Week 20 commencing on 30/01/2012

# Tools for Programming from the Small to the Large

1

## Tool types

- Version control (e.g., **Subversion)**
- Build control (e.g., **make)**
- Debuggers (e.g., **gdb)**
- Unit/regression testing (e.g,. **JUnit)**
- Bug/issue tracking (e.g., **GNATS)**
- Documentation generation (e.g., **JavaDoc)**
- Project management (e.g., **MS Project)**
- Integrated suites (e.g., **RUP)**
- *Others*

2

## Version Control (VC)

3

## Introduction

- Before starting any project (programming or otherwise!) with edits occurring over more than a couple of weeks, you should set up a *version control system*.
- **If you do not, you deserve every one of the many troubles that will be coming your way ☺**

4

## Introduction (2)

- Version control or configuration management systems like **CVS** or **Subversion** track and manage changes during development and maintenance of any long-lived set of files.
- They tell you who changed *what*, *when*.

5

## *Subversion* example

```
UNIX> svn update
UNIX> svn diff -x -u -r 295 tools.tex
-   is teh best
+   is the best
UNIX> svn commit -m "Fixed typo" tools.tex
UNIX> svn log tools.tex
r296 | jbednar | 2008-01-08 | 1 line
Fixed typo
r295 | jbednar | 2007-12-23 | 30 lines
Added section on Subversion
```

6

## Interpretation of example commands

1. I had edited the file **tools.tex**, so I updated my copies to make sure they were current
2. checked the differences, then
3. committed my change.
4. The new version of tools.tex is now 296, with the current date stamp.

7

## Basic *VC* Features

- Have separate repository holding all versions of all files
- Changes in local copies don't affect the repository until a commit (e.g., check-in)
- Commits merge your changes with the repository files
- Locking or merge mechanisms prevent or resolve collisions between users
- Can provide complete history of all changes, reproducing state at any time and tracking who changed what when

8

## Managing Software Releases

- Any decent company will have version control already
- Release to the public: modified snapshot of repository
- Typical sequence for a release:
  - 1. Feature freeze (only bug fixes allowed now)
  - 2. Release candidate (could ship if bug-free)
  - 3. Candidate sent to testing team
  - 4. Bugs are found, patched; new release candidate
- Typically there are different code branches maintained (at least production and development)

9

## Generations of VC Systems

- **1972 Single file system**; locking ensures only one person at a time edits any file. Nearly unusable for multiple developers. Examples: **RCS, SCCS, VSS ($$)**
- **1985 Realization**: file merging works! No need for locking; just merge later. Allows multiple distributed copies to be worked on. e.g. **CVS, Subversion, Perforce**

10

## Generations of VC Systems (2)

- **2002 Realization:** repository merging works! DVCS: no need for one central repository; repositories and not just files can be merged as needed. Useful even for single developers, to handle independent sets of changes separately.
- Examples: **Mercurial, bzr, Git, Bitkeeper**

11

## Which VC to use?

- **CVS was the de facto standard; Subversion is now on top** because it fixes various technical problems with CVS.
- 3rd-generation DVCS tools are much more powerful but no clear frontrunner has emerged yet.
- **Git's picking up** territory, with the announcement of its adoption by Perl in Dec '08, but **Mercurial/hg is out in front at the moment.**

12

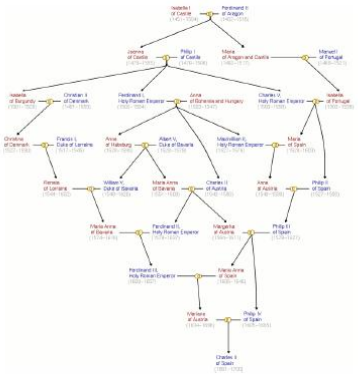
## 3rd Generation VCs

- Local check-in)merge is safe
- "Shelving" current work for quick fixes
- Version DAG: visual history
- History-aware merging: much smarter
- Fast: almost all operations are local
- Can be used offline
- Open source branching

13

Versioning example

Directed Acyclic Graph



14

## VC popularity

**RANKING OF SOURCE CONTROL TOOLS BY REGULAR USE, DEBIAN POPULARITY CONTEST (JAN '09), 70; 000 VOTES.**

| Tool | Rank | % of sample |
|---|---|---|
| Subversion | 655 | 10.47 |
| CVS | 944 | 5.5 |
| RCS | 1,565 | 2.1 |
| Mercurial | 2,256 | 1.03 |
| SVK | 3,672 | 0.37 |
| DARCS | 3,674 | 0.36 |
| Arch/TLA | 4,711 | 0.22 |
| Git | 6,673 | 0.11 |
| Monotone | 7,641 | 0.08 |
| Bazaar | 8,063 | 0.07 |
| Codeville | 16,456 | 0.01 |

15

## Integrated VCs

- Various programs help put a pretty face on VC, such as TortoiseCVS and TortoiseSVN (integrating VC into Windows), and Emacs and Eclipse (integrating VC into editing and development).
- Businesses often use commercial VC tools, such as IBM Rational **ClearCase, Microsoft VSS,** and **Borland StarTeam.**

16

## However,…

- *What large commercial tools do often offer is a workflow that is tightly integrated into a larger process model, of which version control is only a part.*
- Except for possibly **Bitkeeper, the revision** control portions of these systems tend to be quite outdated and much harder to use than necessary.

17

## A demo of subversion control system

- http://www.youtube.com/watch?v=8wYiabh2hpM&feature=related

18

# Build Control (BC)

19

## Introduction

- Build control tools like **make automate the process of** generating an executable or compiled version of a program or other file or document from source files
- Proper build control programs like **make are not the same** as scripting tools like Apple's Automator.

20

## Introduction (2)

- If your build process has 4,356 steps, and it is failing on step 3,963, **make** will start the build process at step 3,963 each time it runs, rather than doing the first 3,962 tasks over again from the beginning.
- This is the main advantage of something like **make** over simpler scripting or build control tools.

21

## Simple example

```
UNIX> make
cc -c file1.c
cc -c file2.c
cc -o a.out file1.o file2.o
UNIX>
```

22

## Other Build Control Systems

- Java's **ant is more portable in some sense, though it is not** as widely used. Supports Java much better than **make.**
- For more complicated projects needing to compile across many UNIX-like systems, consider **autoconf/automake.**

• Integrated development environments (IDEs) like Visual C++ usually replace makefiles with project files, but those are not as easily shareable to other systems.

23

## Debuggers

24

## Introduction

- Debuggers allow you to:
  - Step through code line by line
  - Set break points
  - Allow state to be examined or changed
- They are essential for C/C++, but extremely useful even for interpreted languages.

25

## Introduction (2)

- Many IDEs include integrated debuggers, but there are separate command-line debuggers: **gdb, jdb, pdb, etc.** and many graphical ones: **ddd, Insight, etc.**

26

## Unit/Regression Testing

27

## Introduction

- Unit regression testing frameworks make testing easy to do habitually, which makes everyone's life easier
- **JUnit was developed for XP on Java, based on a Smalltalk** original. It has been ported to many other languages: **pyunit (Python), CppUnit (C++), NUnit (.NET), etc.**
- There are also add-ons like **Coverlipse to check Java code** coverage, **QuickCheck to generate test cases in Haskell,** and **doctest to run tests embedded in doc strings.**

28

## Continuous Integration (e.g., Cruise Control)

29

## Introduction

- Tools like **Cruise Control support continuous integration** by initiating a new automated build and test run of your entire project every time a check-in is made:
- Picks up on source control events.
- Rebuilds project from scratch.
- Runs tests.
- Publishes status online, emails culprits.
- Great for multi-platform development: test all your target architectures at once.
- Needs 1 build server (well, ideally).

30

# Bug/Issue Tracking

31

## Introduction

- Any software package with a decent-sized user base will generate a lot of bug reports, complaints, and feature requests.
- Bug/issue tracking software keeps track of all of those for you; without it many fall through the cracks or end up dominating all your time and concentration.

32

## Introduction (2)

- There is no standard, but **Trac, GNATS, and BugZilla/IssueZilla are widely used free tools**
- **Also: Mantis ("less** confusing than Bugzilla"), **TestTrack Pro.**
- If your code is hosted at a site with an integrated configuration management package like **SourceForge**, it will come with bug/issue tracking.

33

## A demo over a bug issue tracking system with tickets

- http://www.youtube.com/watch?v=cQCmxLEGU6Y

34

## Documentation Generation

35

## Introduction

- No one actually writes documentation consistently, so it always gets out of sync with code. Same goes for comments.
- Documentation generation software like **JavaDoc (Java), NDoc (.NET), PHPDoc (PHP), and Doxygen (C++, C,** Java, etc.) automatically generates documentation from your source code and comments.

36

## Introduction (2)

- Most of it is guaranteed to be up to date, and if developers know their comments are going to be used as-is for the reference manual then they won't consider it wasted effort to update their comments when the code changes.

37

## Drawbacks reported

- Often the result is nearly unusable because it is repetitive, lacks context, and is missing crucial transitions between sections.
- It takes several passes between the source code comments and the generated output to make it all work ok.
- Note that only reference manuals can be generated; user manuals have to be written from scratch with the user in mind, and should never mirror the structure of the code.

38

## Integrated Suites

❑For open-source projects, integrated suites like **SourceForge and Google Code Hosting are freely** available that do all the above and add e.g.:

a) Document release management
b) Binary file release management
c) Web hosting

39

## Integrated Suites (2)

❑Proprietary workflow/process management tools like the IBM **Rational Unified Process suite are even more ambitious.**

❑Integrated packages are great if you need most of their features; separate tools can be used at any time.

40

## Summary

- Every sane person should be using version control and build control tools for everything — docs & code
- Unit/regression testing is good and much easier with the right framework
- Bug/issue tracking can stop you from going mad
- Documentation generation is great, but does not eliminate hand cleanup
- At least use these tools until you find something better ☺

41

## Recommended essential reading

- Software Engineering Principles and Practice, Hans van Vliet, John Wiley & Sons 2008, Chapter 15: Software Tools
- Understanding Version Control Systems

42

## More References

- Nice broad-ranging article on the latest generation of DVCS: "Distributed Version Control Systems: A Not-So-Quick Guide Through"
  - http://www.infoq.com/articles/dvcs-guide
- For a big (but old) list of project management tools for Linux, see
  - http://linas.org/linux/pm.html

43

NEXT WEEK

## The Design Principles and Patterns

44