CMSC 22610 Winter 2007	Implementation of Computer Languages	Handout 3 January 9, 2007

Gforge and subversion

Gforge

We are using a **gforge** server to manage project submissions. A server has been set up with hostname `cs22610.cs.uchicago.edu`. You can access it using your web browser at that address.

Before you can have a project, you need to register yourself as a user. Do this by pointing your web browser to `http://cs22610.cs.uchicago.edu` and clicking on the link "New Account" on the top right corner of the page. Follow the directions (you only need to fill in the starred fields) and submit the form. In a few seconds you should receive an email with a link to confirm your registration. Click on it and log in and you should have an active account. When this is complete, email the TA at `adamshaw@cs.uchicago.edu` with the account name you just set up. As soon as we can, we'll set up your first project and email the class list when they are available. Your first project will be called `project1`*`user`*, where *`user`* is your gforge username. Future projects will follow the same naming scheme. In order to be declared a member of the project, you must log in and go to the project's web page:

`http://cs22610.cs.uchicago.edu/projects/project1`*`user`*`/`

There is a link that says 'Request to join'. Follow it and click the submit button for the form on the linked page. You should receive email once you are approved for the project and you will then be able to use the Subversion repository.

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 "`project1user`," run the following command:

```
svn checkout svn://cs22610.cs.uchicago.edu/project1user project1
```

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 gforge username is different than this name, just press enter on the password prompt and it will then ask you for your username first and then your 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 `main.c` in your `project1`*`user`* 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 main.sml
```

You should see a message like:

```
A         main.sml
```

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

```
svn commit
```

to add the file permanently to the repository. You will be prompted to enter a log message in an editor. You can avoid editors altogether by typing your log message on the command line with the `-m` flag:

```
svn commit -m "added files"
```

After you have entered your message, 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, but matters 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.

The "`svn diff`" command is for comparing differences between versions. 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.

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 local alias for a module. For example:

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

Useful resources

The Subversion home page is at `http://subversion.tigris.org/`. Official documentation is at `http://svnbook.red-bean.com/`.