Version Control

CS169

Lecture 7
Outline

• What is version control?
  – And why use it?
  – Scenarios
All Software Has Multiple Versions

• Different releases of a product

• Variations for different platforms
  • Hardware and software

• Versions within a development cycle
  • Test release with debugging code
  • Alpha, beta of final release

• Each time you edit a program
Version Control

• *Version control* tracks multiple versions

• In particular, allows
  - old versions to be recovered
  - multiple versions to exist simultaneously
Why Use Version Control?

• Because everyone does
  – A basic software development tool

• Because it is *useful*
  – You will want old/multiple versions
  – Without version control, can’t recreate project history
Scenario I: Bug Fix

Time

Releases

1.0

First public release of the hot new product
Scenario I: Bug Fix

Internal development continues, progressing to version 1.3
Scenario I: Bug Fix

A fatal bug is discovered in the product (1.0), but 1.3 is not stable enough to release. Solution: Create a version based on 1.0 with the bug fix.
Scenario I: Bug Fix

- Note that there are now two lines of development beginning at 1.0.
- This is branching.
Scenario I: Bug Fix

The bug fix should also be applied to the main code line so that the next product release has the fix.
Scenario I: Bug Fix

Note that two separate lines of development come back together in 1.4.

This is merging or updating.
Scenario II: Normal Development

You are in the middle of a project with three developers named a, b, and c.
Scenario II: Normal Development

At the beginning of the day everyone checks out a copy of the code.

A check out is a local working copy of a project, outside of the version control system. Logically it is a (special kind of) branch.
Scenario II: Normal Development

The local versions isolate the developers from each other’s possibly unstable changes. Each builds on 1.5, the most recent stable version.
Scenario II: Normal Development

At 4:00 pm everyone checks in their tested modifications. A check in is a kind of merge where local versions are copied back into the version control system.
In many organizations check in automatically runs a test suite against the result of the check in. If the tests fail the changes are not accepted.

This prevents a sloppy developer from causing all work to stop by, e.g., creating a version of the system that does not compile.
Scenario III: Debugging

Time  Releases

You develop a software system through several revisions.

1.5 → 1.6 → 1.7
Scenario III: Debugging

In 1.7 you suddenly discover a bug has crept into the system. When was it introduced?

With version control you can check out old versions of the system and see which revision introduced the bug.
Scenario IV: Libraries

You are building software on top of a third-party library, for which you have source.
Scenario IV: Libraries

You begin implementation of your software, including modifications to the library.
Scenario IV: Libraries

A new version of the library is released. Logically this is a branch: library development has proceeded independently of your own development.
Scenario IV: Libraries

You merge the new library into the main code line, thereby applying your modifications to the new library version.