignore it
    - Risk of multiple commits in the pipeline
  - Use staged builds for longer test suites
    - Fast failed - e.g. all unit tests and simple ones first.
  - Avoid time-taking integrations test. E.g. UI testing, hitting database and etc.
    - Use test doubles (e.g. mocks, stubs and ect.) and dependency injections
  - Keep feature branches small and integrate often to main/master/develop
  - Before pushing user runs local tests



CI Systems

- Test in a Clone of the Production Environment
  - More production-like means more likely to find production issues
  - In embedded, this often means running on the target platform
  - Isolate business logic to get quick feedback from first build stages
  - Use Hardware-In-The-Loop (HIL) testing in later build stages



- Make results easily accessible
  - Display current build state
  - Automate deployment to a demo unit
  - Provide the latest binaries and test results

CI Systems - Best Practices

- Check in regularly
  - A couple of times per day. Every developer!
  - Don’t commit broken code.
- Fix broken build fast
  - Don’t commit on a broken build
- Be prepared to revert
  - Revert if you can’t fix the problem in 10 minutes
  - Don’t comment out the tests

CI Systems - Best Practices

- The one who checks in, must monitor the build process
  - Should not start new work until the build is OK
    - No lunch, don’t go home...
- Run tests both on your computer and on the CI server
  - Update from VCS, get the latest version
  - Run tests and build, keep it fast
  - Commit and push
- Test in production environment

CI Systems - Best Practices

- A view of the build and test result
  - Everyone should see the status
    - Test coverage, performance and code analyses, green/red status and etc.
- Feature branches must be short lived
- Limit the number of open branches and use “pull request”.
- Always do real testing on your branches before merging
- Rebase from master before running tests to get the latest changes
- Tag the branch on releases

CI Systems - Github Actions

- Github Actions was released in 2019
- CI workflows are added in a new directory .github/workflows/
- Workflow files must use the YAML-format

- There are so many predefined actions that can be added to your workflow
  - Actions can be found on the Github Marketplace
- YAML file format is a human readable format for configurations, data storage, etc.
  - Can store strings, integers, floats, lists and associative arrays
  - Comments begin with a #
  - Whitespace indentation with spaces is used for denoting structure
  - For a full reference look at Wikipedia

CI Systems - Github Actions Workflow

```yaml
name: C/C++ CI

on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: configure
      run: ./configure
    - name: make
      run: make
    - name: make check
      run: make check
    - name: make distcheck
      run: make distcheck
```

- **Events**: Used to trigger the workflow
- **Jobs**: A workflow run is made up of one or more jobs.
- **Runners**: Runs command-line programs using the operating system's shell.
- **Steps**: A job contains a sequence of tasks called steps
- **Actions/commands**: Select an action to run as a part of a step in your job
  - E.g. **actions/checkout** source and documentation. actions/checkout
- Github Docs
  - GitHub Actions Documents
  - GitHub Actions - Supercharge your GitHub Flow
  - Workflow syntax for GitHub Actions
  - Events that trigger workflows

CI Systems - Github Actions Artifacts

```yaml
name: FizzBuzzWhiz CI

on:
  push:
    branches:
      - main

jobs:
  build:
    name: Build
    runs-on: ubuntu-latest
    steps:
    - name: Checkout
      uses: actions/checkout@v2.3.4
    - name: Install
      run: sudo apt-get install build-essential
    - name: Build
      run: make
    - name: Run The Test
      run: make check
    - name: Run The Program
      run: make NUM=123 run
    - name: Upload a Build Artifact
      uses: actions/upload-artifact@v2.2.3
      with:
        name: fizzbuzzwhiz
        path: build/main
```

- An artifact is a file or collection of files produced during a workflow run.
- In Github Actions it is possible to share data (artifacts) between jobs in the same workflow.
- These are some of the common artifacts that you can upload:
  - Log files and core dumps
  - Test results, failures, and screenshots
  - Binary or compressed files
  - Stress test performance output and code coverage results
- It is possible to upload, download and delete artifacts
- By default, GitHub stores build logs and artifacts for 90 days
- Read more about Storing workflow data as artifacts

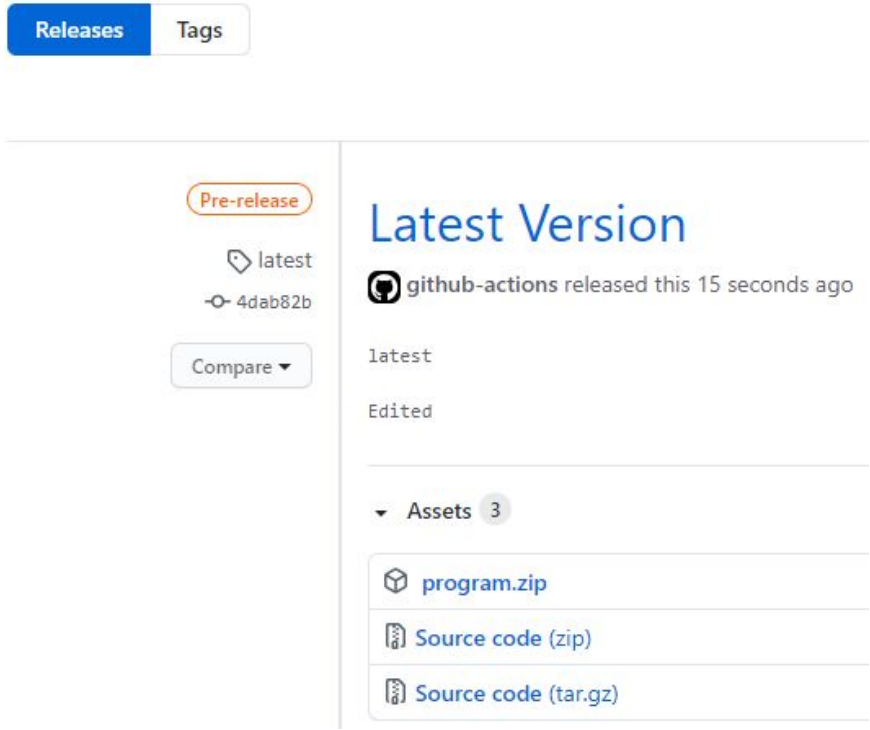
CI Systems - Github Actions Artifacts

```yaml
name: Release

on:
  push:
    branches:
      - main

jobs:
  release:
    name: Release
    runs-on: ubuntu-latest
    steps:
    - name: Checkout
      uses: actions/checkout@v2.3.4
    - name: Install
      run: |
        sudo apt-get install zip
        sudo apt-get install build-essential
    - name: Build
      run: |
        make
        cd build
        zip program.zip main
    - name: Run The Test
      run: make check
    - name: Create Release
      uses: marvinpinto/action-automatic-releases@v1.1.1
      with:
        repo_token: ${{ secrets.GITHUB_TOKEN }}
        automatic_release_tag: latest
        prerelease: true
        title: Latest Version
        files: build/program.zip
```

- You can make an automated release and store artifacts also
- For example [GitHub Automatic Releases] action can be used



CI Systems

- Some useful links
  - Introduction to CI/CD
  - Github Actions CI/CD
  - Github Actions Tutorial
  - Hardware-in-the-Loop
  - GitHub Actions