Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik

Prof. Dr. W.-P. de Roever

Marcel Kyas, Gunnar Schaefer Martin Steffen

P-I-T-M

WS 2004/05

Handout 2.(1)

9. November 2005

Handout2.(1): Platform and software

Ausgabetermin: 9. November 2005

The purpose of this handout is to give infomration about the available platforms, access methods, the currently available software.

1 Hardware + access

We have one (not-too fast, not-too-big, not-too-new ...) PC hooked up to the net, over which we have control. Even it the machine is not a high-performance server, the tool should run

Here is some data descibing the machine.

name	`snert.informatik.uni-kiel.de`
IP	134.245.253.11
disc	ca. 3G available (/home)
Mhz	800Mhz
main memory	128M
swap	ca 256M
OS	Linux 2.6.8 / Fedora core 2 (+ updates)

The purpose of the machine is 3fold[1]

1. subversion server
2. test platform
3. bugzilla server

More details concerning in particular subversion that below.

[1]One particular purpose for which we cannot use the machine directly is for *backup,* since it is not part of the network's backup system. As solution, we are running a cronjob which in regular intervals copies relevant data to the sun-pool.

2 Subversion

From the discussion, we learnt that we will use *subversion*, which is mandatory for all participants. Currently, we have on the solaris pool the *server side* of subversion running.[2]

The command is

```
/opt/local/bin/snv
```

which (probably) means, it's in your execution path. Later we will discuss further rules of the game concerning access to the server, but here are a few easy commands to start with. For deeper knowledge, refer to the subversion manuals. Their are linked to our course pages.

2.1 Technical prerequisites

To access the repos, you need the client-side part of *subversion*. For the solaris pool this is installed (see below). Subversion is also shipped in modern Linux distributions. As far as installation on Windows or Mac platforms is concerned, there's no experience from our side, but it seems that some of the participants have expertise there. If proplems should arise there, please share the knowledge.

Furthermore, for write access to the repos, you need some *password*. We will send more instructions about this later.

2.2 First access and survival guide

Luckily, many commands look very similar to the ones you might be used from cvs. The most important ones are probably:

checking out , i.e., getting the first copy of the repository into ones own workspace

update getting one's own copy into sync with the general development

commit writing back own changes

2.2.1 Checkout

For a read access, checking out the currently rather empty repository, you don't yet need a password. Create (or choose)in your workspace some appropriate directory, say `WORKDIR`, and then do

```
cd <WORKDIR>
svn co http://snert.informatik.uni-kiel.de:8080/svn/coma
```

As an explanation: `co` stands for "checkout" (also `checkout` instead would do), next comes the url of the *repository* which is accessed via http and the apache server running on snert. The part `/svn/coma` gives the *project* `coma` on the repository which is specified by `/snv/`.[3]

[2]The reason why we can't us a solaris machine also for the server side, which would simplify matter a bit is technical: currently subversion cannot deal with NFS'ed file systems very well, and the sys-admin are currently not (yet) setting up a dedicated svn-server machine.

[3]The `/svn/` is one full repository as seen through apache. At the server side, the actual directory containing the repository is kept at a space in the file system accessible by apache.

2.3 Testing

If new to subversion or and version control, it might be help to play around a bit. In order to do this without fear of destroying real work, we set up a *second repository* for that purpose. It is completely separated from the "real" one, so it's not just another project on `/svn/` but is called `/snvtest/` instead.

So you might try:

```
svn co http://snert.informatik.uni-kiel.de:8080/svntest/coma comatest
```

Example

The example uses `$WORKDIR` as name for the *working directory* you choose appropriately. In principle you can have as many working directories at the same time, i.e., more than one checked out copy of your project. A typical scenario would be to have one at the university pool and one at the workplace at home. With the number of checked out working directories, however, the danger of confusion increases; in normal situation, one woring directory per "workplace" suffices.

check out: as explained already above, the following gives you the first copy of the *Coma*-project:

```
mkdir $WORKDIR
cd $WORKDIR
svn co http://snert.informatik.uni-kiel.de:8080/svn/coma
```

As said: Instead of `svn`, you might wish to play around with `svntest` for a start. The "variable" `$WORKDIR` is here just for illustration, pick yourself an appropriate place. You can also pick only parts of the repos, for instance

```
svn co http://snert.informatik.uni-kiel.de:8080/svn/coma/org
```

gives you info about the groups etc.

write back: Once done with your changes, you can hand back the whole work tree, doing

```
cd $WORKDIR/coma
svn commit
```

Afterwards you will be asked to give a comment about what changes you did in some editor. If you do not have much to say for that revision, you can write back faster by typing

```
cd $WORKDIR/coma
svn commit -m"Small bug fix"
```

update: to get the tree in your workspace in sync again, i.e., to obtain possible new changes, is done by:

```
cd $WORKDIR/coma
svn update
```

Typically, this should be done when you start working ...

new files and directories can be added by

```
svn add [filename]
```

They can be removed again by

```
svn remove [filename],
```

Instead of a filename, also a directory can be added. Furthermore, subversion honors regular expression, so it understands things like `svn add *.java`.

moving & copying: Here's something new for cvs-users: you can move really move directories (and files). The fact, however, that this is possible does not mean, that it is useful to keep the project under constant re-construction. Cf. the remarks about policy below.

Strategy and policy

Version control does not guarantee smoothless work. I supports it, however, if used with with one's brain switched on and with discipline. From out side, we have not yet had experience with subversion (with CVS however), therefor more specific do's and don't may follow. The following rules of thumb might give first guidance.

- In general: if there is danger that changes affect negatively other group's work, clarify this before you check in your changes. In particular
  - directly changing other persons's work ("I can fix that better/faster than those ..."): *only* after *careful reflection* and communication with the concerned group
  - No unannounced change in the directory structure. New subdirectories in one's own part are ok, though. No fiddling around with administrative subversion properties.
  - No global *undo* ("quick fix repair") of other person's changes without communication.
- Later in the course: check in only versins that *integrate* with the rest of the code (for instance, the can compile together). We will work out a strategy for that, probably using *make*. After intergration interface changes (whether the interfaces are formal like Java interfaces or informal such as common agreements) *must be announced* and accepted by concerned parties. (for instance during the meetings or the email list).
- `Makefile`s and `Readme`s are useful.

3 Further software

There is a number of other software installed on the machine. we give here a short overview, a later handout will contain an update, when new stuff arrived, but also more specific infos concerning the use and the access.

3.1 Bugzilla

Bugzilla is some server which helps to maintain a common database of bugs during the software devolopment. In particular, it helps to keep track of the status of an error, for instance

- who reported the error?
- what kind of error, short description?
- who is (probably) responsible?
- was it acknowledged?
- has it been repaired?

As with subversion, it can be be a helpful tool when used with care, but obviously does not guarantee solid development. For instance, reporting bugs and having a nice overviews over the various developments does not help in the end, if noone feels responsible for fixing them . . . In previous courses, we used a "manual" kind of bug-tracking system and that was pretty much the situation.

With bugzilla such much more powerful and easer to use than some textual (but formatted) *error-lists*, different dangers might occur, for instance that we are swamped in error logs and we loose the focus. We might hunt down one tiny bug after the the other, the statistics looks nice, but we loose in ther overall goal.

A further danger could be social. People don't like if someone else spot errors in one's own work (and make them public, as well). A reported error as such is first of all something useful, but don't take it as "*sport*" to notch as many (big, small, miniscule) errors of others on your knife. . . . And refrain from error reports as *retaliation!* ("Now if they find 5 bugs in my part, let's sit down and see what kind of shitty code they have, I'll give them back 50, that should teach them". yes, things happen)

Since we currently don't know, how people will adapt to the situation (till next week, we might hand out further hints concerning rules of engagement) we will have to monitor whether bugzilla is used to support the aims of the project. Reacall: bugzilla is a *tool* not a *goal* of the course.

php	4.3.8
python	2.3.3
mySQL	3.23.58
apache httpd	
"tomcat"	partly installed
"java"	1.4

Table 1: further software @ snert

References

[CSFP04] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion (for subversion 1.1)*, 2004.