Subversion, Python, and other Occult Practices

Chris Kees

Computational Mechanics Brown Bag Seminar

U.S. Army Engineer Research and Development Center
Mississippi State University
North Carolina State University

4/2/2008

Outline

- Python scripting and object-oriented programming.
- Version control with subversion.

Pros and Cons of Python

- Python is powerful and fast.
  - Interpreted language with a byte-compiler
  - Large and evolving standard library using C and C++
  - Clear, concise, and readable syntax
  - Fully supports intuitive object-oriented programming, modularity, exception handling.
- Python plays well with others.
  - C/C++ API allows extension modules to be written in C/C++/FORTRAN (call C from python).
  - Easy to embed in other applications (call python from C).
- Python runs everywhere.
  - Large user community and C code base make it very portable.
- Python is friendly and easy to learn.
  - Integrated/free documentation, tutorials.
- Python is open.
  - Has it's own open source license to ensure that it's freely usable and distributable even for commercial use.

Main features of Python

- Interpreted
- Dynamically typed: a = 1.0; a = [0.0,1.0,2.0]
- Strongly typed

```
>>> a=1.0;b='hello'; c=a*b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int
of type 'float'
```

- All variables are references
- Has state-of-the-art data structures and algorithms
- Large, well-documented library of extension modules (e.g. hdf5,cgi,vtk,gui,...)
- Very portable AND has pre-built binaries on most platforms.

Strategy for Scientific Computing

- Reduce initial design/implementation time by
  - Eliminating the compile step
  - Interactive experimentation
  - Using high-level data structures/algorithms
  - Leveraging extension modules
  - Quickly/easily manipulating strings, files, command line options, threads, pipes, sockets...
- Roll over slow Python to C/C++/FOTRAN extensions
  - Use built-in profiling to identify code (usually loops).
  - Use standard C API to build extensions.
- Build stand alone libraries of C/C++/FORTRAN based on extension module requirements.

What is version control?

- From the web:
  - Version control systems keep track of multiple versions of a source file. They provide a more powerful alternative to keeping backup files.
  - Version control allows you to manage the life cycle of a document from conception to final copy, with the ability to roll back versions and track usage within all versions.
  - The process by which the contents of each revision of software, hardware,or documentation are accounted for.
  - The coordination and integration of the history of work submitted by a team.

Useful Terminology

- Revision: A numbered version of a file or directory that a version control system will manage.
- Repository: The place where revisions are stored and obtained. A repository is a file server that remembers the entire history of each file and directory stored on it.
- Working Copy: A file or directory that began its life as a copy of a revision.
- HEAD: The latest revision of a file or directory on the repository.
- BASE: The “pristine” revision on which a working copy is based.

Getting Started

- If the project is already under version control, you want to checkout the latest revision on the repository:
```
>svn checkout REP/projectA
```
- If the project is not yet in version control, you want to import it into the repository and check it back out:
```
>svn import projectB REP/projectB
>mv projectB projectB.backup
>svn checkout REP/projectB
```
- If you are starting from scratch, you want to start a new project by making a directory for it on the repository and checking that out:
```
>svn mkdir REP/projectC
>svn checkout REP/projectC
>emacs projectC/hello.C
>svn add hello.C
>svn commit
```

Keeping track of revisions

- When you want to make a backup copy (revision) of your working copy, you commit your working copy.
```
>cd projectA
>svn commit
```
- Problem: If you have working copies of projectA on machineA and machineB, then one of the working copies is now based on an old revision. You can no longer commit that working copy as a new revision (What revision number would you give it?).
- Solution: Update machineB:projectA with the differences between REP:projectA:HEAD and machineB:projectA:BASE.
```
machineB>cd projectA
machineB>svn update
machineB>svn commit
```

Basic Work Cycle

- Update your working copy
  - `>svn update`
- Make changes
  - Edit your working copy.
  - Make structural changes.
    - `>svn add`
    - `>svn delete`
    - `>svn copy`
    - `>svn move`
- Examine your changes
  - `>svn status`
  - `>svn diff`
  - `>svn revert`
- Update changes on the repository into your working copy
  - `>svn status -u`
  - `>svn update`
  - `>svn resolved`
- Commit your changes
  - `>svn commit`

Branches and Tags

- It’s often convenient to allow different versions of a project to coexist within the version control system. We uses branches for this purpose.
- With svn, a branch is simply a copy of the original project:
  `>svn copy REP/projectA REP/projectA-dev`
- It’s often convenient to give a more user-friendly name to a particular revision. We uses tags for this purpose.
- With svn a tagged version is again just a copy:
  `>svn copy REP/projectA REP/projectA-2.0.0`

Diff and Merge

- To view differences in revisions and working copies, use `svn diff`
  - `>svn diff REP/projectA REP/projectA-dev`
  - `>svn diff --revision 244 projectA`
- To merge differences in branches into a working copy use `svn merge`. For this example assume we created projectA-dev at revision 120.
  - `>svn checkout REP/projectA-dev`
  - `>svn merge REP/projectA-dev@120 \ REP/projectA projectA-dev`
  - `>svn commit`
- Important! Note in the log that you merged projectA and projectA-dev into this revision of projectA-dev.

Best Practices

- Explain what you changed in the log messages.
  - Trick: include the list of modified/added files that svn conveniently generates for you in the editor
- Commit changes often
- Commit changes before doing anything that affects your working copy.
- For new projects with multiple developers, use the “trunk, branches, tags” layout.
- Log information about a merge.
- Create a branch if 1) many people depend on the trunk and 2) you intend to make significant changes.
  - Merge the changes in the trunk to your branch as often as you can stand it.
- Make a tagged version whenever you get a particularly interesting result.

Working with ADH

- Make your own branch:
  ```
  >svn copy REP/adh/trunk \
  REP/adh/branches/xyz-dev
  >svn checkout REP/adh/branches/xyz-dev \ adh
  ```
- Commit your changes often; run update when you log in to a machine.
- Merge changes from the trunk (or other branches) as often as you like:
  - View log to find out the last time you merged:
    ```
    >svn log
    ```
  - Merge:
    ```
    >svn merge \ REP/adh/branches/xyz-def@LAST-MERGE
    \ REP/adh/trunk@HEAD .
    ```
  - Commit and log details of merge!:
    ```
    >svn commit
    ```