

## PM – Workshop 1

Development environment setup

...and how to wield the SVN sword

## ***Introduction***

This workshop introduces a typical development environment by letting you install and extend a toy application that calculates the number of days to next vacation. The application is divided into different archives and typical SVN work flows are experienced while integrating them. I give thanks to Joachim von Hacht for providing the overall structure of the application.

You are required to use some parts of the environment presented here during this course (hosting your project at Google Code with SVN as source code management software). Others are optional, like using Eclipse as IDE. However, we require you to use what's presented here during this workshop. Now, let's enter the free world!

## ***Project site creation***

### ***– Becoming friends with Google***

First, go to <http://code.google.com/hosting/> and click [Create a new project](#).

Fill out the form, but observe:

- Project name (of your real project!): From course PM: "The name should start exactly with 'tda366-' followed by a chosen suffix for example tda366-duckrace or tda366-drawpad". YOU CAN'T CHANGE THE NAME LATER.
- Version control system: YOU MUST CHOOSE Subversion.
- Source code license: This is up to you, but still an important choice! All of the licenses are open source, i.e. your code will be available for anyone to download. Two typical categories of open source licenses are "copyleft" and "permissive"<sup>1</sup>. In short words:
  - A "copyleft license" requires the author of a derivative work (someone extending your code) to provide access to that code, i.e. everything derived from your work will continue to be open source. Perhaps the most well-known license in this category is the "GNU Public License".
  - A "permissive license" is just that, permissive. It doesn't put many demands on someone using your code, for example, it's free for people to modify and redistribute your code as part of a commercial application. So what does a "permissive license" contain? It usually ensures that you won't be held liable for any damage your software might cause a user, and sometimes also claims that an attribution to you shall be included in a derivative work. The "New BSD License" or "MIT License"<sup>2</sup> are typical choices.
  - You can find more information at <http://opensource.org/licenses/category>.
- Use a separate content license: If you want to use separate licenses for your code and the other parts of your application, e.g. the documentation. You can choose the other license here.
- Project labels: Keywords to describe your application.

When enabled, click "Create project", voilà! You're in the game!

You will now be transferred to your project's main page, where the tabs mark different sections and functions on the site. Click "Source" and turn this page!

---

<sup>1</sup> The distinction is blurred for some licenses.

<sup>2</sup> The "MIT license" is sometimes called "X11 license".

## **Initial content setup**

### **– There's a world above you**

Google is running a Subversion (SVN) server that monitors a repository, that is a storage containing your project files (not necessarily all of the files you'll be dealing with, but the ones you'll find convenient to share among everyone in the group). SVN is a system for sharing files and managing contributions from different users, for example when two users edit the same file. It is also a versioning system, it keeps track of all committed changes of all stored files and makes it possible to go back in history to retrieve or restore older versions. The collaboration means that every member in the project have their own copy of the group of files, a *working copy*. Each copy is synchronized with a central copy, the *repository*.

The process of creating a working copy is called *checking out*. On the "Source" tab of your project site you'll find the following text:

```
svn checkout https://<project name>.googlecode.com/svn/trunk/ <project name> --username <your username>
```

This is how to do the check out using the the command line svn client (the *svn* command). In most occasions you'll probably choose a graphical svn client, preferably embedded in your IDE (Integrated Development Environment). On the Chalmers computers, the Eclipse IDE has an SVN client embedded as a plugin, it's called Subclipse. Let's create a working copy from our newly created repository. The information needed can be read from the command listed above.

Let one person in your group will initially both add content to the repository and check out a working copy:

1. Download the project archive holiday\_check-start.zip and import in Eclipse.
2. Right-click on the project folder and choose Team → Share Project...
3. Choose "SVN" and click Next
4. Choose "Create new repository location" if asked for it.
5. Copy and paste the information from the command above: `https://<project name>.googlecode.com/svn/trunk/`
6. Choose "Use project name as folder name" and click Finish.
7. Possibly choose to accept the untrusted site.
8. Enter username and password (the username is listed in the command; the password is not your Google account password, you can reach it from a link on the same page as the command).
9. In the opened "Synchronize" perspective, click the button Commit All Outgoing Changes... (the name will pop-up).
10. All commits (changes to the repository) shall be documented with a note. In the window that pops up, add a short message like "Initial commit". Then start the transfer by clicking OK.

The rest of the members can now check out their own working copy by:

1. Start Eclipse
2. File → New → Project...
3. SVN → Checkout Projects from SVN
4. Choose "Create new repository location"
5. Copy and paste the information from the command above: `https://<project name>.googlecode.com/svn/trunk/`
6. Enter username and password (the username is listed in the command; the password is not your Google account password, you can reach it from a link on the same page as the command).
7. Click to mark the base of the tree (probably at the top, ending with trunk) and click Next.
8. Mark "Check out as a project in the workspace", write a suitable name e.g. your project name.
9. Click Finish.

You've now created a working copy of the content in the repository.

## ***Changing file content***

### ***– When life is good!***

Just changing file content is straight forward, you can use any tool you wish as usual (probably Eclipse now). But you're only changing the content in your working copy! When you want to upload these changes to the repository, you'll do a commit. To bring up the window in the toy application, let someone in your group:

1. Uncomment the corresponding lines in file “Main.java” and save the file.
2. Right-click the project folder and choose Team → Commit...
3. Enter a note and click OK.

The rest of the group, does an update to retrieve the changes:

1. Right-click the project folder
2. Choose Team → Update to HEAD

Check to see that the update is performed, and test the application.

This is the typical way to work: Everyone does changes in their respective working copy, uploads them with “commit” and receives the other's changes with “update”. Try the simple application!

## ***Communication between people***

### ***– Letting Google be the man in the middle***

Even though a source management system like SVN can notify about conflicts and alleviate file sharing, it's still only prevention of total clashes. It doesn't help your group work effectively dividing tasks, organize, manage the work flow etc. This you'll have to discuss and set up! There are tools to use though...

#### **Wiki**

Your project site has a wiki section that lets you document your project and tag different pages with labels.

Read: [http://code.google.com/p/support/wiki/GettingStarted?tm=6#Documenting\\_your\\_Project](http://code.google.com/p/support/wiki/GettingStarted?tm=6#Documenting_your_Project)

#### **Issues**

To make a list of requested changes, the issue section of the project site can be used. This is typically used to report bugs, ask for reviews of changes, and sometimes also for requesting new features. An issue is usually tagged with a severity to announce how important it is considered. It is also changing status over time as work on it progresses, from being issued ("Open") to being fixed ("Closed"). Several predefined options are provided to choose from.

Read: [http://code.google.com/p/support/wiki/GettingStarted?tm=6#Tracking\\_Project\\_Tasks](http://code.google.com/p/support/wiki/GettingStarted?tm=6#Tracking_Project_Tasks)

Observe that while editing an issue, you have to click in the text area below "Add a comment and make changes" to make more (important) options appear.

## ***Changing the repository tree***

### ***– Be special, use SVN!***

Now the ground is set to change the content of the repository. But remember! When you add, remove, move or copy files or folders, you have to use the svn client, not your usual OS commands, i.e. don't use a file manager<sup>3</sup> or commands like mv, cp etc. on the content of your working copy. If you do, SVN won't track those changes and be able to synchronize them with the repository. Eclipse (Subclipse) is an obedient disciple though, it automatically handles this for you when you drag and drop files in the Package Explorer view. But even though the changes happen directly in the working copy, they won't be reflected in the repository until you do a commit.

The toy application calculates when the next holiday occurs. But wouldn't it be nice if you could set the current date?

Let someone in the group do the following:

1. holiday\_check-changes.zip contains updates of two files, extract them.
2. Delete the current file MainFrame.java and add the new Mainframe.java
3. Replace all of the content in HolidayController.java with the new file's content by copy and paste.
4. Add the jar-file to the external library path.
5. Commit the changes and observe how the different changes are presented in the lower part of the commit window.
6. Add a new issue at the project site for a Code review of the changes.

Someone else shall now change the issue's status to “Accepted” and start reviewing (after updating!). Test the new features and close the Review issue.

---

<sup>3</sup> E.g. Nautilus, Dolphin or Windows explorer



## **Conflicts**

### ***– Can't live with them, won't live without them***

Even though the application calculates Swedish holidays, the application is in English. But the message box popping up shows a message in Swedish (complete Swedish message if it is run on a machine with a default Swedish locale). Let two people in the group simultaneously change this by editing method `nextHoliday()` in `HolidayController.java`. One of you exchange the second to last line with its corresponding commented version, while the other exchange the last line.

Now, the last one of you won't be able to commit your change, a conflict has arisen! You solve a conflict by first updating your working copy. When you have changed your working copy to a consistent state, you can commit your changes. There are some options when solving a conflict on a file: you can either fully discard the changes in the repository, discard all changes you made, or merge the changes together. The merge can either be done manually or (if possible) by letting SVN do it for you. In this case you can just run an update to HEAD and then do the commit (Eclipse will force the update). For more information about resolving conflicts, see chapter 2 in "Version Control with Subversion" ([link below](#)).

## ***Getting information***

### ***– The difference between here, there, now and then***

You can see differences over time by going to the History view in Eclipse:

Right-click on an item and choose Team → Show history. There you can also generate comparisons between versions in different revisions.

To browse and examine the content of the repository you can open the SVN Repository Exploring perspective:

Window → Open perspective → Other... → SVN Repository Exploring.

## ***Command line client***

This workshop has only been dealing with the Subclipse SVN client. The command line client is run by simply typing “svn”. A useful command is:

```
svn help
```

which gives you a listing of available commands, many of them have abbreviations (e.g. checkout = co). The commands are run with:

```
svn <command> [options]
```

Help on a specific command is received by:

```
svn help <command>
```

It's important to know that many commands are dependent on which the current directory is, the command might only affect that part of the tree. Observe also which commands only affect your working copy and which affect the repository. Some of them can affect either depending on what argument you're giving.

We won't deal with the commands, but you're welcome to try them out and ask for help! Here's a non-exhaustive list, many of the names shall now be familiar:

```
svn checkout
```

```
svn update
```

```
svn commit -m “Descriptive message”
```

```
svn add
```

```
svn mv
```

```
svn cp
```

```
svn mkdir
```

```
svn status
```

```
svn diff
```

```
svn list
```

```
svn cat
```

```
svn import
```

```
svn export
```

## Appendix

### Notes

- When you're checking out, it's one folder (and it's corresponding subfolders) that you get, i.e. not necessarily everything in the repository. It is possible to have several different so called "branches" of your application in the repository (in different folders). This is useful if you want to have several variants of your application. The folder "trunk" then typically contains the default branch of your application. You don't have to worry about this unless you choose to divide the project into different variants.
- Never commit code containing bugs or something that you know might trouble your fellow developers!
- Remember that you don't have to commit everything you've changed all the time. But try to update as often as possible! If you don't want to commit all changes, you can right-click on part of a folder-tree, or only a specific file.
- Again: Spend time to divide and coordinate the work between yourselves! It pays off in the end... and the fastest way to solve conflicts is not through SVN, it's by working in the same room so you can talk.
- Binary files shall be locked while edited, since they can't be merged. But you might not need to add them to the repository...
- It's possible to postpone the resolution of conflicts... see "Version Control with Subversion" (link below).
- I recommend reading chapter 1 and 2 in "Version Control with Subversion" (link below).

### Links

Google Code project hosting

Main homepage: <http://code.google.com/hosting/>

Support wiki: <http://code.google.com/p/support/wiki/GettingStarted>

Subversion

Main homepage: <http://subversion.apache.org/>

Free (extensive) O'Reilly book "Version Control with Subversion": <http://svnbook.red-bean.com/>

Subclipse Eclipse SVN client plugin: <http://subclipse.tigris.org/>