Subversion

- CVS “done right”
- If you know CVS, you know SVN.
- Some new operations are possible:
  - Copy
  - Move
  - Remove entire directories
  - Externals: referring to other parts of the repository
- Some old operations are much easier
  - Branching and Merging
  - Switching branches in-place

$ svn help [command]
External and src trees

• The repository is located on ginger, in our lab.
• We have set up access to the repository using ssh keys.
• The external tree contains packages not easily available through the packaging system, or for which we need a particular version.
  – Built using a minimal Makefile and a perl script to determine what needs to be built using which compiler.
  – Each package's own build system is invoked.
  – Build everything with a simple 'make' command
• The src tree contains our own software
  – Built using a python tool called waf
External tree

- May be shared between multiple src trees
- Should be checked out and built as a common user/group shared between all users

```bash
$ su demo
$ svn checkout
svn+ssh://ace@ginger.autonomy.ri.cmu.edu/trunk/external/home/shared/external
```
• **Src tree**

  • Compiling and installation will create sibling directories
  • Looks for external tree (or a symlink to it) as a sibling

```bash
$ cd ~
$ svn co svn+ssh://ace@ginger.autonomy.ri.cmu.edu/trunk/src src
$ ln -s /home/shared/external external
```
Waf

- Build system with a workflow similar to GNU autotools

  ```bash
  $ ./waf configure
  $ ./waf [build]
  $ ./waf install
  $ ./waf clean
  $ ./waf distclean
  ```

- The waf tool is written in python, and contained in the repository.

- `wscript_build` files take the place of Makefiles in each directory
  - These files declare objects to be built, such as staticlib or program.
Where files are placed

• Two compiler versions are used in the tree
• Waf builds files in a mirror of the source tree under obj/default (for gcc3.3) or obj/gcc4
  – Generated xdr and tdl sources
  – Object files
  – Libraries
  – Binaries
• Waf installs files (by default) to ../bin, ../lib, etc.
• We often skip the build step and rely on the install step to build files so they are easy to find
Limiting what is built

- To create only generated source files (XDR and TDL), start with:
  
  ```bash
  $ ./waf [build|install] --only-generate
  ```

- Running waf from a subdirectory in the build tree will only build that subtree.

- You may also specify particular targets to be built or installed.
  
  ```bash
  $ ./waf --targets=targetA,targetB
  ```

- With the exception of generated source files, waf handles dependencies in different directories.
Waf wscript_build files

- See src/ReferenceMaterial/README.waf for details

```python
# BUILD A LIBRARY

obj = bld.create_obj('cpp', 'staticlib')
obj.target = 'spacemouse'
obj.source = 'spacemouse.cc'
obj.includes = '.
obj.uselib = 'XDRV'

# BUILD BINARY

obj = bld.create_obj('cpp', 'program')
obj.target = 'spacemouse_test'
obj.source = 'spacemouse_test.cc'
obj.includes = '.
obj.uselib = 'X11 IPC PTHREAD'
obj.uselib_local = 'spacemouse common'
```
Deploying

• It is convenient to build software on multiple machines or in different directories
• Microraptor looks for programs in a particular directory
• A standalone shell script, deploy.sh, copies files (over network, if necessary) to the proper location.
  – Programs
  – Shared libraries
  – Configuration files
  – Data files
• deploy.sh looks for files in your installation directories