Tools for Software Projects

Massimo Felici
Automating Drudgery

• Most of the techniques we will talk about can benefit from **automated tools**, and some would be totally impractical without them (e.g. the continual code changes in XP).

• Discussing all relevant tools is outside of the scope of this course. We will look at a few extremely useful categories, focusing on baseline open-source packages that everyone should be using unless their organisation has something better.

• We primarily consider useful **individual tools**, not integrated project management suites, because they are more widely applicable.
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
1st Step: Version Control (VC)

• 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.

• 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.
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

• Here JB had edited the file tools.tex, so he updated his copies to make sure they were current, checked the diffs, then committed his change.
• The new version of tools.tex is now 296, with the current date stamp.
Basic VC Features

- Have separate repository holding all versions of all files
- Changes in local copies do not affect the repository until a commit (a.k.a. checkin)
- 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
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 bugfixes allowed now)
  2. Release candidate (could ship if bug-free)
  3. Candidate sent to testing team
  4. Bugs are found, patched \( \Rightarrow \) new release candidate

- Typically there are different code branches maintained (at least production and development)
3 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: Revision Control System (RCS), Source Code Control System (SCCS)

1985 **Realization** — file merging works! No need for locking; just merge later. Allows multiple distributed copies to be worked on.
Examples: Concurrent Versions System (CVS), Subversion, Perforce

2002 **Realization** — repository merging works! DVCS (Distributed Version Control System): 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, Bazaar (bzr), Git, BitKeeper
Which VC tool to use?

- Surely everyone is already using VC for your own projects, even when not required? If not, set one up today and try using it for a few weeks.

- 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.
Advantages of DVCS

- Local check-in $\Rightarrow$ 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
Ancestry of King Charles II, the last Habsburg King of Spain. [source: AbsInt]
## VC popularity

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

Debian Popularity Contest (Jan ’09): ranking of source control tools by regular use, ∼ 70,000 votes.
Integrated VC

• Businesses often use commercial VC tools, such as IBM Rational ClearCase, Microsoft Visual SourceSafe (VSS) and Borland StarTeam.

• Except for possibly BitKeeper, the revision control portions of these systems tend to be quite outdated and much harder to use than necessary. 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.

• Various programs also 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).
Subversion in SAPM

• We will use Subversion to coordinate work between members of each team.

• It is not optional. If you have trouble getting it to work, we need to sort that out. There is no way to do well on either assignment without the whole team using Subversion.

• Subversion will allow each team member to submit work for the rest of the team to read, comment on, and edit.
Build Control

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:

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

Does everyone know how to use make or gmake? Does everyone actually use them or tools like them for your own projects even when not required to do so?
Build Control is not Scripting

- Proper build control programs like make are not the same as scripting tools like Apple’s Automator (drag-and-drop way to specify a series of actions to replay). They do not simply replay a series of actions, but instead encode a complex web of dependencies so that only the next appropriate actions are taken (based on datestamps).

- That is, 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.
Other Build Control Tools

- 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 make files with project files, but those are not as easily shareable to other systems.

- Note that build control is not just for compiling — it is for making releases, or for automating any other complex task.

  *Goal: do not do anything by hand more than 3 times...*
...except when you should!

Consider:

- The time you will save by automating a repeated task
- Whether automating might stop you, or someone else, making a mistake in the manual process
- The time you will lose by developing and maintaining the automation
- What is going to happen to the task in future.

*Automation invites complexity.* Sometimes better just to document the manual process (e.g. in a README file).
Debuggers

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

- Unit regression testing frameworks make testing easy to do habitually, which makes everyone’s life easier (and is required by XP, as we’ll see).

- 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 addons like Coverlipse to check Java code coverage, QuickCheck to generate test cases in Haskell, and doctest for Python to run tests embedded in doc strings.
Continuous Integration

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 $\geq 1$ build server (well, ideally).
Bug/Issue Tracking

• Any software package with a decent-sized userbase 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.

• There is no standard, but Trac (bug/issue tracking embedded in a wiki), GNATS, and BugZilla (new version supposed to be much better than before)/IssueZilla are widely used free tools; also: Mantis (‘less confusing than Bugzilla’), TestTrack Pro (Seapine).

• If your code is hosted at a site with an integrated configuration management package like SourceForge (discussed later), it will come with bug/issue tracking.
Documentation Generation

- 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.

- 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.
Generated Documentation Traps

- Even though generated documentation is often quite impressive looking, it is crucial for a human to go over it eventually to make sure it is also readable.

- 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.
Project Management

• Keeping track of all the sub-tasks in a large project, such as who they are assigned to, what order they have to be done, and how long each will take, is a pain.

• Project management software like Microsoft Project helps with this.
  Main features: automatic Gannt charts and PERT charts.
  Basecamp is a bit more ambitious, e.g. adding distributed time tracking.

• Also many free programs, such as: Planner, GanttProject, KPlato, etc.
Integrated Suites

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

- Document release management
- Binary file release management
- Web hosting

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.
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
Readings

Required Readings

- Fundamental Concepts, Chapter 1 of the SVN Book
- Martin Fowler, Continuous Integration

Suggested Readings

- Eric S. Raymond, Understanding Version-Control Systems — longer and more discursive article on version control
- Sebastien Auvray, Distributed Version Control Systems: A Not-So-Quick Guide Through — nice broad-ranging article on the latest generation of DVCS
- Call Center, Bug Tracking and Project Management Tools for Linux — for a big (but old) list of project management tools for Linux