Lecture 9 — Version Control

Jeff Zarnett & Patrick Lam
jzarnett@uwaterloo.ca & p.lam@ece.uwaterloo.ca

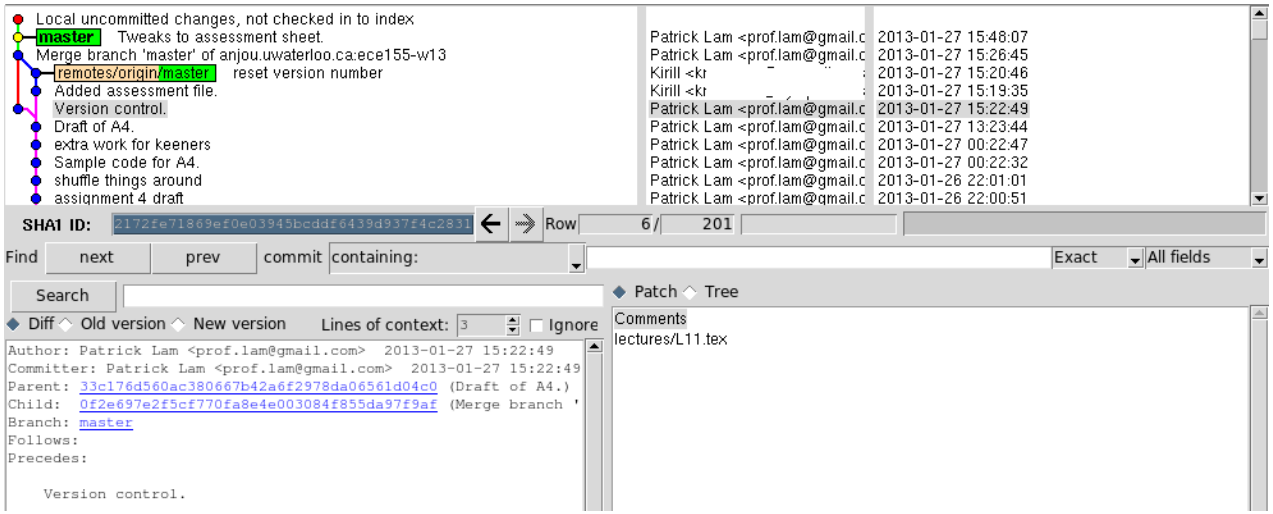
Department of Electrical and Computer Engineering
University of Waterloo

December 29, 2014

Ever wanted to undo your changes to software?

Ever needed to collaborate with others to develop software?

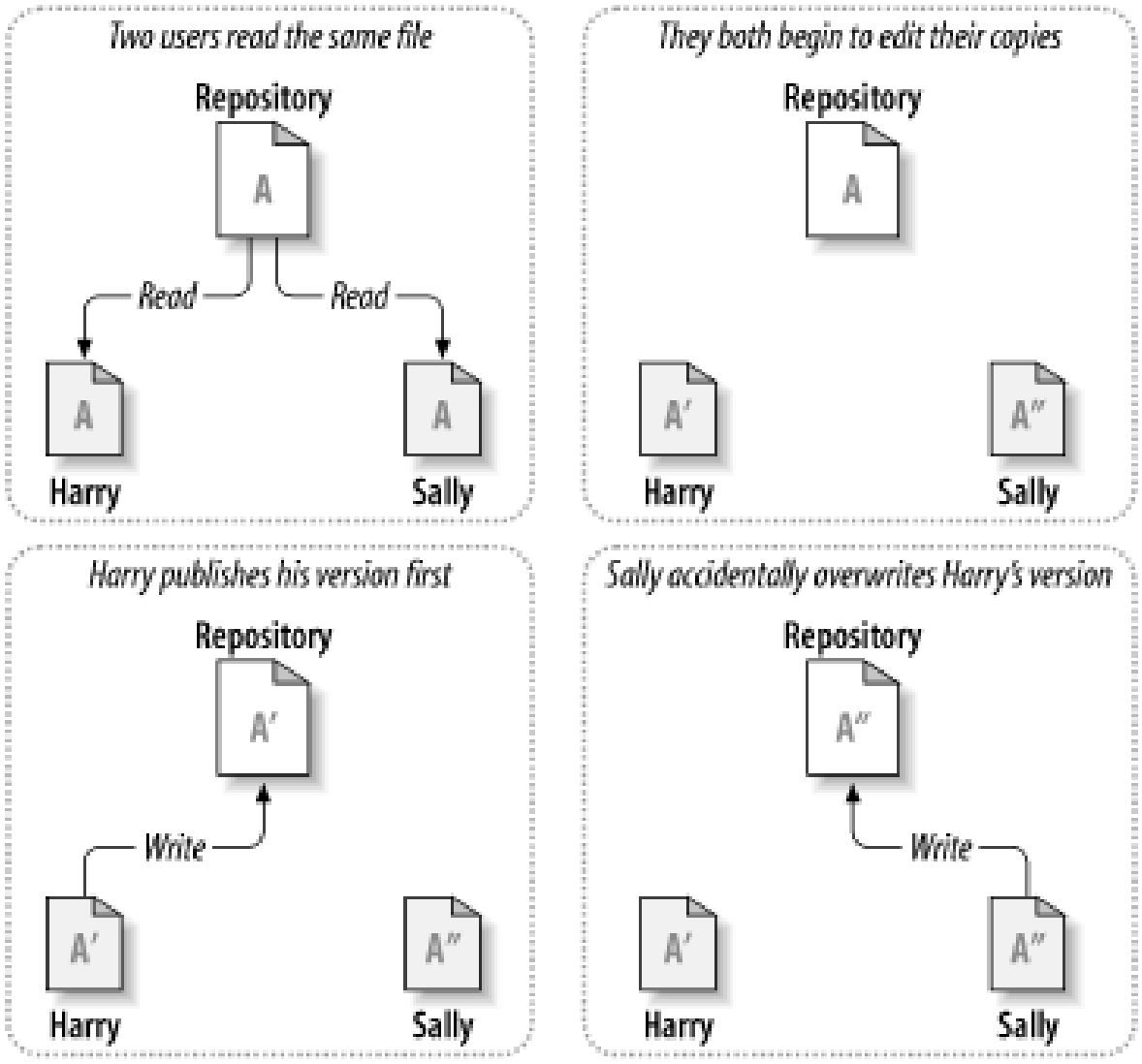
Conceptual Idea



- Store a repository of revisions.

  Each revision is a snapshot of a set of files.
- Can search by author, date, commit comment.
- Can revert to previous revisions.

Without Version Control



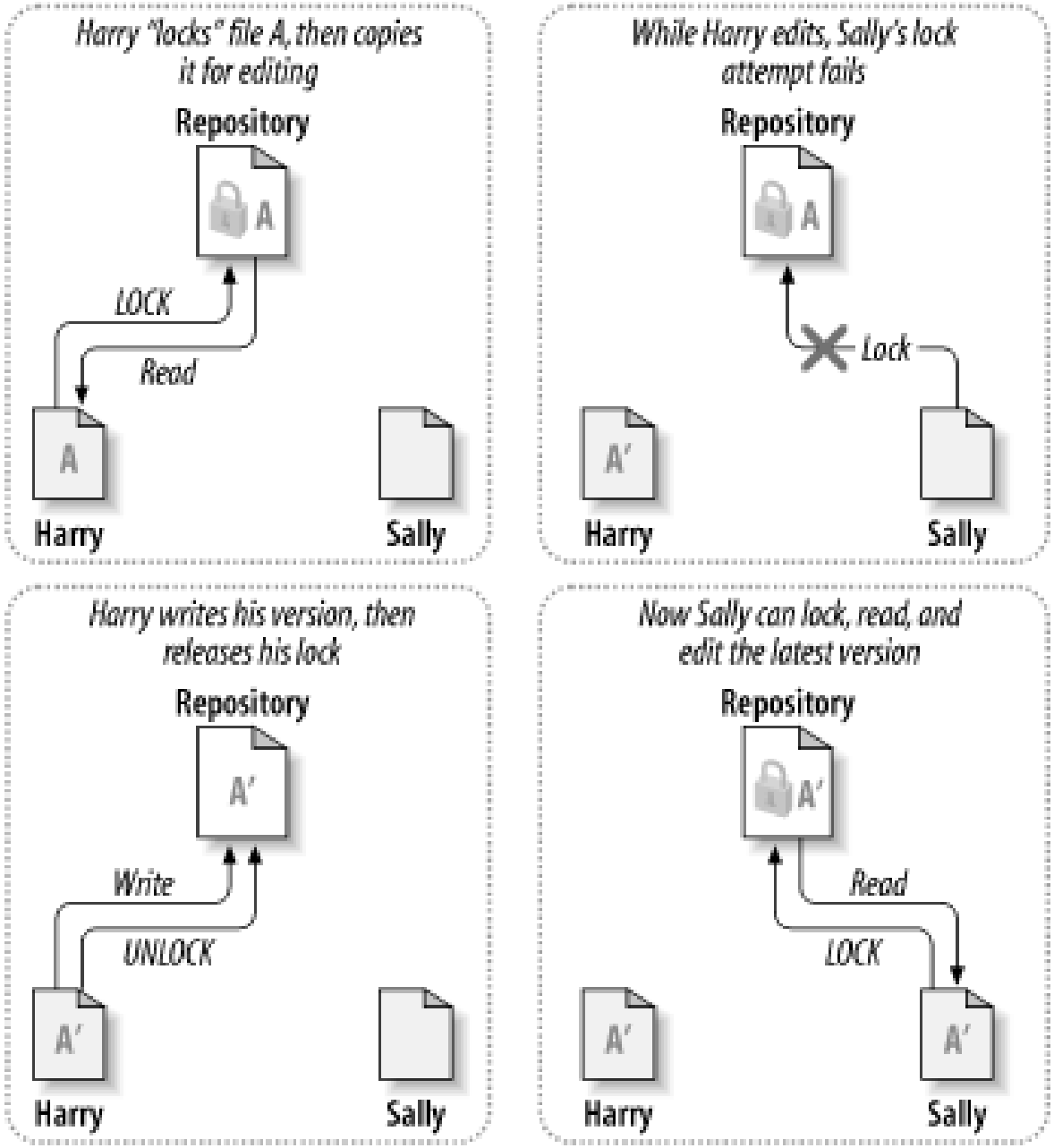
http://svnbook.red-bean.com/en/1.6/svn.basic.version-control-basics.html

Lock-Modify-Unlock

The first model: Lock-Modify-Unlock

Considered obsolete, but worth learning about.

Lock-Modify-Unlock



Problems with Lock-Modify-Unlock

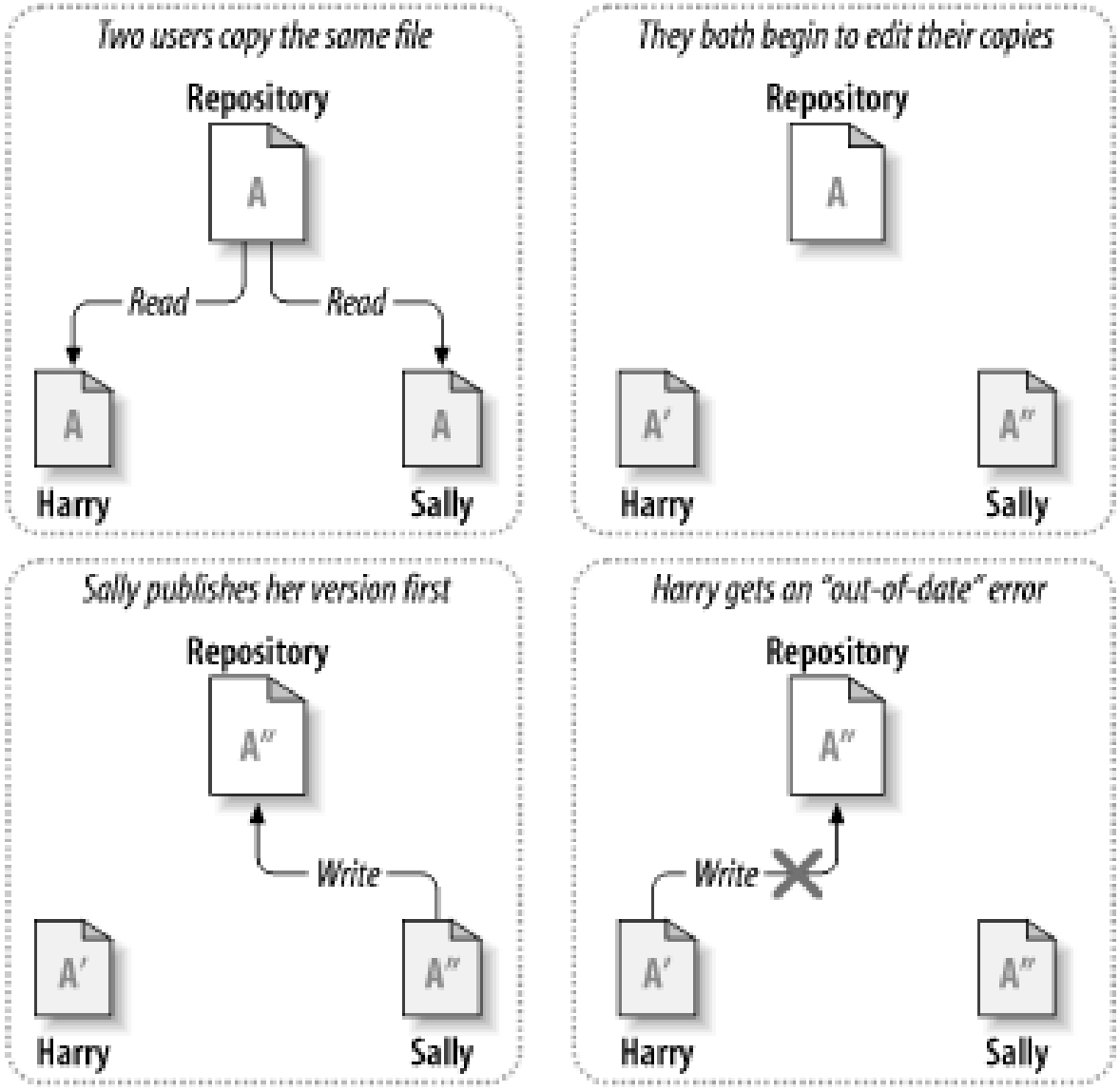
- Forgot to Unlock.
- Unnecessary Waiting
- Deadlock
- Parallel Modification

Copy-Modify-Merge

The current model: Copy-Modify-Merge

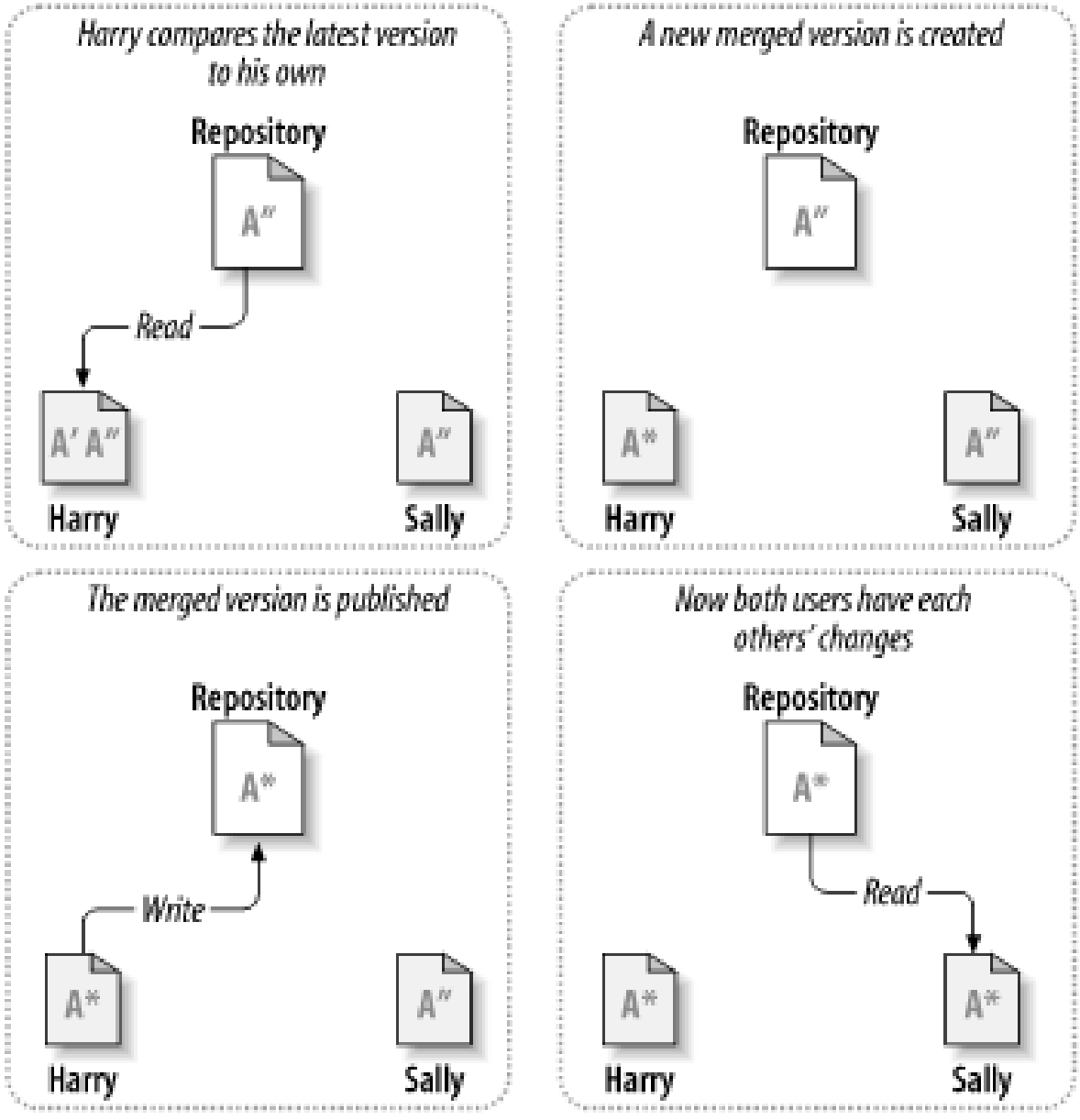
Yes, merging works. Have you tried it?

Copy-Modify-Merge



http://svnbook.red-bean.com/en/1.6/svn.basic.version-control-basics.html

Copy-Modify-Merge



http://svnbook.red-bean.com/en/1.6/svn.basic.version-control-basics.html

Merging

Usually succeeds automatically.

Sometimes you will have to solve a conflict manually.

Consider: the common ancestor, your change, and the other change.

Advantage of C-M-M: More parallelism. Conflicts infrequent.

CVS

One of the first version control systems: cvs

Developed in the 1980s; mature technology

Introduced the concept of branches.

Branches split off from the trunk (mainline).

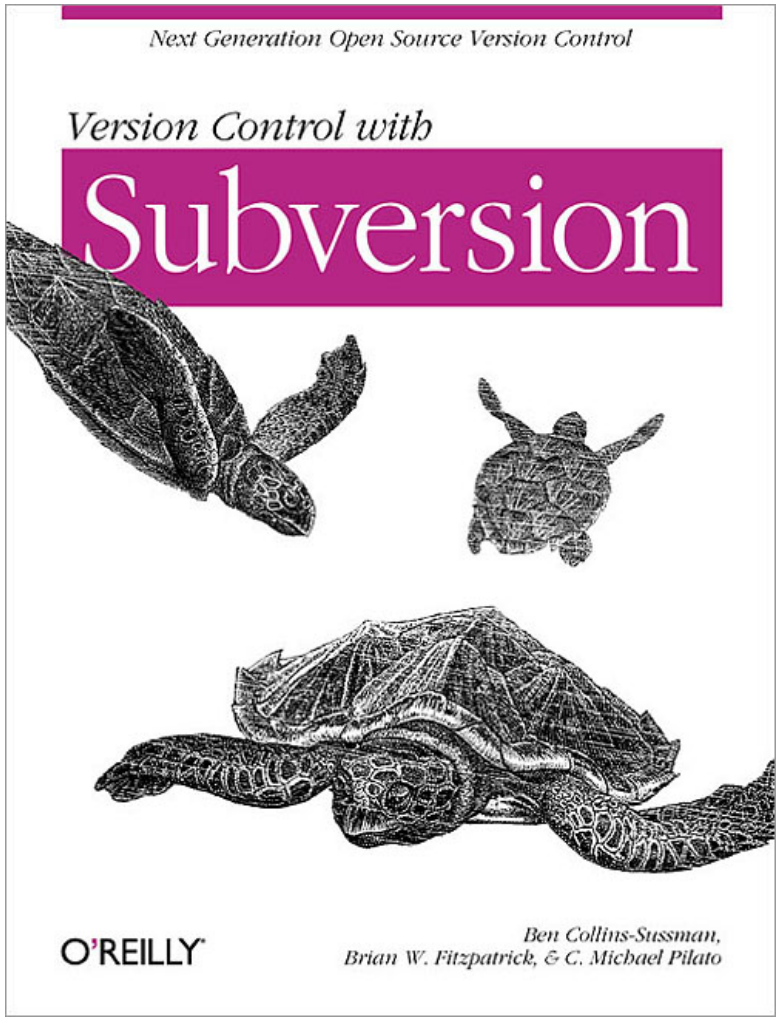
CVS: Shortcomings

- No Moving/Renaming Support
- Branches Expensive
- Commits Not Atomic

From CVS to SVN

Attempts to address these problems led to: Subversion (svn).

Case Study: Subversion



(`http://svnbook.red-bean.com/`)

You are using Subclipse; I'll talk about command-line usage.

Creating a new repository

Create one from scratch:

```
svnadmin create c:\svn\repos
```

More commonly, check out a repository:

```
svn checkout http://k9mail.googlecode.com/svn/k9mail/trunk/
    k9mail-read-only
```

- creates a working copy. (You've done this.)

Adding and Ignoring Files

You've seen how to add files to the repository
(Team > Add to Version Control).
command-line: `svn add filename`

- failure to add files: leading cause of build breakage

You've seen ignored files like `R.java`.

- Generally, do not commit generated files!

Instead, tell Subversion to ignore them,
e.g. using wildcards.

Committing Files

On SVN, a commit makes your changes available to the world.
In decentralized version control, a commit records current version.

When to commit?

- What you commit must compile!
- Generally, one feature at a time, after testing. (Varies by source control system.)

Commit Messages

- An important form of project documentation[1].

Start with a one-line summary.

Establish the specific context of the change:

- Why is it necessary?
- How does it work?
- What are the effects?

Meta-commit message[2]:

```
Summarize clearly in one line what the commit is about

Describe the problem the commit solves or the use
case for a new feature. Justify why you chose
the particular solution.
```

[1] http://who-t.blogspot.ca/2009/12/on-commit-messages.html, accessed 27Jan13

[2] https://github.com/erlang/otp/wiki/Writing-good-commit-messages, accessed 27Jan13

Updating your repository

Pull changes from the repository to your working copy.

Use `svn update` to do that. If all goes well, you'll get output like this:

```
plam@noether:~/production/11.aosd.modularity$ svn up
D    example.tex
A    studies.tex
U    introduction.tex
A    sketch.tex
U    main.tex
```

Conflicts: the bane of your existence

This is a pain:

```
plam@noether:~/production/11.aosd.modularity$ svn up
C    example.tex
```

Why?

- Both the latest version and your version differ from the common ancestor.

Example conflict

How?

1. I wrote: “Here’s a line of text”.
2. Programmer X changes it to “Here’s a line of text that I modified.”
3. I change it again to “Here’s a modified line of text.”

The result:

```
<<<<<<< HEAD
Here’s a line of text that I modified.
=======
Here’s a modified line of text.
>>>>>>> zzz
```

You need to fix it and tell SVN that you’ve fixed it.

Stepping Back in Time

Major win of version control:

- can undo sketchy changes.

Can update to a past revision number or a date/time.

How to know which version to revert to?

- your detailed log messages!

Note: you can't commit an update, but you can merge it to your working copy.

Diffs

Basic unit of version control is the diff:

- describes what's different between two versions.

Inspect your diffs before committing. Commit minimal diffs.

```
===================================================================
--- Text/abstract.tex (revision 17379)
+++ Text/abstract.tex (working copy)
@@ -1,10 +1,10 @@
 Runtime monitoring enables developers to (1) specify important program
 properties and (2) dynamically validate that these properties hold.
 In recent research, we have found that static analysis techniques can,
-in many cases, verify that runtime monitors never trigger.  In
-this paper, we describe a system which enables developers to visualize
-the remaining cases---potential
+in many cases, verify that runtime monitors never trigger.  In this
+paper, we describe a tool which enables developers to visualize the
+remaining cases---potential points
```

Basic SVN Workflow

Repeat until done:

- Update your working copy.
  (`svn update`)
- Edit files. Manipulate tracked files.
  (`svn add`, `svn rm`, `svn copy`, `svn move`)
- Examine changes.
  (`svn status`, `svn diff`)
- Undo changes, if necessary.
  (`svn revert`)
- Commit changes to the server.
  (`svn commit`)

SVN vs CVS

SVN was specifically intended to address the shortcomings of CVS. Some properties:

- Atomic Commits
- Branch Operations
- Support for Moving or Renaming

SVN vs CVS

SVN was specifically intended to address the shortcomings of CVS. Some properties:

- Atomic Commits
- Branch Operations
- Support for Moving or Renaming

Issues with SVN

Subversion is better than CVS, but it still has some problems:

- Centralized
- Slow

Centralized vs. Distributed

Traditionally, there was one canonical central repository for a software project.

Centralized systems work on that model.

Newer version control systems can be decentralized.

Git and Mercurial

One of those decentralized systems is `git`.

Designed to be a new model, not just “better” `svn`.

Created by Linus Torvalds (yes, of the Linux Kernel).

Decentralized – can view everything offline.

Fast!

Branch operations inexpensive and recommended workflow.

Similar to svn: add, ignore, commit, update, merge, resolve, etc.

Complex (steep learning curve)

Git Workflow

The basic work cycle of git is the following:

- Update working copy of the repository (`git pull`).
- Create a feature branch (`git checkout -b`).
- Edit files. Manipulate set of files with `git add`, `git rm`.

Git Workflow

- Examine changes (`git status`, `git diff`).
- Undo changes (`git checkout`, `git reset`).
- Commit changes (`git commit`).
- Merge your feature branch into the parent branch:
  - Check the parent branch out (`git checkout`)
  - `git pull` to update
  - merge (`git merge`).
- Finally, share your changes with others using `git push`.

Mercurial

About Mercurial (`hg`)

Distributed like `git`.

Easier to learn; similar to `svn`.

Better Windows support.

Can be used alongside git on the same repository.