A Crash Course in Subversion
and
Python Documentation

David Arbour
Cmpsci 320
Roadmap

- SVN and Version Control
- PyDoc and Documentation
- The rest of class....
What is version control?

- A tool for collaborative software development
- Can come in the form of Client/Server architecture such as CVS and SVN or distributed such as Git and Mercurial
- All forms allow for a way for multiple developers to have local, independent copies of some shared piece of code that can be integrated in at the appropriate time.
- Allows for developers to work on a piece of software concurrently and resolve conflicts that may occur
Subversion

- Client/Server architecture
- Open Source, an Apache Foundation project
- Widely used in industry and the open source community
- Relatively simple to use either from the command line or with a readily available Eclipse plugin, subclipse
  - Command line client can be found here: [http://subversion.apache.org/packages.html](http://subversion.apache.org/packages.html)
  - Subclipse can be found here: [http://subclipse.tigris.org/](http://subclipse.tigris.org/)
  - TortoiseSVN: [http://tortoisesvn.net/](http://tortoisesvn.net/)
  - Other good modern IDEs such as IntelliJ and Netbeans offer SVN support as well
What We'll Cover

There are a few concepts that you should be familiar with:

- Checking Out
- Adding/Deleting
- Checking the Status
- Updating
- Commiting
- Merging

Hopefully by the time we're done you'll feel comfortable with all of them.
Checking Out

- Allows you to have a local copy of the repository on your machine.
- From the command line, cd to the directory you'd like to have the code in and type in "svn --username[your username]co [url of repository]"
  - For this course the url is svn://elnux7.cs.umass.edu:42320
  - Your username and password will be the same as your edlab account login.
- From Eclipse:
  - New -> Project -> Checkout Projects From SVN
Local vs. Committed

- Local files can be copies of files that are in the repository or your own files
  - The contents of local files are **not** reflected in the repository
- Committed files' content are reflected in the repository
  - Others will have a copy of that file when they checkout or update
Adding/Deleting

- To add a file to the project use the command "svn add [file pattern]" from the svn directory.
  - For example, I have created "HelloWorld.java" to add this file to the repository (assuming the file was created in the directory of the svn project) I would enter "svn add HelloWorld.java"
- To delete a file from the project use "svn delete [filepattern]"
- Right click the file from Eclipse to add/delete files from svn
- You must use svn add/delete to have the file be recognized by svn
- These files will not affect the central repository until the changes are committed
Checking The Status

- This will show you the status of your repository (what has been added/deleted and not committed)
- A good way to check whether a commit is necessary
- Eclipse will show you this information through the GUI
- From the command line "svn status" from the project directory
Update

- Update synchronizes your local copy to the last committed version of the project
- A good idea whenever you sit down to work (especially when there are a lot of changes) so you avoid conflicts and merging
- From command line "svn update" all updated files will then be displayed
- From Eclipse: Team -> Update
Committing

- Attempts to update the files in the repository with your modified files
- Some best practices:
  - Include a meaningful message with every commit
  - Update before commit, this will help prevent conflicts
  - Avoid committing "broken" code if others are depending on it
    - This is when branches are useful, not necessary for this class but a good best-practice
Conflicts

- What happens when two people try to edit the same copy of a file in different ways?
- Commit won't work, for example:
  - `$ svn commit -m "Fixed a massive amounts of bugs and made project amazing" Sending Main.py Transmitting file data .svn: Commit failed (details follow): svn: Out of date: '/RockStar/Main.py'
- Someone has changed the file you are working on between commits!
- Luckily, we can deal with this using merge
Merging

- Merging allows you to choose which file's contents to keep for each conflicting chunk
  - For Example:
    - A: "Mary had a little lamb"
    - B: "Mary is a little lamb"
  - Merging allows you to choose to keep either A or B and place that version in the repository
- You *can* do this from the command line, but since it's so tedious consider using one of many available tools
  - Included in any half-decent GUI-based SVN Client such as:
    - Subclipse for Eclipse
    - Tortoise
  - Additionally there are standalone tools such as KDiff3, for linux, that are quite good
General Merge Flow

- You will see three columns
  - One is your copy of the file
  - One is the file that your file conflicts with
  - The third is the 'resolved' file
- You go through each merge and choose which version to keep
  - You can simplify this by using 'use all of mine' or 'use all of theirs' but this isn't usually wise for obvious reasons
- After you have gone through the whole file and chosen what to keep the file is 'resolved' and you are ready to commit
Recap of the SVN Workflow

Checkout
Add/Edit
Look at diffs
Update
Resolve
Commit
Repeat...
Bonus!

Branches

- When working on a moderate to large size program it is important to maintain a stable version of the code-base.
- Making updates to code can make it unstable, so it's not safe to work within the 'Trunk' (main branch) of the repository.
- Branches allow you to create a copy to work on separately without affecting the main branch.
Creating and Merging a Branch

- You can create a branch by using the 'copy' command
  - `svn copy trunk/ branch/ -m"look ma a new branch!"`
  - This copies both the files and the histories of those files
- To merge a branch
  - From the trunk:
    - `svn merge branch`
    - All conflicts created during the merge will have to be resolved
  - Commit
Some Thoughts on Branches

- Don't create unnecessary branches
  - Merging can be very tedious
- Give your branch a meaningful name
- Have a "target" for when you will merge the branch back into trunk
SVN Related Questions?
Documentation

- Why should you Care?
  - If you do a good job writing software, someone else is going to want to use it
  - They will need to be able to understand your code without reading the entire thing and deciphering your logic
    - Reading undocumented code is tedious enough that people will end up rewriting it (or worse)
  - You want your future coworkers (and bosses) to like you :)

Documentation

- In Java the standard is javadoc
  - You learned about this in CS-220?
- Python has something similar (but not quite as nice) called PyDoc
  - This is what is used to create the information that you receive from help() in the python translator
  - In the future, you may want to consider learning and using Sphinx, but that is beyond the scope of what we need for this project
- You are expected to submit fully documented code for this class
Docstrings

- According to the Python spec (PEP 257):
  - "A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the __doc__ special attribute of that object."
  - String literals can be a few different things, but for documentation use three double quotes
  - String literals can span multiple lines
  - """" This is what I mean """"
PyDoc and Docstrings

- PyDoc will take the docstrings from your code and format them into documentation
- It is capable of doing many things, including serving the documentation to a web page, the interpreter runs it every time you call help
  - That suffices for our purposes
  - For more info:
    - [http://docs.python.org/library/pydoc.html](http://docs.python.org/library/pydoc.html)
What Should Go In Documentation?

- Documentation provided with code should serve as a way to understand it without ever having to read the code itself
  - A good rule of thumb is "What would I need to know to use this?"
- For classes:
  - Provide a short class description.
- For methods:
  - Provide:
    - Method description
    - Input variables (what they are and what they are expected to be)
    - Output (what you are returning, in what format)
    - Exception Handling
Example:

class ExampleClass:
    """
    This is an example class to show documentation """

def doSomethingGreat(a, b):
    """
    This function takes two integers and raises and adds them together.
    
    Input: Two integers
    Output: An Integer that is the sum of the two input integers
    Exception: If non-integers are passed in a FormatException is thrown """

    if not isinstance(a, (int, long)) or not isinstance(b,(int,long)):
        raise Exception('You failed to pass in two ints!')
    else:
        return a + b
Other Tools

For your project you may also find the following tools useful

- For Javascript
  - Firebug or Chrome's built in Developer tools (especially Javascript console)
    - This will take away a lot of the mystery from Javascript errors
  - Aptana for either Eclipse or standalone
    - Nice IDE for Javascript
- For Python
  - PyDev for Eclipse
  - Stain's Python Editor (standalone Python IDE)
- For All Things Code
  - StackOverflow.com, easily the best Q&A site for software development, most things will already be answered!
Questions?