



# Today's Agenda

---

- File Permissions
- Pipes example
- Version Control Introduction
- Lab Assignment #1 (@ appx 10:15/10:20 )

# **CSCI 414 / 553: Networking III**

## **Unix Network Programming**

Spring 2008



---

Version Control



# Introduction

---

- Five things distinguish professional programmers from amateurs:
  - Using a version control system
  - Automating repetitive tasks
  - Systematic testing
  - Using debugging aids rather than print statements
  - Advanced familiarity with a Programming Editor
- This lecture introduces the first of these
  - Even if it's the only thing you take away from this course, you'll be more productive than you are now



# Problem #1: Collaboration

---

- What if two or more people want to edit the same file at the same time?
- Option 1: make them take turns
  - But then only one person can be working at any time
  - And how do you enforce the rule?
- Option 2: patch up differences afterwards
  - Requires a lot of re-working
  - Stuff always gets lost

# Solution: Version Control

- The right solution is to use a *version control system*
- Keep the master copy of the file in a central *repository*
- Each author edits a *working copy*
- When they're ready to share their changes, they *commit* them to the repository
- Other people can then do an *update* to get those changes
- This is also a good way for one person to manage files on multiple machines
  - Keep one working copy on your personal laptop, the lab machine, and the departmental server
  - No more mailing yourself files, or carrying around a USB drive (and forgetting to copy things onto it)

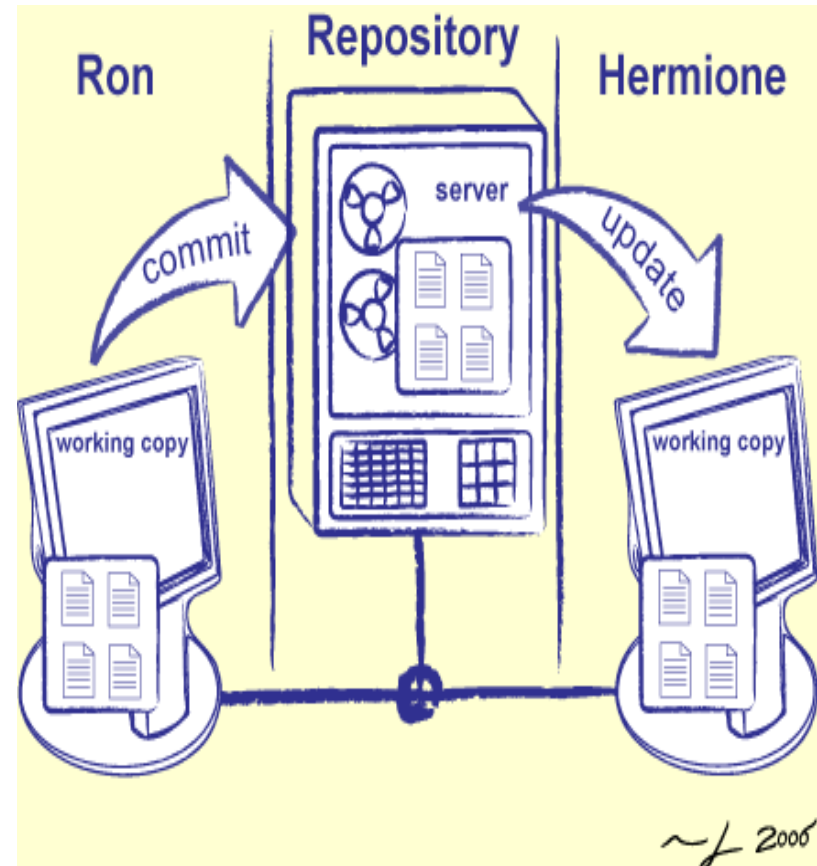


Fig L4.1: Managing Multi-Author Collaboration



# Problem #2: Undoing Changes

---

- Often want to undo changes to a file
  - Start work, realize it's the wrong approach, want to get back to starting point
  - Like “undo” in an editor...
  - ...but keep the whole history of every file, forever
- Also want to be able to see who changed what, when
  - The best way to find out how something works is often to ask the person who wrote it

# Solution: Version Control (Again)

- Have the version control system keep old *revisions* of files
  - And have it record who made the change, and when
- Authors can then *roll back* to a particular revision or time

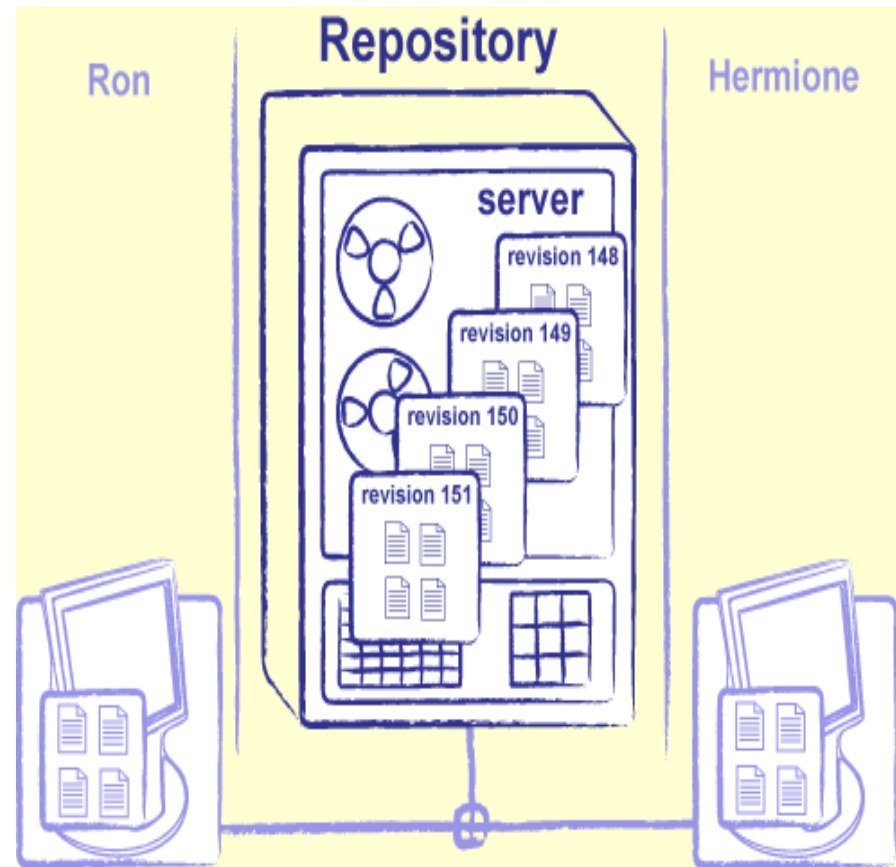


Fig L4.2: Version Control as a Time Machine

# Which Version Control System?



---

- Many systems are available commercially
  - If you have a large group, or a budget, **Perforce** is excellent
- **CVS** and **Subversion** are:
  - Open source
  - Reliable
  - Well documented
- **CVS** has been around since the 1980s
  - Very popular, but showing its age
  - Flaw #1: it keeps track of each file separately
    - So there's no reliable way to ask, "Which files were changed together?"
  - Flaw #2: you can create new directories, but can't delete old ones
- **Subversion** developed from 2000 onward as a workalike replacement
  - Feels the same, but eliminates CVS's major weaknesses
  - Many projects have already switched



# Basic Use

---

- Ron and Hermione each has a working copy of the solarsystem project repository
  - See below how they got it
  - Ron wants to add some information about Jupiter's moons
  - Runs svn update to synchronize his working copy with the repository
  - Goes into the jupiter directory and creates moons.txt

Name	OrbitalRadius	OrbitalPeriod	Mass	Radius
Io	421.6	1.769138	893.2	1821.6
Europa	670.9	3.551181	480.0	1560.8
Ganymede	1070.4	7.154553	1481.9	2631.2
Callisto	1882.7	16.689018	1075.9	2410.3

- Runs svn add moons.txt to bring it to **Subversion**'s notice
  - Runs svn commit to save his changes in the repository
    - Repository is now at revision 121
- That afternoon, Hermione runs svn update on her working copy
  - **Subversion** sends her Ron's changes

# Basic Use

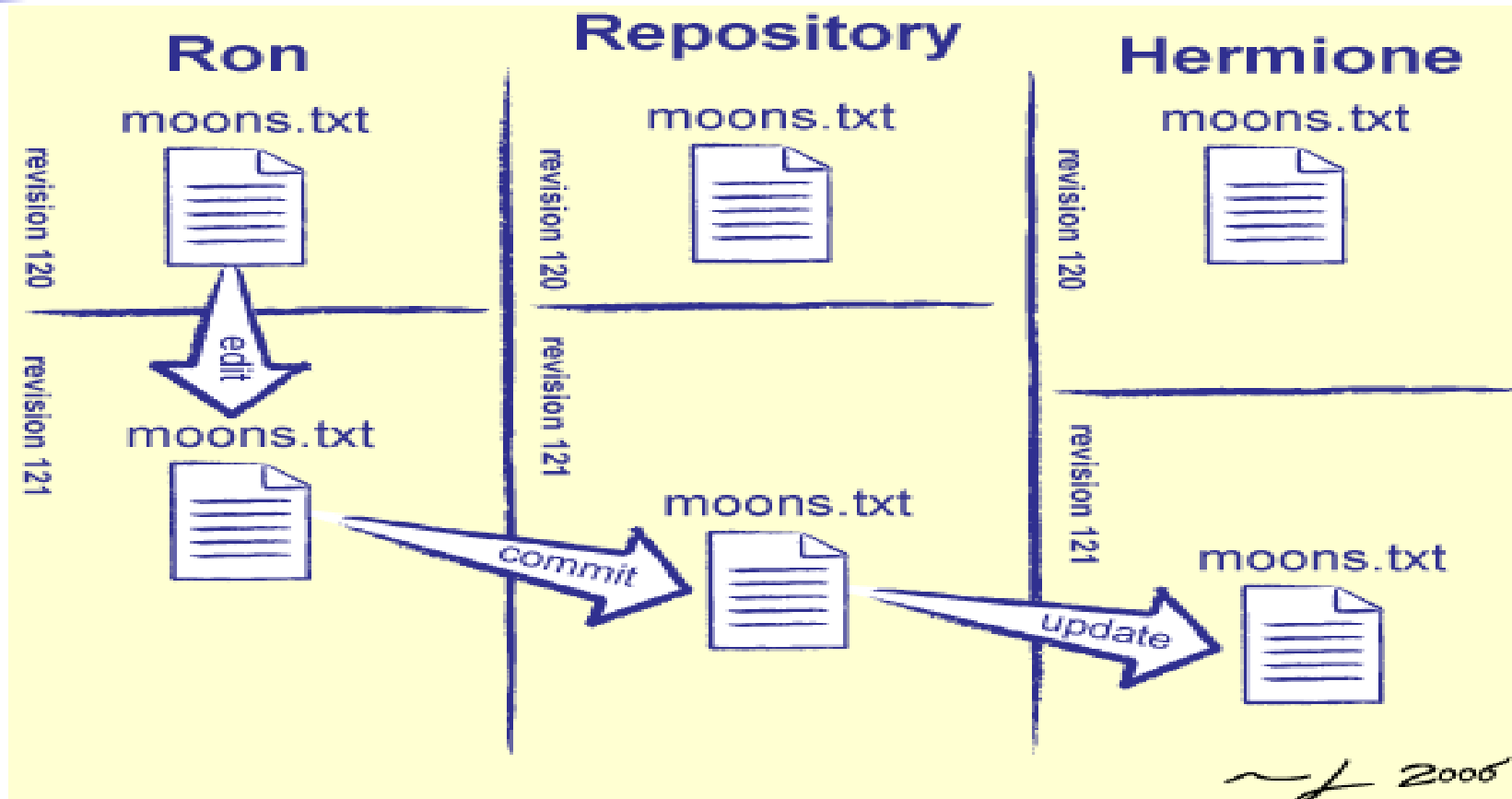


Fig L4.3 The Basic Edit/Update Cycle

# How To Do It

- One way to use Subversion is to type commands in a shell
  - A lowest common denominator that will work almost everywhere
- **RapidSVN** is a GUI that runs on Windows, Linux, and Mac
  - Well, maybe “walks” is a better description—Version 0.9 isn't particularly fast

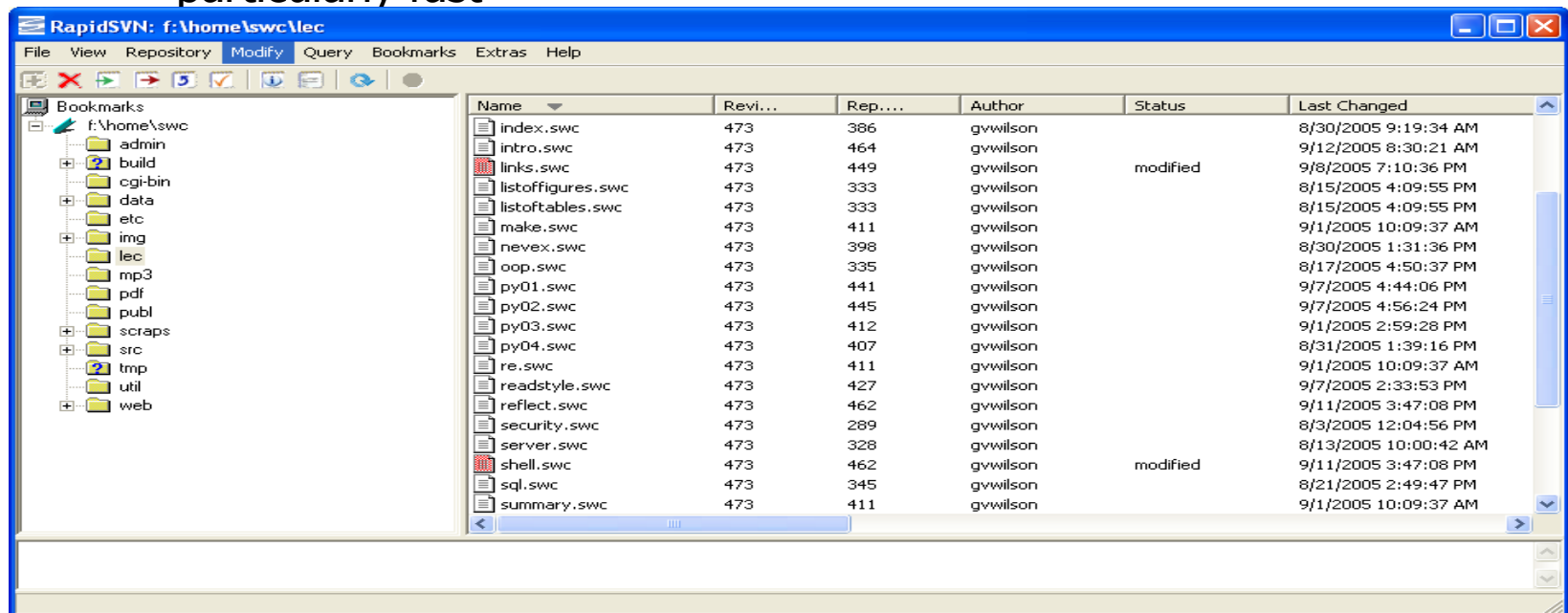


Fig L4.4 RapidSVN

# How To Do It

- **TortoiseSVN** is a Windows shell extension
  - Integrates with the file browser, rather than running separately

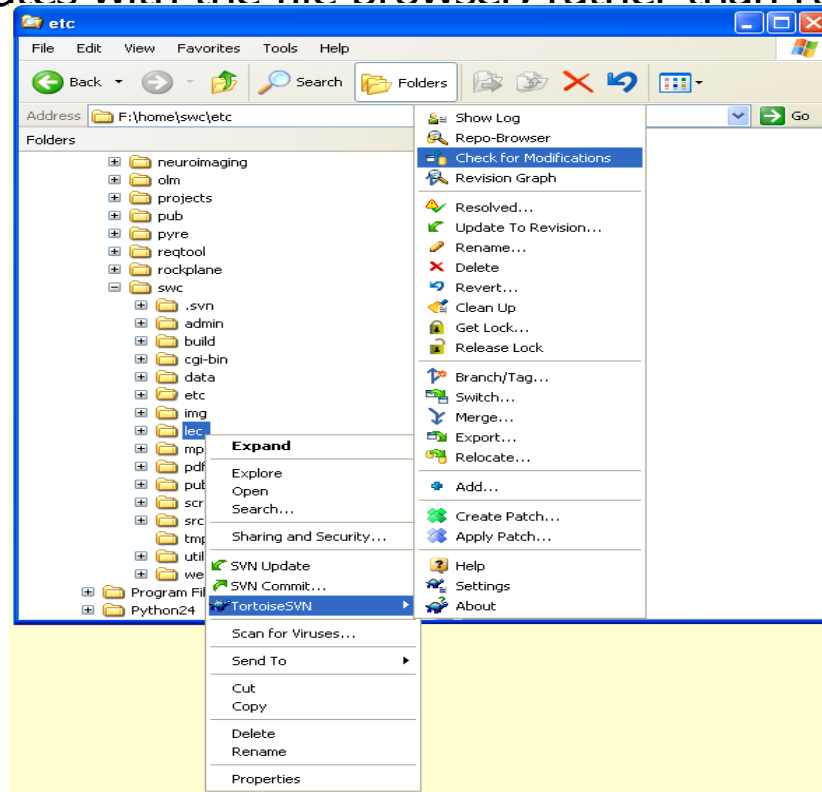


Fig L4.5 TortoiseSVN



# Resolving Conflicts

---

- Back to the problem of *conflicting edits* (or, more simply, conflicts)
- Option 1: only allow one person to have a writeable copy at any time
  - This is called *pessimistic concurrency*
  - Used in Microsoft Visual SourceSafe
- Option 2: let people edit, and *resolve* conflicts afterward by *merging* files
  - Called *optimistic concurrency*
  - “It's easier to get forgiveness than permission”
  - Most modern systems (including **Subversion**) do this



# Example of Resolving

---

- Ron and Hermione are both synchronized with version 151 of the repository
- ~~Ron edits moons.txt and commits his changes to create version 152~~

Name	OrbitalRadius	OrbitalPeriod	Mass	Radius
Io	421.6	1.769138	893.2	1821.6
Europa	670.9	3.551181	480.0	1560.8
Ganymede	1070.4	7.154553	1481.9	2631.2
Callisto	1882.7	16.689018	1075.9	2410.3
Amalthea	181.4	0.498179	0.075	131x73x67
Himalia	11460	250.5662	0.095	85
Elara	11740	259.6528	0.008	40



# Example of Resolving

---

- Simultaneously, Hermione edits her copy of moons.txt

Name	OrbitalRadius (10**3)	OrbitalPeriod (days)	Mass (10**20kg)	Radius (km)
Io	421.6	1.769138	893.2	1821.6
Europa	670.9	3.551181	480.0	1560.8
Ganymede	1070.4	7.154553	1481.9	2631.2
Callisto	1882.7	16.689018	1075.9	2410.3
Amalthea	181.4	0.498179	0.075	131
Himalia	11460	250.5662	0.095	85
Elara	11740	259.6528	0.008	40
Pasiphae	23620	743.6	0.003	18
Sinope	23940	758.9	0.0008	14
Lysithea	11720	259.22	0.0008	12

# Example of Resolving

- When she tries to commit, **Subversion** tells her there's a conflict
  - A *race condition*: two or more would-be writers racing to get their changes in first

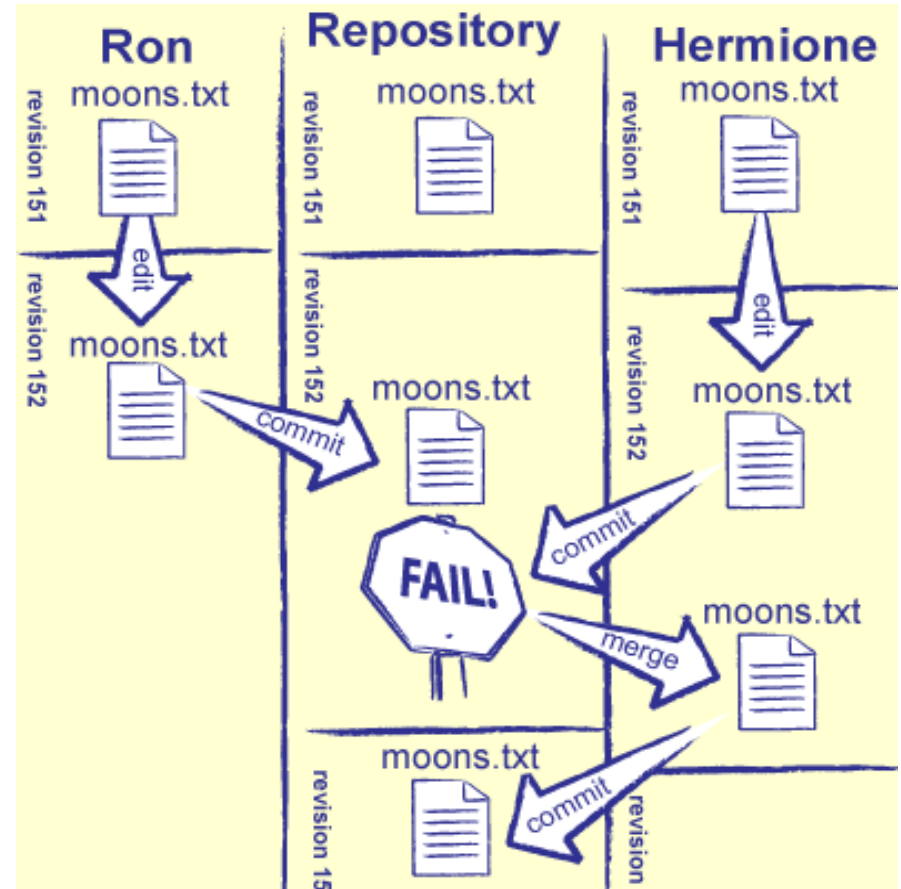


Fig L4.6: Merging Conflicts

# Example of Resolving (continued)

- **Subversion** puts Hermione's changes and Ron's in moons.txt
  - Adds *conflict markers* to show where they overlapped

```
Name      OrbitalRadius      OrbitalPeriod      Mass      Radius
          (10**3 km)        (days)             (10**20kg) (km)
Io         421.6              1.769138            893.2     1821.6
Europa    670.9              3.551181            480.0     1560.8
Ganymede  1070.4             7.154553            1481.9    2631.2
Callisto  1882.7             16.689018           1075.9    2410.3
<<<<<<< .mine
Amalthea  181.4              0.498179            0.075     131
Himalia   11460              250.5662            0.095     85
Elara     11740              259.6528            0.008     40
Pasiphae  23620              743.6               0.003     18
Sinope    23940              758.9               0.0008    14
Lysithea  11720              259.22              0.0008    12
=====
Amalthea  181.4              0.498179            0.075     131x73x67
Himalia   11460              250.5662            0.095     85
Elara     11740              259.6528            0.008     40
>>>>>> .r152
```

- <<<<<<< shows the start of the section from the first file
- ===== divides sections
- >>>>>> shows the end of the section from the second file
- **Subversion** also creates:
  - moons.txt.mine: contains Hermione's changes
  - moons.txt.151: the file before either set of changes
  - moons.txt.152: the most recent version of the file in the repository



# Example of Resolving (continued)

---

- At this point, Hermione can:
  - Run `svn revert moons.txt` to throw away her changes
  - Copy one of the three temporary files on top of `moons.txt`
  - Edit `moons.txt` to remove the conflict markers
- Once she's done, she runs:
  - `svn resolved moons.txt` to let **Subversion** know she's done
  - `svn commit` to commit her changes (creating version 153 of the repository)



# Starvation

---

- But what happens if Ginny commits another set of changes while Hermione is resolving?
  - And then Harry commits yet another set?
- *Starvation*: Hermione never gets a turn because someone else always gets there first
- This is a management problem, not a technical one
  - Break the file(s) up into smaller pieces
  - Give people clearer responsibilities
  - The version control system is trying to tell you that people on your team are working at cross purposes



# Binary Files

---

- Subversion can only merge conflicts in text files
  - Source code, HTML—basically, anything you can edit with Notepad, Vi, or Emacs
- But images, video clips, Microsoft Word, and other formats aren't plain text
  - When there's a conflict, Subversion saves your copy and the master copy side by side in your working directory
  - Up to you to resolve the differences
- It's not Subversion's fault
  - Most creators of non-text formats don't provide a way to find or merge differences between files



# Reverting

---

- After doing some more work, Ron decides he's on the wrong path
- svn diff shows him which files he has changed, and what those changes are
- He hasn't committed anything yet, so he uses svn revert to discard his work
  - I.e., throw away any differences between his working copy and the master as it was when he started
  - Synchronizes with where he was, *not* with any changes other people have made since then
- If you find yourself reverting repeatedly, you should probably go and do something else for a while...

# Rolling Back

- Now Ron decides that he doesn't like the changes Harry just made to moons.txt
  - Wants to do the equivalent of "undo"
- svn log shows recent history
  - Current revision is 157
  - He wants to revert to revision 156
- `svn merge -r 157:156 moons.txt` will do the trick
  - The argument to the `-r` flag specifies the revisions involved
  - Merging allows him to keep some of Harry's changes if he wants to
  - Revision 157 is still in the repository

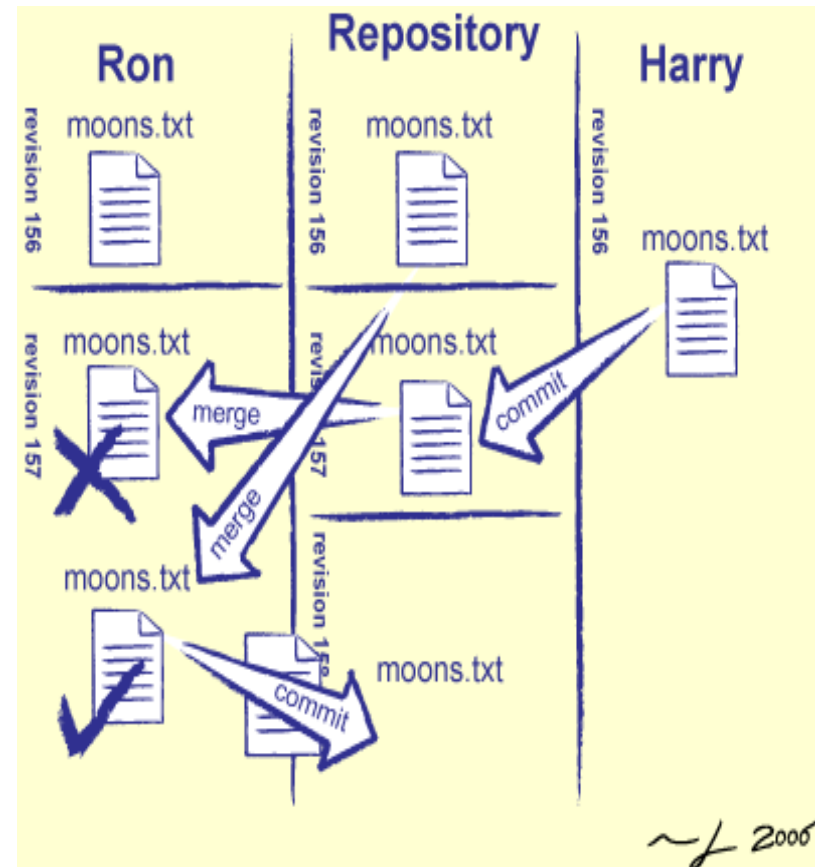


Fig L4.7: Rolling Back

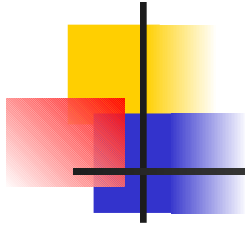


# Creating and Checking Out

---

- To create a repository:
  - Decide where to put it (e.g., /svn/rotor)
  - Go into the containing directory: `cd /svn`
  - `svnadmin create rotor`
- Can then check out a working copy
  - Directly through the file system: `svn checkout file:///svn/rotor`
  - Through a web server: `svn checkout http://www.hogwarts.edu/svn/rotor`
    - Note: requires your system administrator to configure the web server properly
- Only use `svn checkout` once, to initialize your working copy
  - After that, use `svn update` in that directory
- If you only want part of the repository, use `svn co http://www.hogwarts.edu/svn/rotor/engine/dynamics`

# Subversion Command Reference



Name	Purpose	Example
* <b>svn add</b>	Add files and/or directories to version control.	svn add newfile.c newdir
* <b>svn checkout (co)</b>	Get a fresh working copy of a repository	svn checkout <a href="https://your.host.name/roto/repo">https://your.host.name/roto/repo</a> rotorproject
* <b>svn commit (ci)</b>	Send changes from working copy to repository (inverse of update)	svn commit -m "Comment on the changes"
* <b>svn delete (del, rm, remove)</b>	Delete files and/or directories from version control.	svn delete oldfile.c
<b>svn help</b>	Get help (in general, or for a particular command).	svn help update
<b>svn merge</b>	Merge two different versions of a file into one.	svn merge -r 18:16 spin.c
* <b>svn rename (mv, move)</b>	Rename a file or directory, keeping track of history.	svn rename temp.txt release_notes.txt
<b>svn resolve</b>	Let subversion know that a conflict has been resolved	svn resolve planets.txt
<b>svn revert</b>	Undo changes to working copy (i.e. resynchronize with repository).	svn revert spin.h
<b>svn status</b>	Show the status of files and directories in the working copy.	svn status
<b>svn update</b>	Bring changes from repository into working copy (inverse of commit).	svn update



# Reading Subversion Output

---

- `svn status` compares your working copy with the repository
  - Prints one line for each file that's worth talking about

```
$ svn status
M jupiter/moons.txt
C readme.txt
```
  - `jupiter/moons.txt` has been modified
  - `readme.txt` has conflicts
- `svn update` prints one line for each file or directory it does something to

```
$ svn update
A saturn/moons.txt
U mars/mars.txt
```

  - `saturn/moons.txt` has been added
  - `mars/mars.txt` has been updated (i.e., someone else modified it)



# Summary

---

- Version control is one of the things that distinguishes professionals from amateurs
  - And successful projects from failures
- Everything that a human being had to create should be under version control
- You'll see the benefits almost immediately



# Today's Agenda

---

- Version Control Introduction
- Lab Assignment #1 (@ 10:20 )