Approximate Methods in Geometry

Practical Part

Jens K. Mueller

jkm@informatik.uni-jena.de

Department of Mathematics and Computer Science
Friedrich-Schiller-University Jena

Friday 12th November, 2010

Today's lecture: Git & CMake

Writing Good Tests

- It's seems annoying but it isn't
- Boundary Cases
- You can use your code
- It pays off!

Version Control

Distributed Version Control with Git

What is Git?

- Distributed Version Control System
- Developed by Linus Torvalds
- Runs almost everywhere
- Used by Linux kernel, Samba, X.Org, Qt, GNOME, Android, ...

Features:

- Very flexible work flows
- Fast and Scalable
- Cryptographic Secure History

How Git stores its Data

Everything is an object with a SHA1

All git objects have a type, content, and size (of the content). For a given object its (object) name is a 40-digit hash (SHA1) of its content.

Object Types:

- Blob Object
  Content: data
- Tree Object
  Content: list of blob and tree names with its type and file name

How Git stores its Data (cont.)

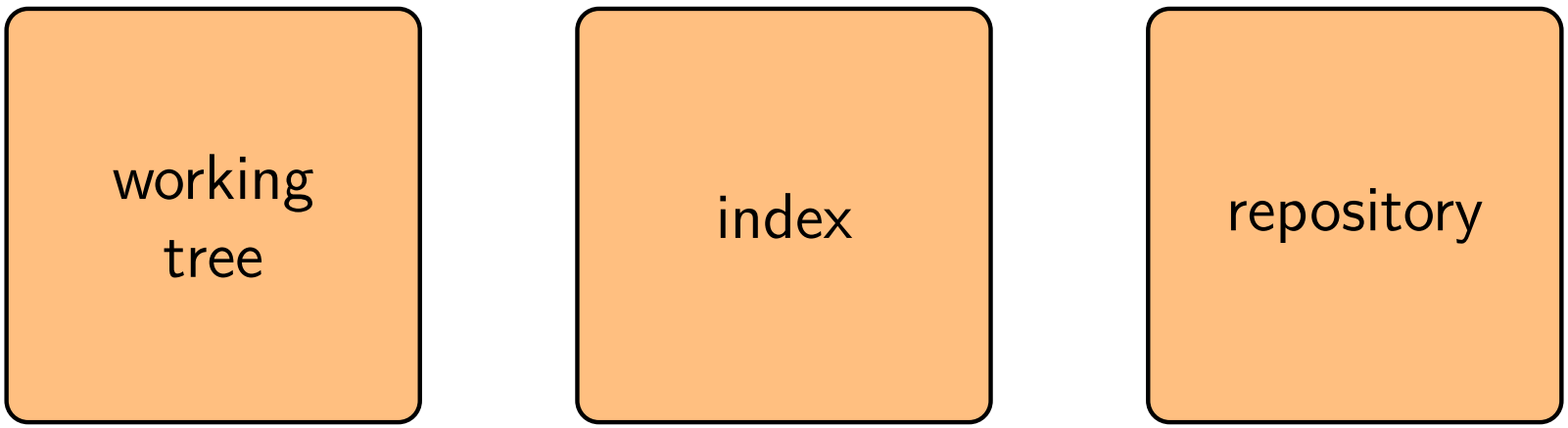
Everything is an object with a SHA1

- Commit Object
  Content: 1 tree name, 0+ parent commit name(s), author, committer, and a commit message
- Tag Object
  Content: type, name, tagger, tag message

The commit history of a project forms a directed acyclic graph.

Assuming SHA1 is safe: Given a commit and its SHA1 the whole history of the project is secured. I.e. a change in the history can be noticed.

Interacting with Git



Adding

- Adding a file pattern to the index
  ```
  $ git add filepattern
  ```
- Adding a file to the index ignoring untracked files
  ```
  $ git add -u filepattern
  ```
- Automatically stage files and commit
  ```
  $ git commit -a
  ```

Checkout

- Checkout a file from the index
  `$ git checkout - file`
- Checkout a file from the repository
  `$ git checkout HEAD file`

Diffing

- Diff file against the index
  ```
  $ git diff file
  ```
- Diff file in index against the repository
  ```
  $ git diff -cached file
  ```
- Diff file against the repository
  ```
  $ git diff HEAD file
  ```

Commit Message

```
A summary in a single line less than 50 charaters

A more detailed explaination, if necessary. Wrapped at
about 72 characters. Write in imperative present tense.
It should include your motivation for the change and
contrast the new implementation/behavior with the old.

- bullet point
  indent here

- bullet point
```

History

- To get the commit history
  ```
  $ git log
  ```

Git References

The Git Community. The Git Community Book. 2010. URL: http://book.git-scm.com/

There is much more:

- Much documentation (man pages, cheat sheets, tutorials, ...)
- Many users

Building Software

Configure, Build, and Install

1. Configure
   Are all dependencies (like libraries, system calls) fulfilled?
   Is there a working compiler? What features to enable?
2. Build
   Controls the building process and inter source dependencies.
3. Install
   Copy files to the appropriate places.

In the Unix world the configuration is done via autotools's configure. This generates a Makefile that is used for building. The Makefile provides an install target.

CMake

- Portable software configuration tool
  Works on *nix based (Linux, MacOS X, ...) and Windows.
- Supports different software build tools
  NMake (Microsoft's make), Unix Makefile, Visual Studio, KDevelop, Eclipse
- Integrates software packaging and testing

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project(HelloWorld)

# flags
set(CMAKE_CXX_FLAGS "-Wall -Wextra -ansi -
    pedantic")

# building
include_directories(src)
add_library(helloworld SHARED src/
    helloworld.cc)
add_executable(helloworld_main src/
    helloworld_main.cc)
target_link_libraries(helloworld_main
    helloworld)
```

CMakeLists.txt (cont.)

```
# testing
enable_testing()
include(CTest)

set(REQUIRED_LIBRARIES "helloworld")

find_package(GTest REQUIRED)
include_directories(${GTEST_INCLUDE_DIR})
set(REQUIRED_LIBRARIES ${REQUIRED_LIBRARIES
    } ${GTEST_LIBRARIES})

find_package(Threads REQUIRED)
set(REQUIRED_LIBRARIES ${REQUIRED_LIBRARIES
    } ${CMAKE_THREAD_LIBS_INIT})
```

CMakeLists.txt (cont.)

```
add_executable(test_helloworld test/
    helloworld.cc)
target_link_libraries(test_helloworld ${
    REQUIRED_LIBRARIES})
gtest_add_tests(test_helloworld "" test/
    helloworld.cc)

# installing
install(FILES helloworld.h PUBLIC_HEADER
    DESTINATION include)
install(TARGETS helloworld helloworld_main
          RUNTIME DESTINATION bin
          LIBRARY DESTINATION lib)
```

Guidelines when Building the Software

- Enable warnings
- Decent optimization level
- Release/Testing/Debug builds