Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CS Basics
16) C Tools

E. Benoist & C. Grothoff
Fall Term 2018-19

Computer Science Division

C Tools

Writing code

GNU indent

Indentation matters:

- Code faster to read
- Avoids bugs
- Formatting changes result in unnecessary commits
- Best practice: automatically indent

GNU indent provides automatic indentation for C, different styles available!

ctags

Identifiers matter:

- Code search
- Cross-referencing
- Autocompletion

ctags extracts “tags” from C code. Output supported by various editors.

Version control

Learn about:

- Branches (experimental developments, maintenance)
- Tags (checkpoints, release marking)
- Bisection (identify time bugs were introduced)
- Push, pull, rebase, merge
- Signing of commits
- Hooks
- Gitolite (access control, repository management for groups)

Documenting code

Doxygen

Comments matter:

- Internal logic (normal comments)
- API documentation
- Manuals
- Man pages (see also: nroff/troff)

Doxygen excells at API documentation, creates HTML, PDF, etc.

Example

```
/**
 * @author Christian Grothoff
 * @brief First ``**'' indicates doxygen-style ↘
 →comment.
 *
 * Here you write the general documentation ↘
 →about
 * the ``fun'' function.  Using #hash adds a
 * link to the ``hash'' symbol.  You can also
 * refer to function symbols using fun().
 *
 * @param argname documents an argument
 * @param result[out] indicates that result is ↘
 →set
 * @return documents the return value(s)
 */
int
fun (int argname, float *result);
```

GNU TexInfo / sphinx

Manuals matter:

- Write documentation in TexInfo or markup
- Output HTML, PDF, info, man pages, etc.
- Requires some planning, due to differences in how output is structured (HTML nodes, PDF chapters, etc.)

Testing code

automake

Build-in test suite logic:

- TESTS = testcases
- `check_PROGRAMS` = binary-tests
- `check_SCRIPTS` = script-tests
- Return 0 on success
- Return 77 to indicate “skip” (test could not run)

Example

```
TESTS = $(check_PROGRAMS)
check_PROGRAMS = \
  test_foo
test_foo_SOURCES = \
  test_foo.c
# test_foo needs to link against libfoo
test_foo_LDADD = \
  $(top_builddir)/src/foo/libfoo.la
```

gcov / lcov

gcc support for code coverage analysis:

- Figure out which parts of code are (un)tested
- Compiler instruments functions and branches to track execution
- GCC option “–coverage” (use with “-O0”)
- Linker option “-lgcov”
- Analyze result with “lcov” tool
- Generates HTML report

Continuous Integration

Automatically trigger tests across platforms:

- Buildbot
- Gitlab

Can automatically notify developers about regressions!

Debugging

Debuggers

- gdb
- ddd (graphical)
- strace (system calls)

Dynamic analysis

gcc support for undefined behavior:

- -fsanitize=address,undefined
- -fno-omit-frame-pointer (helps analyze stack issues)

valgrind

Dynamic instrumentatation:

- –tool=memcheck (use-after free, double-free, out-of-bounds)
- –leak-check=yes (memory leaks)
- –show-reachable=yes (include leaks from globals on exit)

Profiling / benchmarking code

Benchmarking tools

- time (CPU time, system time, real time)
- strace (number of system calls, time spend in system calls)
- top (CPU usage, memory usage)
- gprof (rough CPU usage per function/line, gcc option “-pg”)
- valgrind –tool=massif (memory)
- valgrind –tool=callgrind
- valgrind –tool=cachegrind
- kcallgrind

Static analysis

Static analysis

- gcc -Wall
- clang
- cppcheck
- FindBugs (Java)
- Coverity (proprietary)
- CodeSonar (proprietary)

Deployment

- Internationalization: GNU gettext
- Autotools: “make dist”
- gnupg: cryptographically sign code
- rpm, dpkg, guix (build packages)
- emscripten: compile to JavaScript

Conclusion

Conclusion

All software developers need to know:

- Languages
- Libraries
- Tools
- Processes

More in Software Engineering and Project Management courses!