

## L04 Continuous Integration

Markus Raab

Institute of Information Systems Engineering, TU Wien

This work is licensed under a Creative Commons  
*"Attribution-ShareAlike 4.0 International"* license.



# Use

- 1 Use
- 2 Reproducibility
- 3 Improve
- 4 Meeting
  - Recapitulation
  - Assignments
  - Preview

# Learning Outcomes

After successful completion of L04  
students will be able to

- use continuous integration
- remember basics of reproducibility
- improve continuous integration

# Workload

- maintainers of FLOSS usually have high workload
- review focus on difficult aspects
- use CI to answer trivial questions

# Check If

- the code compiles without warnings (with different compilers)
- all test suites run successfully (on different systems)
- release notes are written
- code is correctly formatted
- consistency rules are fulfilled
- ABI/API is compatible
- all content has a license

# Artifacts

The build server gives us:

- documentation
- executables
- reports of
  - test runs
  - memory leaks
  - code coverage
- website, ...

# Trigger Build

In Elektra:

- Build server runs on every push in every branch
- Nightly/monthly/. . . builds
- Jenkins is the main CI
- Manual trigger by writing a comment:  
“jenkins build libelektra please”
- Sometimes CI problems need to be reported and fixed
- Maintenance reports in [issues.libelektra.org/160](https://issues.libelektra.org/160)

# Conclusion

- Start of build server automatically
- Reading of build server results manually
- Set label “ready to merge” iff build server passes
- Make sure that new test cases are executed
- CI handled like other code



## L04 Continuous Integration

Markus Raab

Institute of Information Systems Engineering, TU Wien

This work is licensed under a Creative Commons  
*"Attribution-ShareAlike 4.0 International"* license.



# Reproducibility

- 1 Use
- 2 Reproducibility
- 3 Improve
- 4 Meeting
  - Recapitulation
  - Assignments
  - Preview

# Goals

- ① get identical behavior from tools, e.g.:  
same error messages or reformatting of source code
- ② get same failures when running tests
- ③ (re)create identical binaries

## Goal 1: Compilation

For identical results installation of exactly the same tools is needed.

### Problem

*Different distributions offer different tool chains, compiled differently.*

### Solution (Possible Solution)

*Install tools with identical version manually.*

## Goal 2: Tests

### Problem

*For identical results the whole execution environment need to be replicated.*

### Solution (Possible Solution)

*Virtualization but even then with limitations.*

# Dockerfile

- allows to define an image to be used for the build server
- FROM defines a base image
- RUN are commands to be executed that modify the image

Alternatives: NixOS, Guix

# Example

scripts/docker/debian/bullseye/Dockerfile:

```
1 FROM debian:bullseye
2
3 ENV LC_ALL C.UTF-8
4
5 RUN apt-get update && apt-get -y install \
6     build-essential
7 RUN ldconfig
8
9 RUN useradd jenkins
10 USER ${JENKINS_USERID}
11 RUN git config --global user.email 'Jenkins<autobuilder@libel
12     && git config --global user.name 'Jenkins'
```

## Goal 3: Binaries

### Problem

- *Different distributions offer different tool chains.*
- *Tool chained can be compiled or configured differently.*
- *Any current dates or build paths influence binaries.*
- *Any input from outside the source code (e.g. Internet) is volatile.*



# Solutions

- Virtualization of build environment
- Recording of build environment (e.g. buildinfo files)
- Let build process ignore the environment  
→ reduce problem to Goal 1

# Reproducible Builds

- verifiable path from source to binary
- reproducible by default (without virtualization)
- reduces influence of build environment
- make build system deterministic

# Limitations

Build system and macros enable/disable code depending on:

- compiler
- operating system
- installed dependencies

E.g. in Elektra plugins get disabled depending on missing dependencies.

## Solution

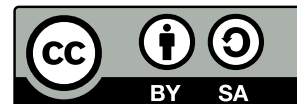
*In practice a combination of recording and ignoring the environment is used.*

## L04 Continuous Integration

Markus Raab

Institute of Information Systems Engineering, TU Wien

This work is licensed under a Creative Commons  
*"Attribution-ShareAlike 4.0 International"* license.



# Improve

- 1 Use
- 2 Reproducibility
- 3 Improve**
- 4 Meeting
  - Recapitulation
  - Assignments
  - Preview

# Jenkins

- cron on steroids
- many plugins
- implemented in Java
- Jenkinsfiles directly in source code

# Jenkinsfile

for Elektra in scripts/jenkins/Jenkinsfile

- either declarativ or in Groovy
- enables review of CI
- different CI for different PRs

# Jenkinsfile Example

Run one script:

```
1  def tasks = [:]
2  tasks.failFast = false
3  tasks << buildIcheck()

1  def buildIcheck() {
2  def stageName = "icheck"
3  return [(stageName): {
4      stage(stageName) {
5          withDockerEnv(DOCKER_IMAGES.bullseye) {
6              sh "scripts/build/run_icheck"
7              deleteDir()
8          }
9      }
10 }]
11 }
```



# Useful Tests

## Problem

*CI must be fast enough.*

- Verify a fact everyone agreed on is actually the case.
- Unit or integration tests run with different compilers and operating systems.
- Verify that a bug is fixed (regression).
- ABI/API tests: that behavior for customers is unchanged.

# Order is Important

- first check locally
- then implement automatic check
- then add automatic check to CI

# Definition of Done

Continuous improvements:

- underlying issue for discussion
- updated documentation
- added tests
- improved code comments
- review done
- QA (valgrind, ASAN, ...)

## L04 Continuous Integration

Markus Raab

Institute of Information Systems Engineering, TU Wien

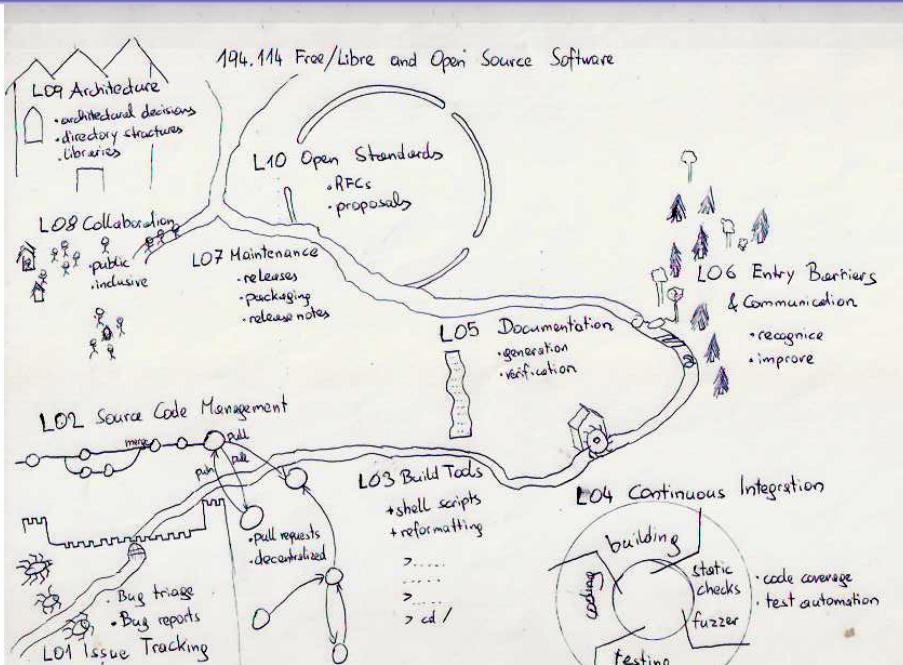
This work is licensed under a Creative Commons  
“*Attribution-ShareAlike 4.0 International*” license.



# Meeting

- 1 Use
- 2 Reproducibility
- 3 Improve
- 4 **Meeting**
  - Recapitulation
  - Assignments
  - Preview

## Recapitulation



# Learning Outcomes

After successful completion of L04  
students will be able to

- use continuous integration
- remember basics of reproducibility
- improve continuous integration

# Brainstorming: What can CI check for us?



# Trigger Build

## Task

How do you trigger a rebuild?

How do you report CI problems?

# Trigger Build

In Elektra:

- Build server runs on every push in every branch
- Nightly/monthly/. . . builds
- Jenkins is the main CI
- Manual trigger by writing a comment:  
“jenkins build libelektra please”
- Sometimes CI problems need to be reported and fixed
- Maintenance reports in [issues.libelektra.org/160](https://issues.libelektra.org/160)

# Brainstorming: Which artifacts can CI produce for us?

Task

Break.

# Presentation reproducibility (Juri Schreib)

Task

Any questions?

# Goals

## Task

What are the three goals of reproducibility?

# Goals

- ① get identical behavior from tools, e.g.:  
same error messages or reformatting of source code
- ② get same failures when running tests
- ③ (re)create identical binaries

# Docker

Goal 2:

Task

Insofar can docker improve reproducibility when running tests?



# Docker

- use the same base image
- modify the image in the same way
- use the same user
- get the same environment (LC\_ALL, ...)

Task

Break.

# Useful Tests

## Task

What do you need careful about when adding tests?

Which tests are particularly useful?

# Useful Tests

## Problem

*CI must be fast enough.*

- Verify a fact everyone agreed on is actually the case.
- Unit or integration tests run with different compilers and operating systems.
- Verify that a bug is fixed (regression).
- ABI/API tests: that behavior for customers is unchanged.

# Order is Important

## Task

Which steps do you usually do before creating an PR to fix a problem?

# Order is Important

- first check locally
- then implement automatic check
- then add automatic check to CI

Task

Break.

# P0 Feedback

## Task

User-facing documentation



## H2 Reviews

### Task

Which reviews did you find useful?

### Task

Don't review trivial PRs.

# T1 Continuous Integration

Task

Any questions?

# L05 Documentation