CMSC 22620 Spring 2011	Implementation of Computer Languages 2	Handout 3 April 1, 2011

Phoenixforge and Subversion

Phoenixforge

We are using the CS department **phoenixforge** server to manage project submissions. The server hostname is `https://phoenixforge.cs.uchicago.edu`. You can access it using your web browser at that address.

Before we can set up youy can have a project, you need to register yourself as a user. Do this by pointing your web browser to `https://phoenixforge.cs.uchicago.edu` and log in using your CNetID (CNet user id) and password. This will register you as a phoenixforge user, unless you have logged in previously, in which case you will already have a phoenixforge account.

After you have done this, email your CNetID to the TA at `sepopley@cs.uchicago.edu`. As soon as we can, we'll set up your first project and email the class list when they are available. Your personal phoenixforge project will be called *`user`*`-icl2`, where *user* is your CNetID. Future projects will follow the same naming scheme.

Using Subversion

Once a project is created for you, it will have a Subversion repository on the server. You are expected to keep the source code of your projects in the repository. To *checkout* a copy of a project called "`project1`," run the following command:

```
svn checkout https://phoenixforge.cs.uchicago.edu/user-icl2/svn/project1
```

where *user* is replaced by your CNetID. On your first checkout, you should be prompted for your password. It will assume you are using the username of the account executing svn. If your CNetID is different from your login name, just press enter at the password prompt and it will then ask you for your phoenixforge username (i.e. your CNetID) and then your corresponding password. If everything checks out, a directory called `project1` will be created in the current directory. All the files related to your project should live in this directory.

Now suppose you create a file called `foo.sml` in your `project1` directory. In order for Subversion to keep track of it, it needs to be added to the repository. You do this using the following command:

```
svn add foo.sml
```

You should see a message like:

```
A         foo.sml
```

This command records the fact that `foo.sml` has been scheduled to be added to your repository, but the file will actually be added only when you commit your changes. To do so, type the following command:

```
svn commit -m "added foo.sml"
```

to add the file permanently to the repository. If you omit the `-m` option, you will be prompted to enter a log message in your default editor. After you have entered the commit command, you will see a message like the following:

```
Adding         main.sml
Transmitting file data .
Committed revision 1.
```

Changes you make to your files are recorded in the repository every time you do a `svn commit`. Before you make changes to your files, you can ensure that you have a current version, by running `svn update`. This fact is not of tremendous significance for individual projects, becomes important when more that one person can modify the same files.

Not all the files in your project directory need to be in the repository. For example, you should not put your executable files in the repository — these can always be recreated (hopefully!) by compiling the source. In particular, you do not need to add the files in the `.cm` subdirectory that are created by the compiler when you run `CM.make`.

The `svn diff` command is used to comparing the version of a file (or files) in your working directory with the version(s) in the repository. If no files (or options) are specified, all working files are compared to their last committed versions, otherwise only the specified files are compared. There are also flags to compare other versions, see the man pages or the online manual for details.

Other useful svn commands are:

svn status This lists files in the current directory and subdirectories that have been modified (M) but not committed, or have not been added to the repository (?).

svn delete *filenames* This deletes the designated files.

svn move *path1 path2* This moves the file designated by *path1* to the location/name specified by *path2*.

The Subversion home page is at `http://subversion.apache.org/`. Full official documentation is available at `http://svnbook.red-bean.com/`.

Coding standards

It is important that your code be easy to read and understand. To help you achieve that goal, here are some suggestions for coding conventions.

- Use the module system to structure your program and have a separate file per top-level module. Signatures can either go into the file of the module they belong to or in their own file. If a signature is referenced in multiple files, then it should be in its own file (*e.g.*, if it has multiple implementations).
- Give your modules signatures.

- Document your code: every module and every top-level definition should have a comment explaining its purpose and invariants. Non-obvious implementation techniques should also be explained in comments.

- SML has a number of different classes of identifiers and there are common conventions for how they are named.

  Signature names Signature names are all caps with underscores to separate words (*e.g.*, `FINITE_SET`).

  Module names Structure and functor names are mixed case with a leading upper-case letter (*e.g.*, `FiniteSet`).

  Type names Type names are lower case with underscores to separate words (*e.g.*, `finite_set`).

  Type variables Type variables are lower-case (*e.g.* `'a`).

  Exception and data constructor names Data constructors (including exceptions) follow the rules for structures.

  Variable names Variables are mixed-case with an initial lower-case letter (*e.g.*,`numElements`).

  Field names Field names follow the rules for variable names.

- Do *not* use **`open`**; instead define a shorthand local alias for a module. For example:

```
structure S = FiniteSet
  ... S.numElements s ...
```