

ECE 3574: Applied Software Design

Meeting 2: Version Control

The goal of the next few meetings is to learn about programming tools other than the compiler and how to use representative versions of them.

- ▶ Source Code Management (Version Control)
- ▶ Source Code Configuration and Build
- ▶ Testing Tools

Text Manipulation

- ▶ Plain text is the raw material of programming and the most stable digital form of storing knowledge.
- ▶ You need to be adept at creating it, modifying it, searching it, and filtering it.
- ▶ A key tool is your editor.

Pick a powerful one and learn how to use it effectively (e.g. keyboard shortcuts).

Source Code Management, also called version control or source code control

- ▶ keeps track of every change made to your code (and by whom).
- ▶ It can act as an unlimited undo for your project
- ▶ documents what changes were made, by whom, and when
- ▶ provides archiving and reproducibility of software builds.

Always use some form of source code management, even for small projects.

We will be using git for this, although there are other good alternatives (mercurial, subversion).

Software Configuration and Build tools

You should be able to build all dependencies and the code itself, in debug and release mode, for all platforms supported **in a single step**.

This can be done by a variety of means, including custom scripts and IDE tooling. We will be using a popular open source tool for this called cmake.

Testing and Status

- ▶ Untested Code is Broken Code
- ▶ You should be able to also run the tests associated with a project in a single step.
- ▶ This should be a regular part of the build process.
- ▶ Examples: Catch, CxxTest, Google Test, CppUnit (and many more)

CMake includes a way to specify tests to run as part of the build process so we will use that. We will spend an entire class meeting on this important topic in week three. This week we will look at the very basics using Catch.

Static Checkers

- ▶ Static typed languages like C++ allow a large class of bugs to be caught before the code is run.
- ▶ Modern compilers provide a number of checks that can be applied, although external tools exist as well.
- ▶ The compiler can generate warnings to prevent a large class of bugs. In particular **code should compile cleanly with no warnings**.
- ▶ There are also tools like cppcheck and cpplint that can carry out specific checks. Teams often have their own per-projects scripts as well.

Dynamic Checkers

- ▶ Once code is compiled and run it can also be checked for errors, usually by instrumenting the code automatically or at the run-time level.
- ▶ This can be used to detect a wide variety of problems, **notably memory mismanagement**.
- ▶ Examples: the valgrind suite of tools under unix and the VC++ leak detection facilities on Windows.

Code Style Checkers

- ▶ C++ is a very large and complex language. It is often desirable to limit the parts of the language used.
- ▶ Because code is read much more than it is compiled, uniform formatting and style rules are needed.
- ▶ This applies to personal projects, where you might be reading your own code months (or years) after writing it, as well as multi-person projects.
- ▶ Each organization and project will have its own style guidelines as there is no one-true-style
- ▶ Fortunately many of the pedantic rules, where braces go, etc., can be fixed automatically.
- ▶ Example tools: Artistic Style, clang-format

Code Generators

- ▶ Code generators are “code that writes code”, programs that output code, usually from some form of textual input.
- ▶ These come in two forms, passive and active.
- ▶ Passive generators are run only once and are appropriate for a few use cases, for example generating source files from templates or for generating look-up tables.
- ▶ Active generators run each time the build is run. We will see an example of an active code generator from the QT library this semester, the Meta-Object Compiler, or moc.

Course Tools

In this course we will use just a few tools from those available

- ▶ git for managing source code (command-line or GUI)
- ▶ cmake for build configuration
- ▶ Testing using Catch and Qt testing framework
- ▶ code coverage and memory checkers

Git Demo

- ▶ init - initialize a new repository
- ▶ clone - clone an existing repository: local and remote
- ▶ status - see the state of your repository
- ▶ diff - examine details of what has changed
- ▶ add - adding changes to the index
- ▶ commit - committing the changes in the index
- ▶ log - list previous changes
- ▶ push - pushing changes “upstream”
- ▶ tag - give your commits names

Exercise 02: Git

See Website.

Next Actions

- ▶ Reading on VirtualBox and Vagrant