Inf-1400 - Object Oriented Programming

A programmer's toolchest

John Markus Bjørndalen

Department of Computer Science
University of Tromsø
Norway

2017-02-03

NB

This lecture is considered work in progress.

Useful tools for a programmer

In addition to the compiler and the editor, there are some useful tools (not complete list):

- File comparison and updating files
- Version control
- Build automation tools (make, ant, etc)
- Debugging
- Profiling
- Syntax checkers, source analysis tools
- Runtime analysis tools (memory leaks etc)

Some of these tools may be provided by IDEs (Integrated Developer Environments)

File comparison and patching

Low-level comparison and patching:

- diff – shows the difference between two files.
- patch – applies a patch (“diff file”) to given files.
  - Can be used to update files with modifications provided by others.

Graphical tools for file comparison:

- tkdiff
- Meld

Some of these tools also provide 3-way comparisons

Version control and collaboration

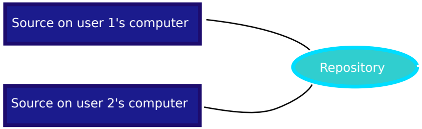
For single user:

- When did I introduce this bug?
- What did I change yesterday?
- Do I have the same version running on all computers?
- Update your copy or revert to older versions

For multiple users:

- Merging code written by different programmers
- When did this bit (that I didn’t write) change?
- Do I have the latest version of the source code?

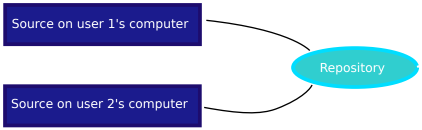
Version control systems



Provides source file comparison, patching and revision control / tracking

Repositories used to store and keep track of source code and modifications made to the source code

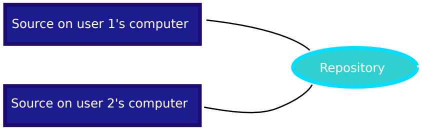
Version control systems



Terminology (Subversion)

Check out:	get a copy of the source from the repository
Update:	update your copy from the repository (and merge with your changes)
Commit:	update repository with your changes (some systems call this check in)

Version control systems



Other common operations:

- Compare your copy of a file (or entire source) with one of the revisions in the source code
- Branching and merging: managing code forks and merging forks within the same repository (useful when you rewrite bits that might break other people’s code)
- Update: update your copy from the repository (and merge with your changes)
- Commit: update repository with your changes (some systems call this check in)

Generations of version control

1. repository kept with the source (RCS etc)
    - An extra “version control” file for each source file to keep track of revisions (“patch/diff log”)
2. Centralized repository (CVS, Subversion)
    - Advantage: collaboration easier, “backup”
    - Disadvantage: may need Internet access to check in and out (offline development can be annoying)
3. Distributed repository (git, mercurial)
    - Advantage: can be used without Internet access
    - Disadvantages:
        - ⋆ must remember that it is multilevel.
        - ⋆ May be more complex for users.

Web-based version control

Some examples:

- github
- gitlab
- sourceforge
- systems that allow you to host your own repositories (Trac[1], Fossil[2], Gitlab)

These systems often use standard version control systems (like SVN and git) but provide extra functionality through the web:

- wiki
- issue tracking
- web-based revision comparison
- source browsing

[1] **`http://trac.edgewall.org/`**
[2] **`http://www.fossil-scm.org`**

Static checking

Compilers:

- Syntax error: when the compiler is unable to translate what you wrote into running code
- Warnings: things that may or may not be bugs or that often cause bugs.
  - GCC: use -Wall to get as many warnings as possible.

Other tools:

- lint / plint / splint – a C syntax checker
- pylint for Python

Runtime

Memory leak detection

- Valgrind
- Lots of other tools

Robustness checking

- Look for hacking tools
- Tools that stress test your system
- Trying random parameters / data
- Fuzz testing: `http://en.wikipedia.org/wiki/Fuzz_testing`

Profiling

Where do you spend your time and which functions do you call often?

Often sampling based: periodically checks where in the program your instruction pointer is and what the call stack is like.

Provides a profile of where you spent your time: see proftest.c

Debuggers

Inspect running code.

Pause code at specified points in the code or in time (breakpoints).

Check (and possibly modify) values in the running program.

Examples:

- C/C++ : gdb, xxgdb, ddd, nemiver, Visual Studio, +++
- Python: pydb (can also be used from, for instance, ddd)

Integrated Developer Environments

Some examples:

- Microsoft Visual Studio
- Eclipse
- PyCharm / IntelliJ

Usually provides several tools integrated in the developer environment or provides a simple plug-in system to provide necessary tools.

Tools for studying

Wiki to Collaborate with other students. Earlier years: students made a wiki for a course as a method for organizing information and rehearsing for the exam

Others have used dropbox (with mind maps etc)

Can also use other tools (github/gitlab etc)

Conclusions

This lecture only gives you a quick overview of some of the useful classes of tools. There are other tools out there as well, and more will come.

One very important thing that is not covered here: ALWAYS USE BACKUPS! (And check that you can restore!)