Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung

Subversion User Training

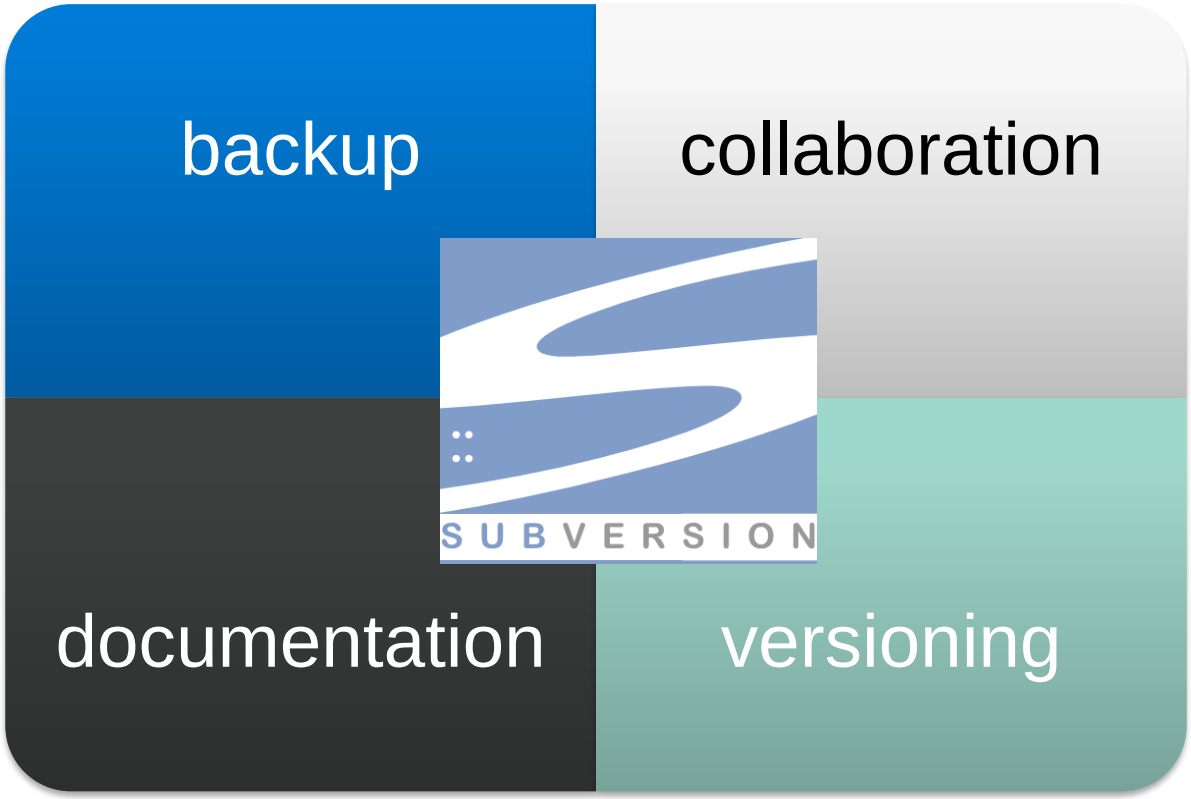
SUBTRAIN by POLARION

Agenda

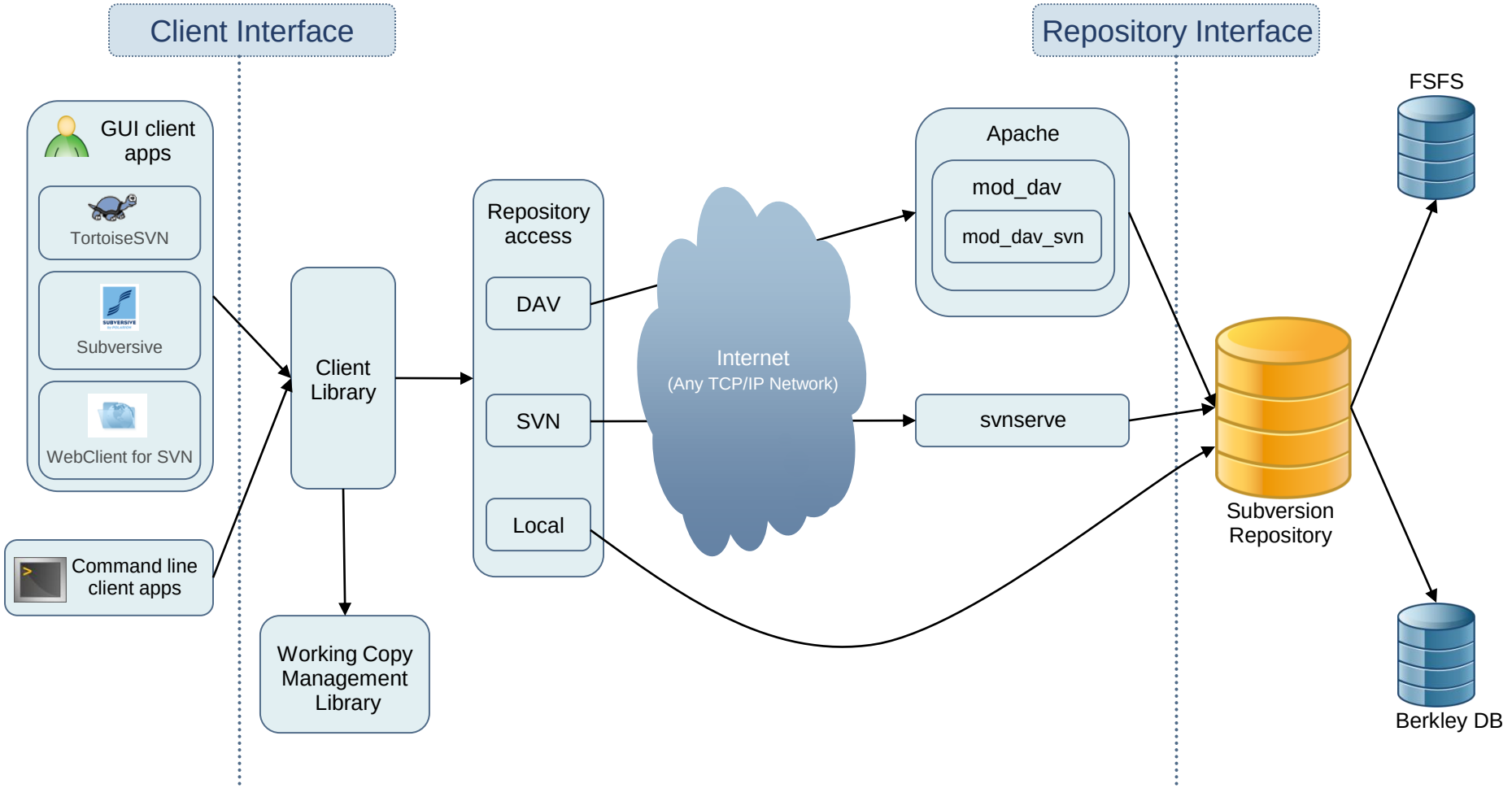
- Introduction
- Architecture
- Working cycle
- Locking
- SVN Properties
- Managing versions
- Branching strategies
- Merging
- Configuration
- NTSVN

Introduction

What is Subversion?



Architecture


Client Interface
Repository Interface
GUI client apps
TortoiseSVN
Subversive
WebClient for SVN
Command line client apps
Client Library
Working Copy Management Library
Repository access
DAV
SVN
Local
Internet
(Any TCP/IP Network)
Apache
mod_dav
mod_dav_svn
svnserve
Subversion Repository
FSFS
Berkley DB

- `file://`

  Direct repository access to local or network drive.

- `http://`

  Access via WebDAV protocol to Subversion-aware Apache server.

- `https://`

  Same as http://, but with SSL encryption.

- `svn://`

  Unauthenticated TCP/IP access via custom protocol to an svnserve server.

- `svn+ssh://`

  Authenticated, encrypted TCP/IP access via custom protocol to an svn server.

- **`file://hostname/path/to/repos`**

  On local machines the **hostname** part must either be absent or **localhost**.

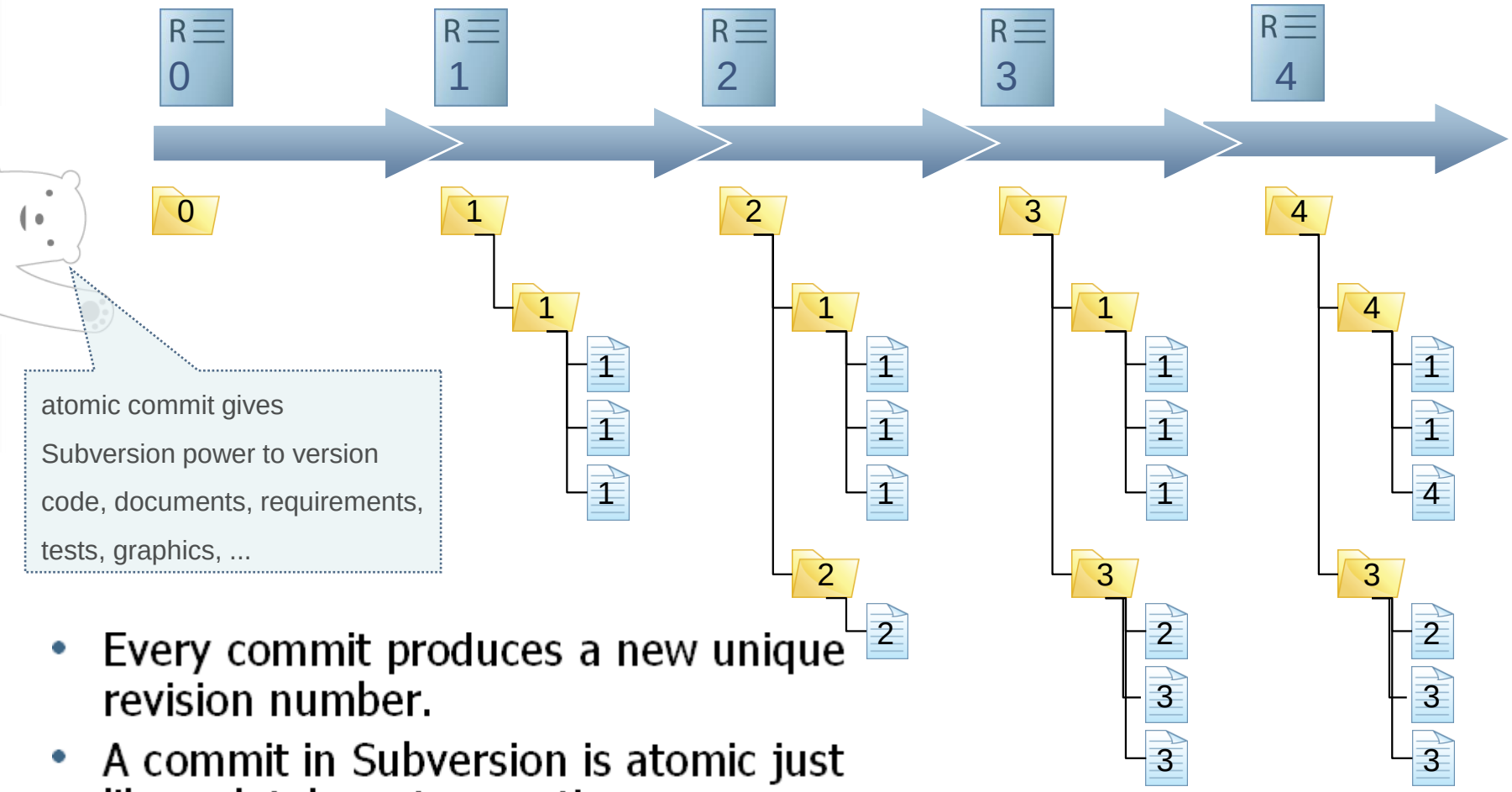
- This results in a path like this one:

  `file:///path/to/repos`

- On Windows you have to specify the drive where to find the repository:

  `file:///X:/path/to/repos`

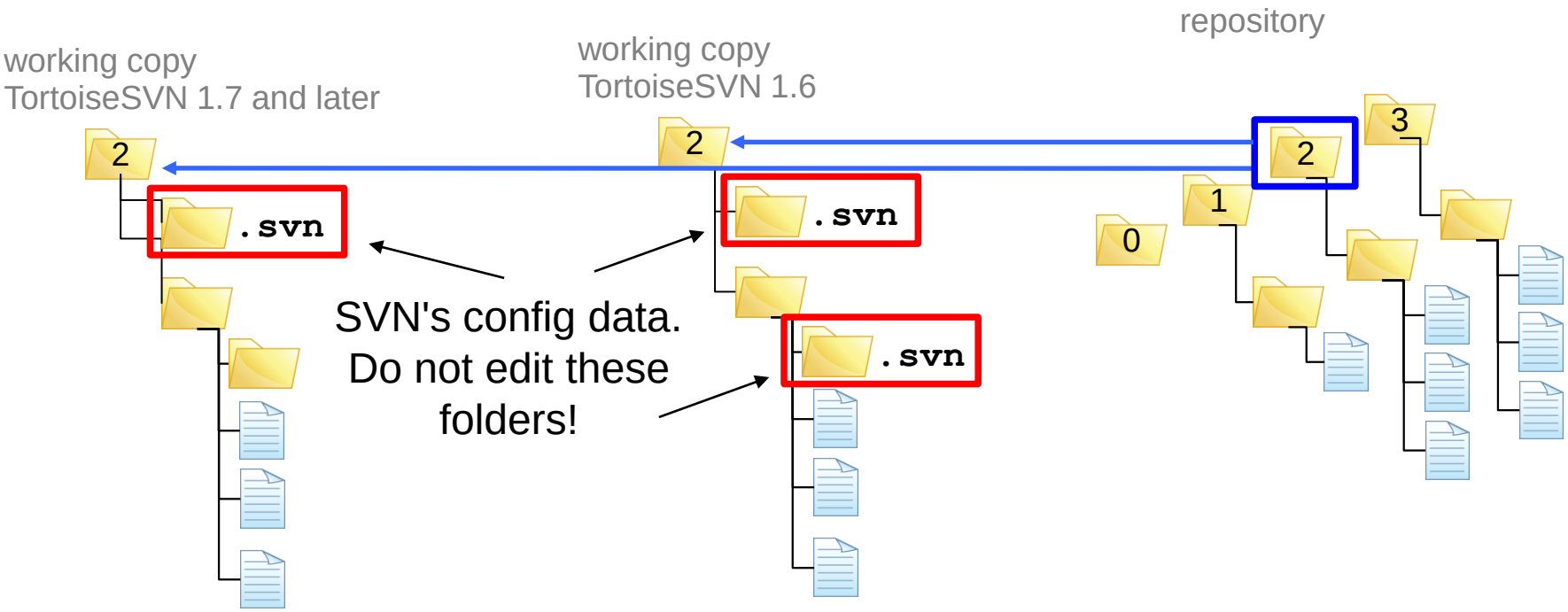
Architecture – The Revision numbers

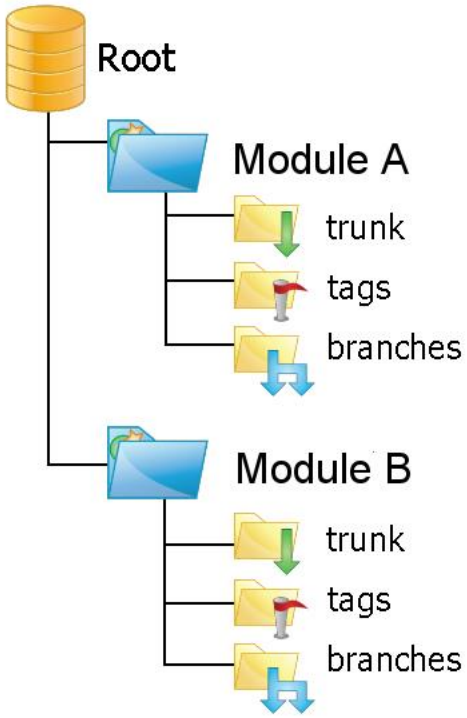


atomic commit gives Subversion power to version code, documents, requirements, tests, graphics, ...

- Every commit produces a new unique revision number.
- A commit in Subversion is atomic just like a database transaction.

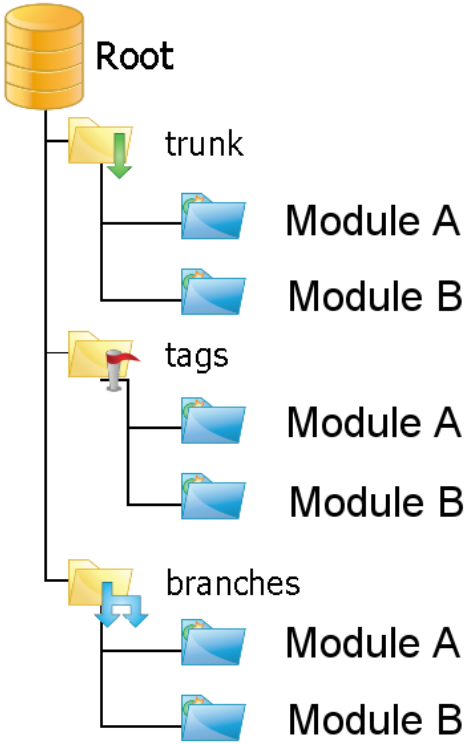
"Checking out" creates a working copy
of a specific revision of the repository





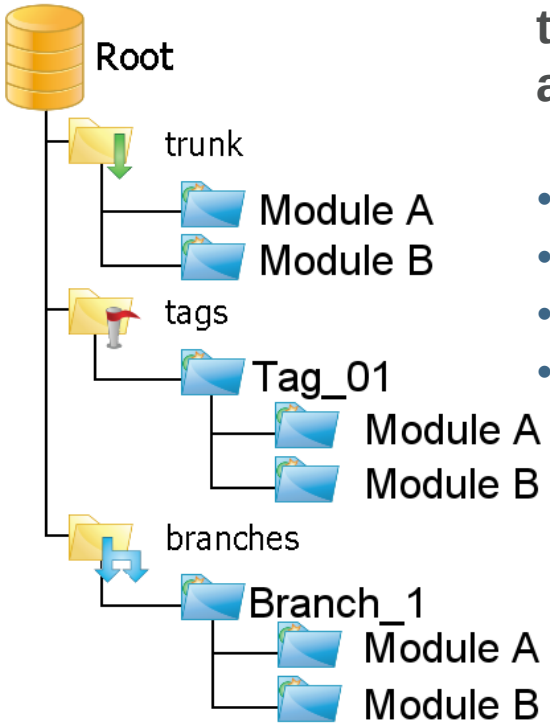
modules on root

- each module has to be checked out separately
- commits can not span multiple modules
- tags can not span multiple modules in single commit
  (workaround by 3rd party tools)
- „official“ svn layout



trunk/tags/branches on root;
each module will be tagged separately

- checkout of all modules is possible
- commits can span multiple modules
- tags can not span multiple modules in single commit
  (workaround by 3rd party tools)



trunk/tags/branches on root
all modules will be tagged

- checkout of all modules is possible
- commits can span multiple modules
- tags can span multiple modules in single commit
- maybe difficult to determine relevant modules of a tag

NewTec Repository Layout

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung

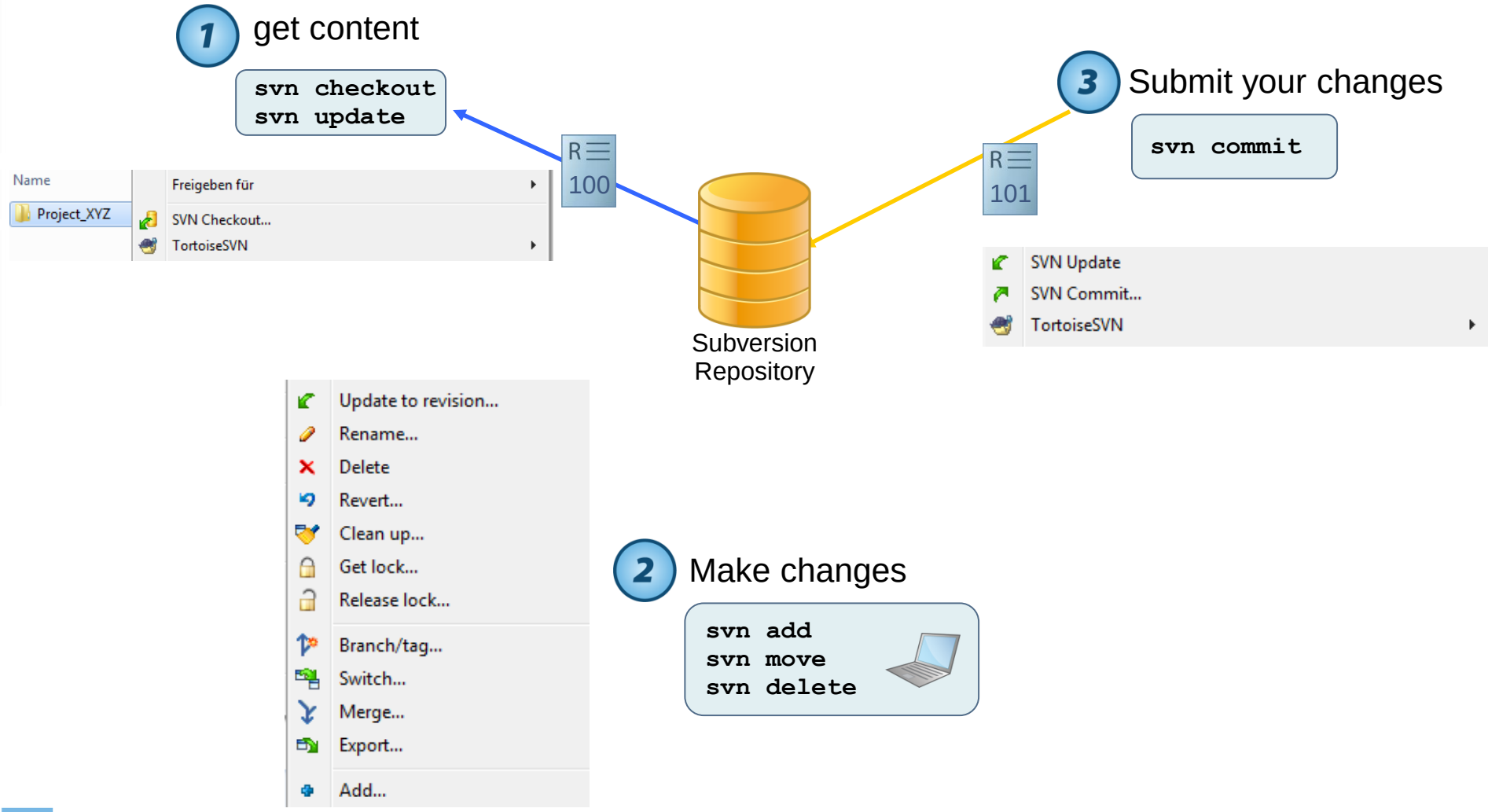
```
http://svn/kunde
└─Produkt
    ├─trunk                                 (Produktverzeichnis)
    │  ├─system                             (Systemverzeichnis)
    │  │    kkkkk_pppp_tttt_CMP.txt         (Konfigurationsmanagementplan)
    │  │    Produktelement 1                (Datei)
    │  │    Produktelement 2                (Verzeichnis)
    │  │
    │  ├─Teilprodukt_1                      (Teilproduktverzeichnis)
    │  │  └─system                          (Systemverzeichnis)
    │  │       Produktelement 1             (Datei)
    │  │       ├─Produktelement 2           (Verzeichnis)
    │  │       ├─Produktelement 3           (Verzeichnis)
    │  │       └─Produktelement 3           (Verzeichnis)
    │  │
    │  └─Teilprodukt_2                      (Teilproduktverzeichnis)
    │     ├─system                          (Systemverzeichnis)
    │     │    Produktelement 1             (Datei)
    │     │    ├─Produktelement 2           (Verzeichnis)
    │     │    └─Produktelement 3           (Verzeichnis)
    │     │
    │     ├─Teilprodukt_A                   (Teilproduktverzeichnis)
    │     │  └─system                       (Systemverzeichnis)
    │     │       Produktelement 1          (Datei)
    │     │       └─Produktelement 2        (Verzeichnis)
    │     └─Teilprodukt_B                   (Teilproduktverzeichnis)
    │        └─system                       (Systemverzeichnis)
    │             Produktelement 1          (Datei)
    │             ├─Produktelement 2        (Verzeichnis)
    │             └─Produktelement 3        (Verzeichnis)
    │
    ├─tags                                  (Produktversionsverzeichnis)
    │  ├─V0.1                               (V-Versionsverzeichnis)
    │  │    kkkkk_pppp_tttt_VDV.txt         (Versiondatei)
    │  │
    │  ├─P0.1                               (P-Versionsverzeichnis)
    │  │    kkkkk_pppp_tttt_VDP.txt         (Produktversionsdatei)
    │  │
    │  ├─Teilprodukt_1                      (Teilproduktversionsverzeichnis)
    │  │  └─V0.1                            (V-Versionsverzeichnis)
    │  │       kkkkk_pppp_tttt_VDV.txt      (Versiondatei)
    │  │
    │  ├─Teilprodukt_2                      (Teilproduktversionsverzeichnis)
    │  │  └─V0.2                            (V-Versionsverzeichnis)
    │  │       kkkkk_pppp_tttt_VDV.txt      (Versiondatei)
    │  │
    │  └─Teilprodukt_2                      (Teilproduktversionsverzeichnis)
    │     ├─Teilprodukt_A                   (Teilproduktversionsverzeichnis)
    │     └─Teilprodukt_B                   (Teilproduktversionsverzeichnis)
    │
    └─branches                              (Branchverzeichnis)
       ├─BV0.1                              (V-Branchverzeichnis)
       │
       ├─Teilprodukt_1                      (Teilproduktbranchverzeichnis)
       │  └─BV0.1                           (V-Branchverzeichnis)
       │
       ├─Teilprodukt_2                      (Teilproduktbranchverzeichnis)
       │  └─BV0.2                           (V-Branchverzeichnis)
       │
       └─Teilprodukt_2                      (Teilproduktbranchverzeichnis)
          ├─Teilprodukt_A                   (Teilproduktbranchverzeichnis)
          └─Teilprodukt_B                   (Teilproduktbranchverzeichnis)
```

The working cycle
-single person-

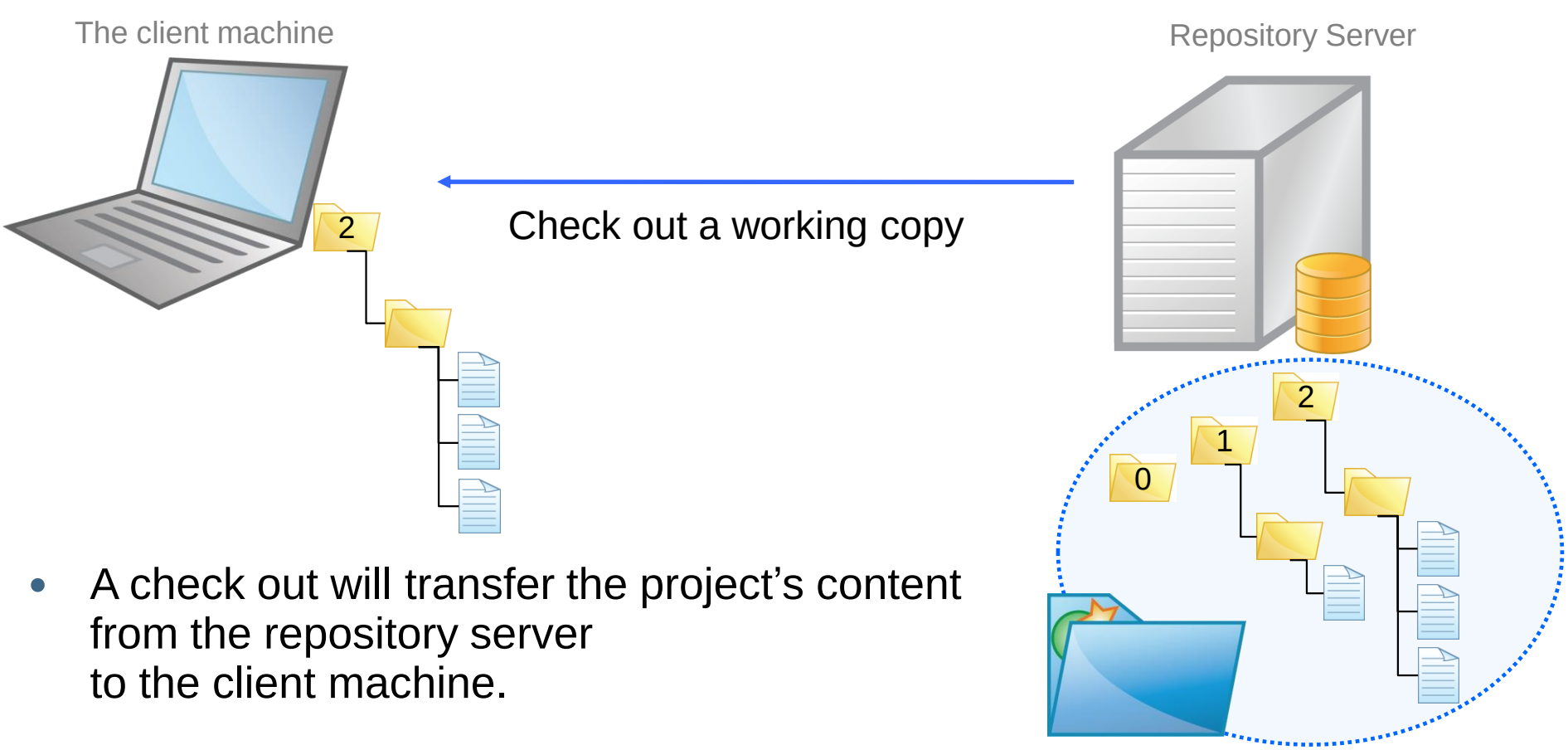
The Working Cycle
-single person-

Creating safety.
With passion.

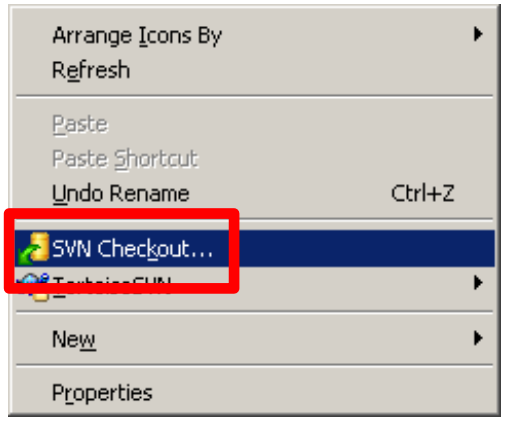
NewTec
System-Entwicklung und Beratung

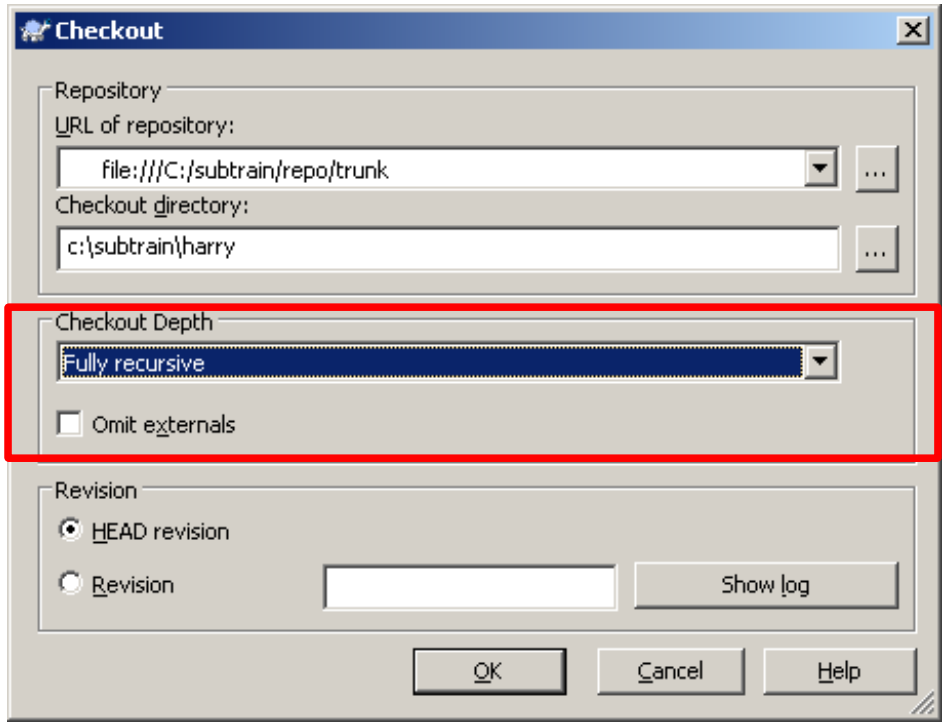


The Working Cycle – Check Out



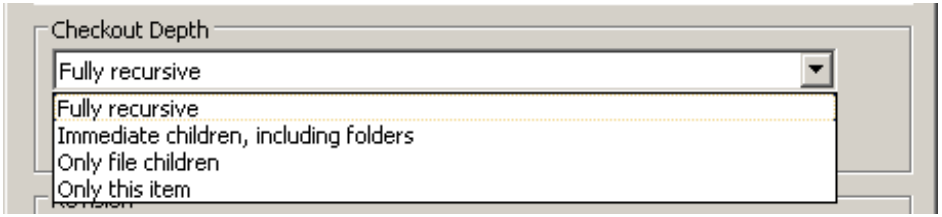
- A check out will transfer the project's content from the repository server to the client machine.

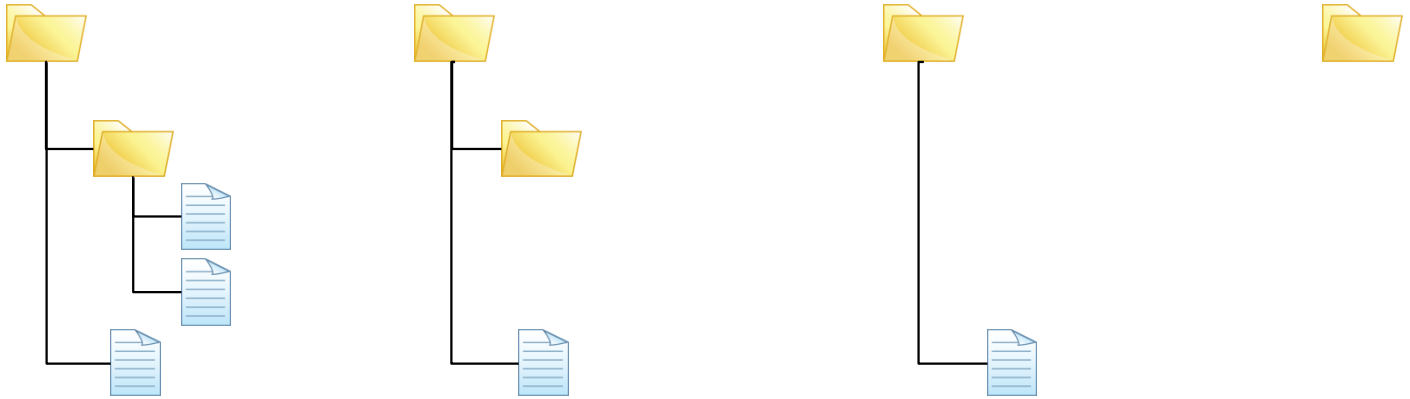





```
svn checkout file:///c:/subtrain/repo/trunk c:\subtrain\harry
```

The Working Cycle
Sparse Check Out





Fully recursive

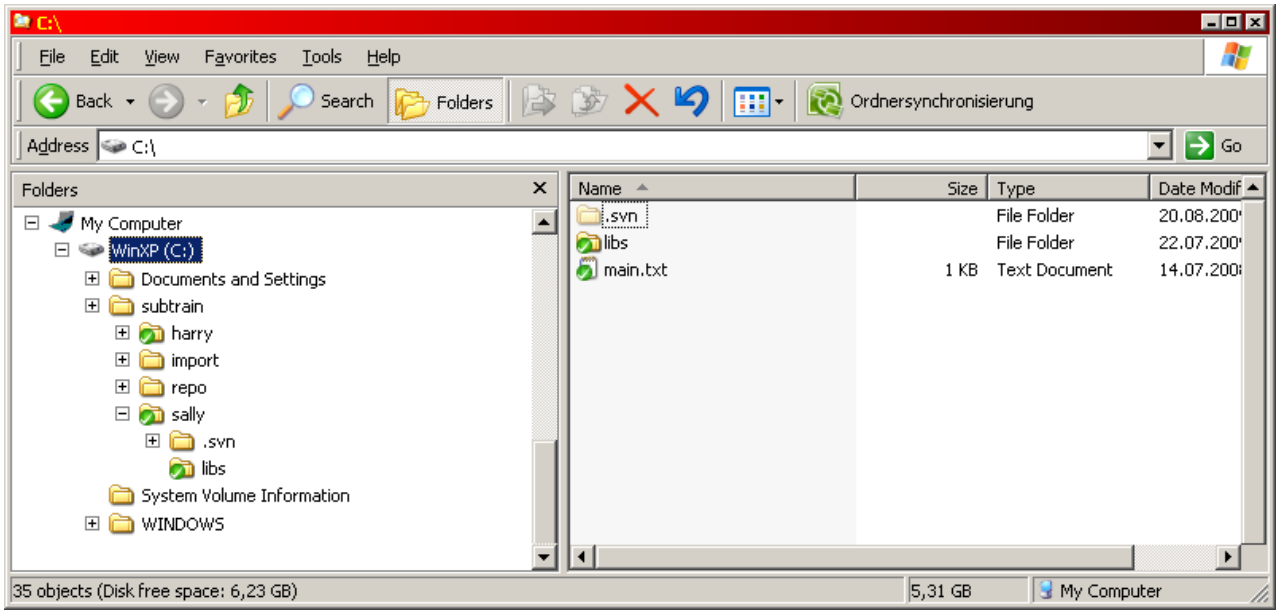
Immediate children, including folders

Only file children

only this item

Important for restructuring repository (eg. /branches directory)!

The Working Cycle
TortoiseSVN Icon Overlays



normal

deleted

conflict

modified

non-versioned

locked

added

ignored

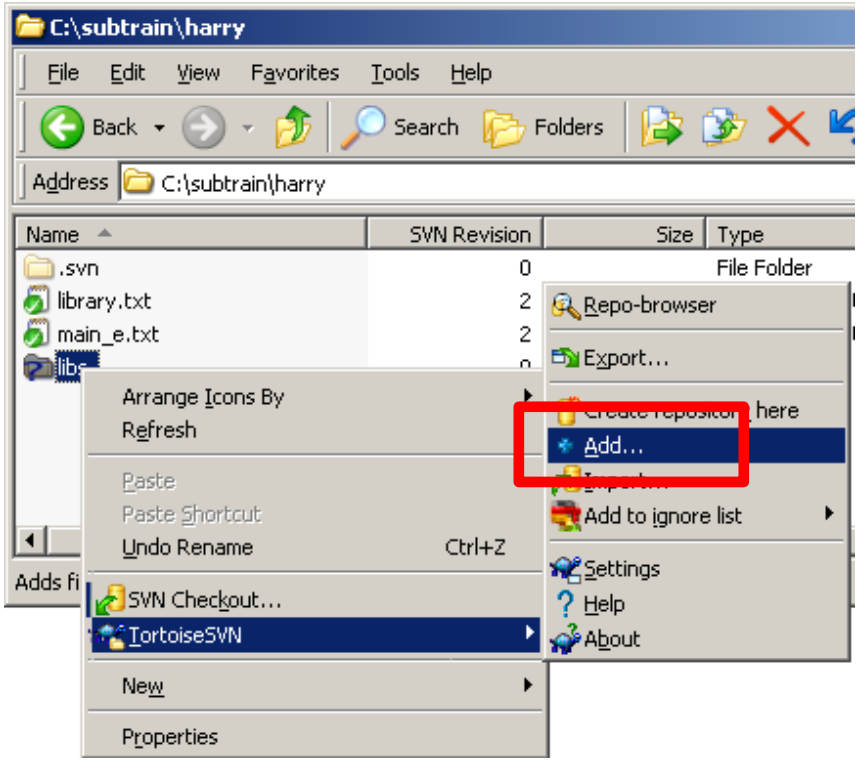
read only

```
svn status c:\subtrain\harry
```

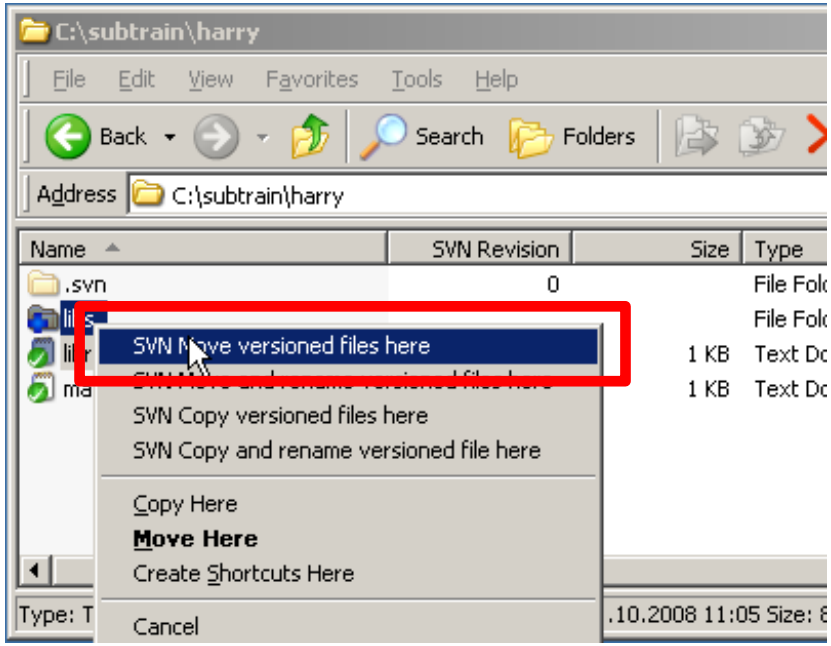
The Working Cycle – Adding and moving files

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung



add folder/files via context menu to SVN

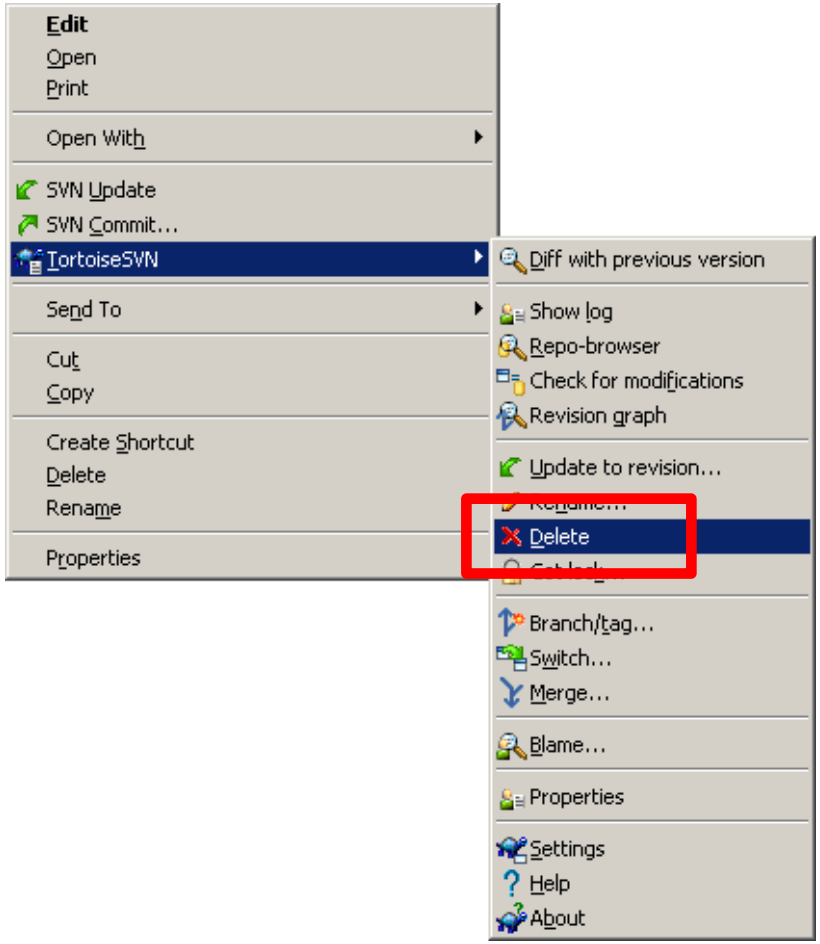


move or copy folder/files via dragging with **right** mouse button

```
svn add c:\subtrain\harry\libs
```

```
svn move
 c:\subtrain\harry\library.txt
 c:\subtrain\harry\libs\library.txt
```

The Working Cycle – Deleting files



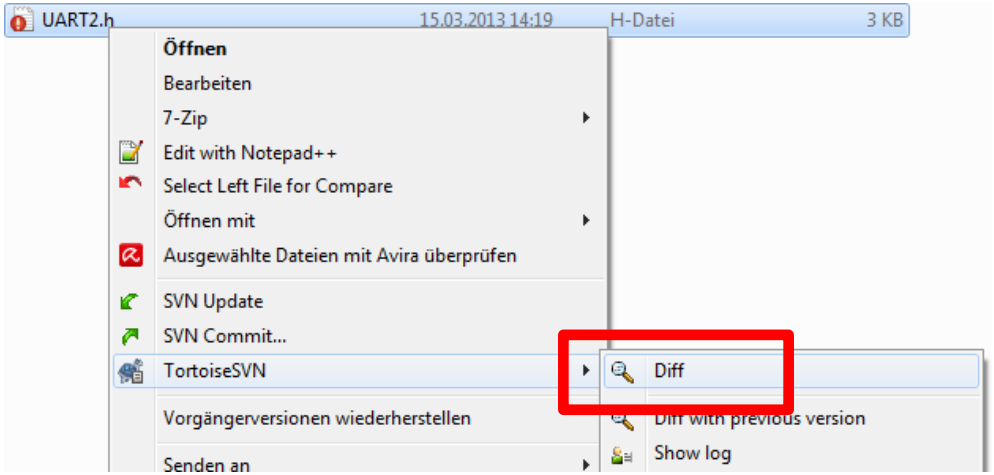
- files will immediately be removed from your working copy
- folders will be marked as deleted until commit

```
svn delete c:\subtrain\harry\file2del.txt
```

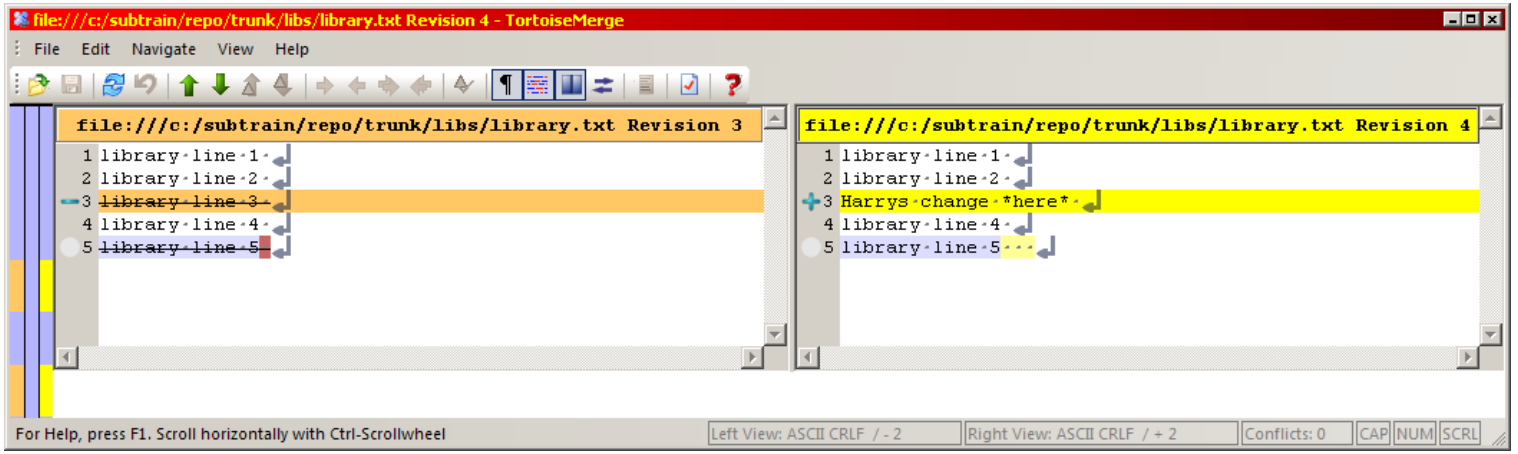
The Working Cycle – Comparing files

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung



It is also possible to tell TortoiseSVN to use external diff tools like BeyondCompare

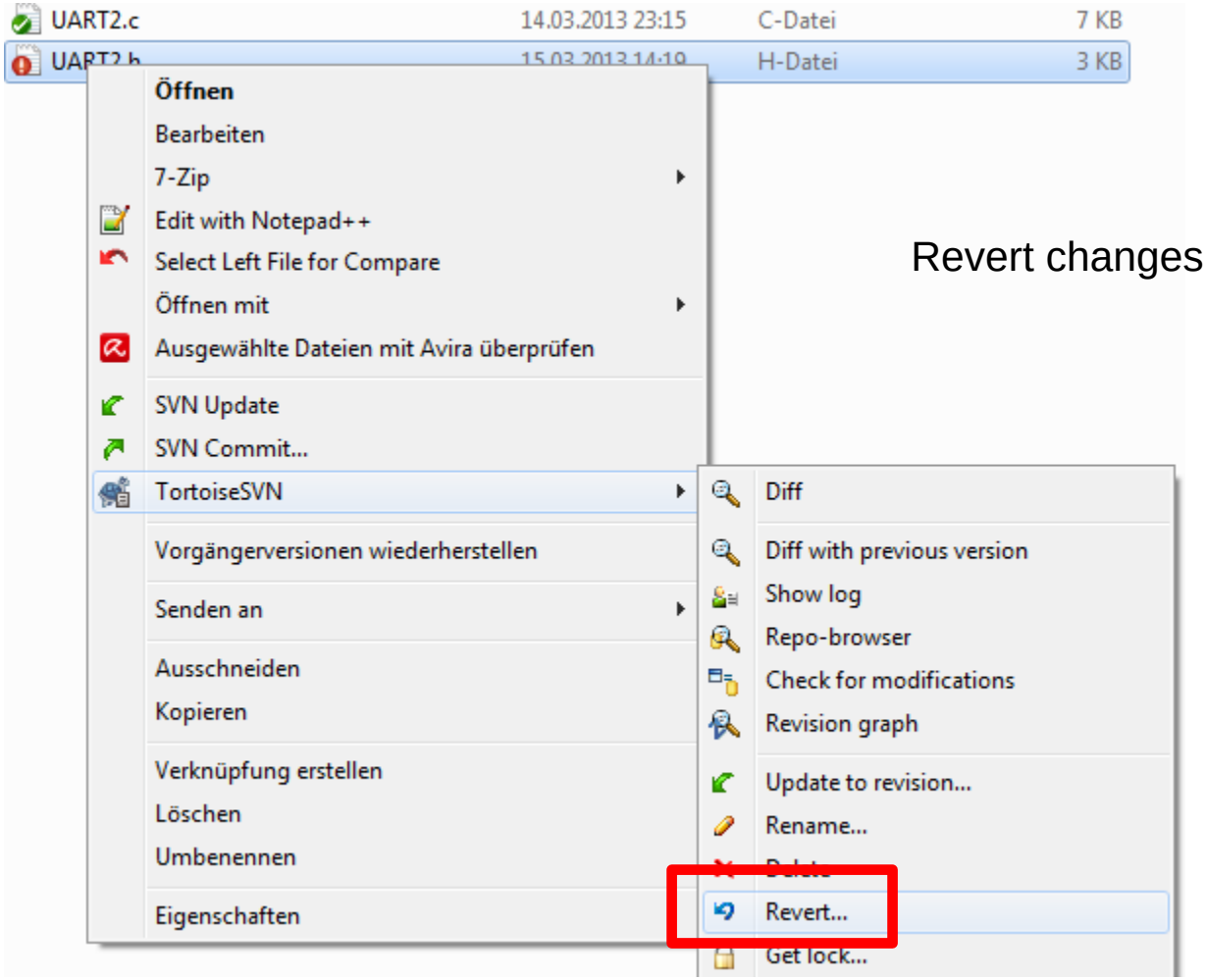



```
svn diff c:\subtrain\harry\libs\library.txt
```

The Working Cycle – revert changes

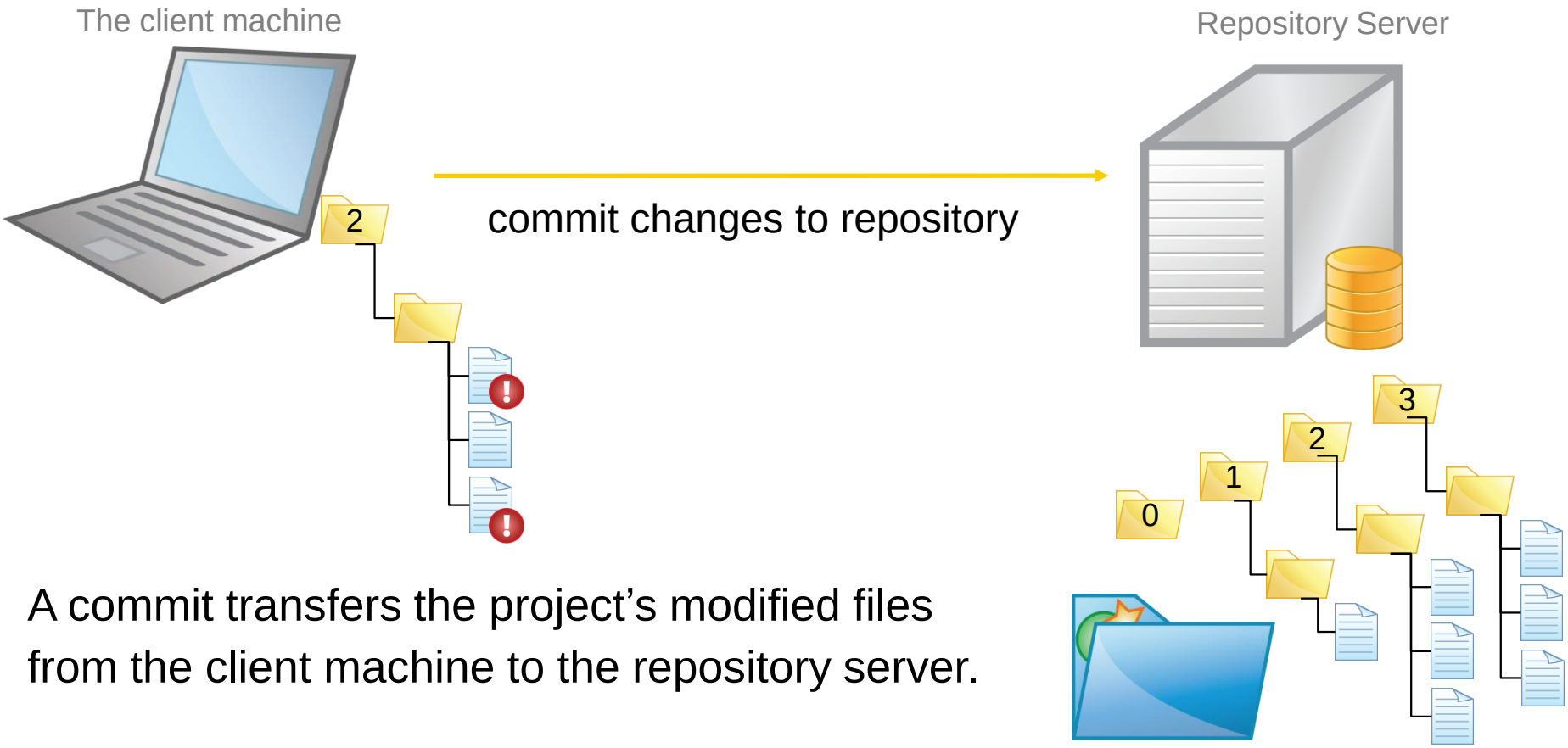
Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung



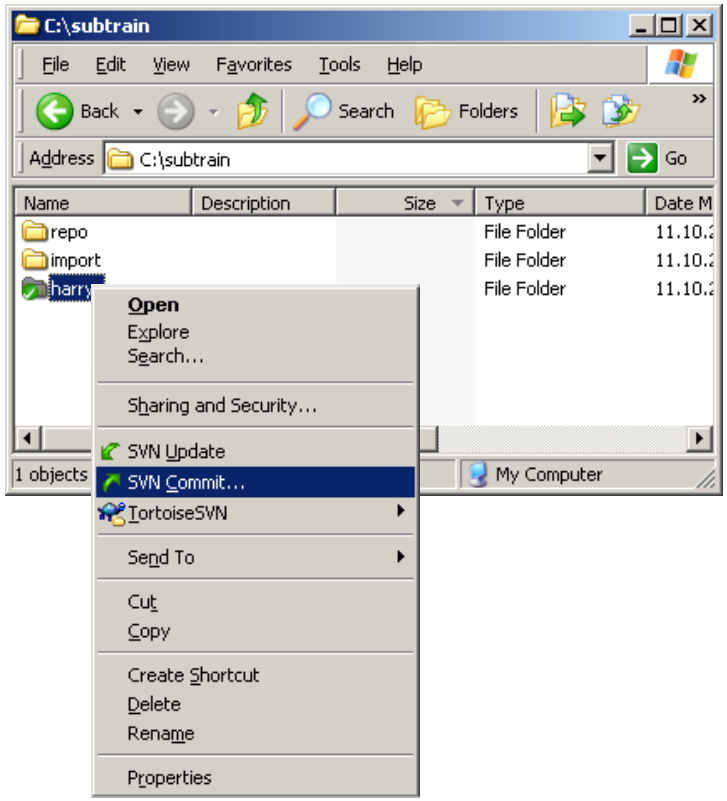
Revert changes to checkout state

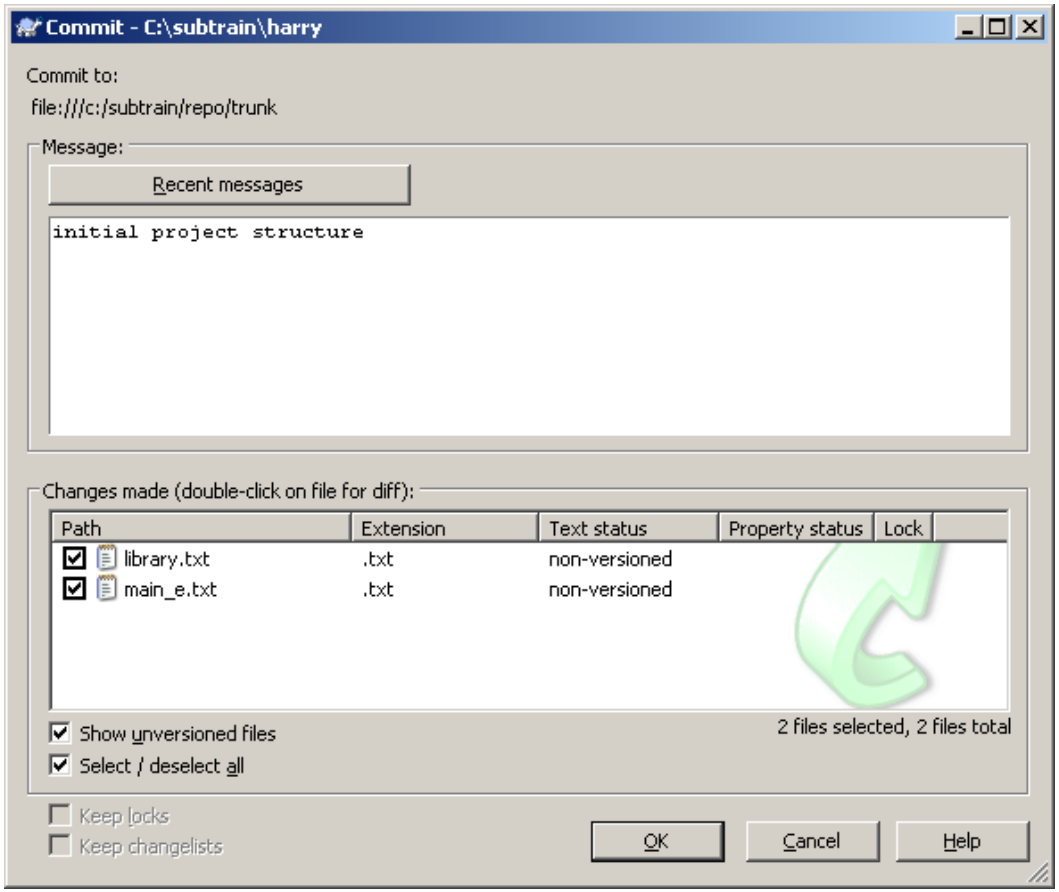
The Working Cycle - Commit



A commit transfers the project's modified files from the client machine to the repository server.

The Working Cycle – Commit



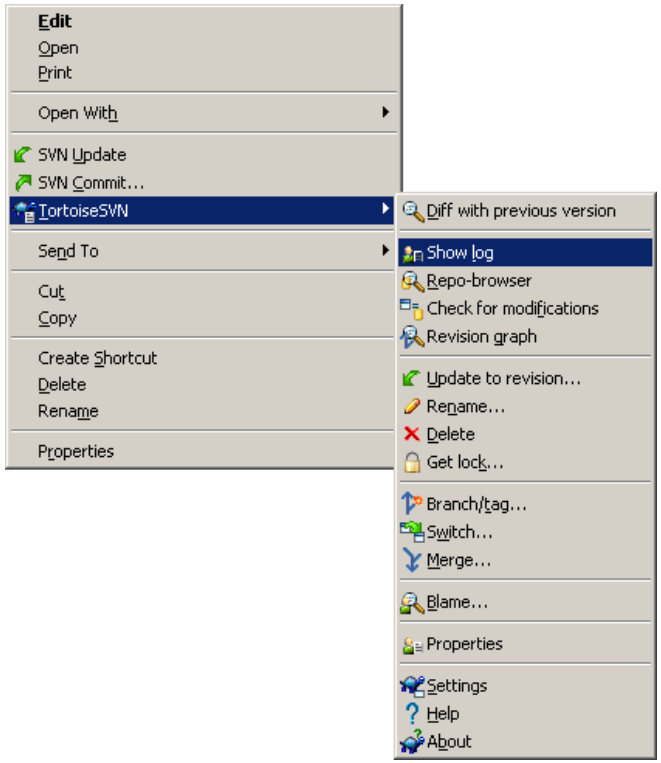


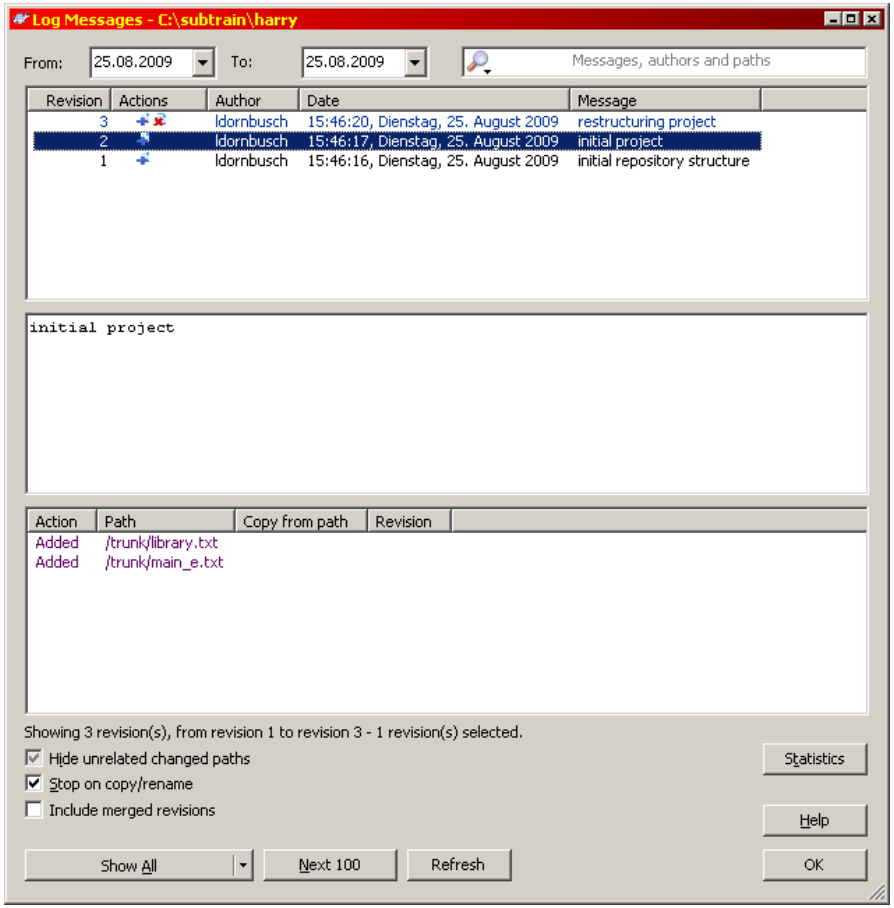

```
svn commit -m"initial project" c:\subtrain\harry
```

The Working Cycle – Show log / show history

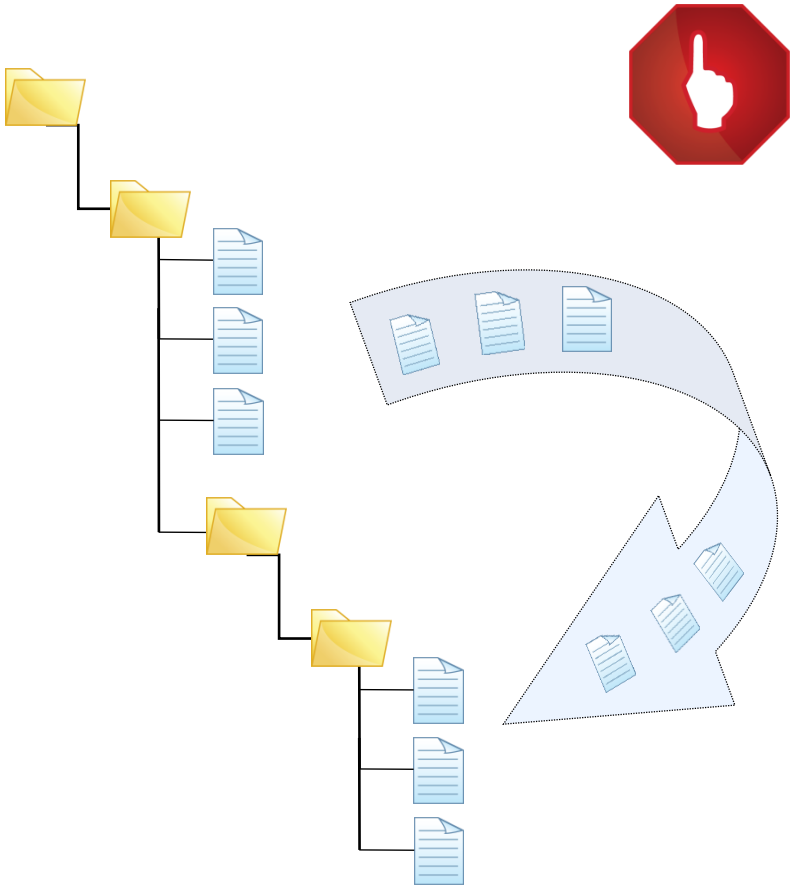
Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung






```
svn log c:\subtrain\harry
```



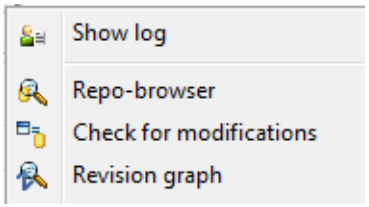
WARNING

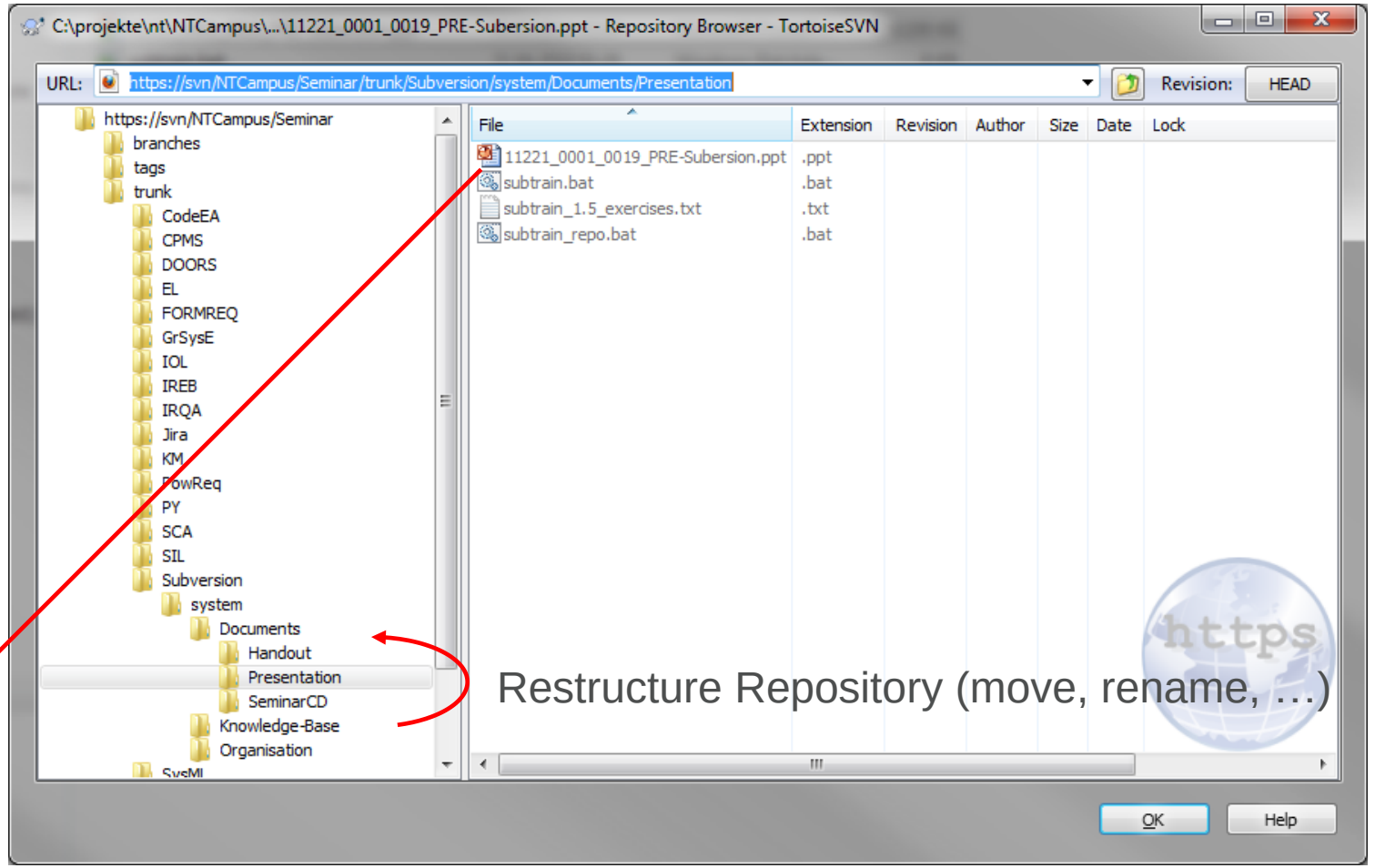
- Never move, delete, copy files or create directories without the Subversion commands:
  - svn move
  - svn delete
  - svn copy
  - svn mkdir (Tortoise 1.6)
- Easy if using TortoiseSVN as it is integrated directly in Windows Explorer, or Subversive integrated into Eclipse

The Working Cycle – Repo Browser

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung





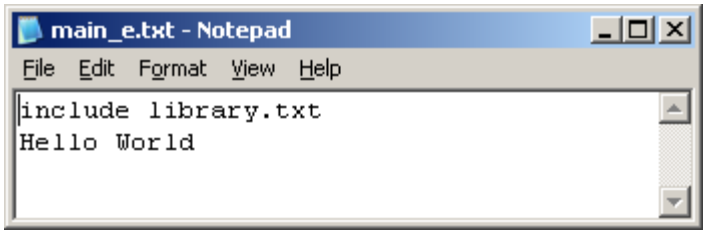
Drag & Drop files to Explorer

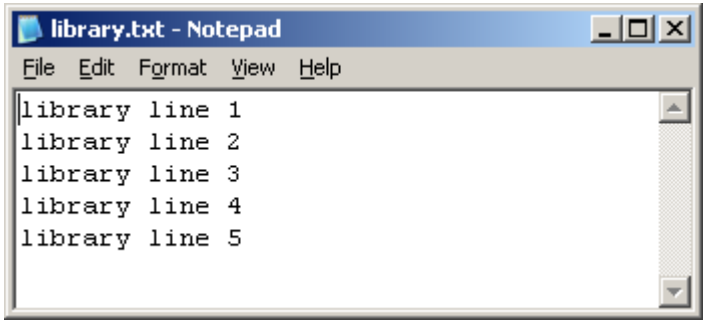
Restructure Repository (move, rename, …)

The Working Cycle – Exercise I: adding files

- Execute ***subtrain_repo.bat***
- checkout trunk, into a folder named „harry“ (file:///C:/subtrain/repo/trunk/)

- create two files in your working copy (create the following content):
- **`main_e.txt`**
- **`library.txt`**





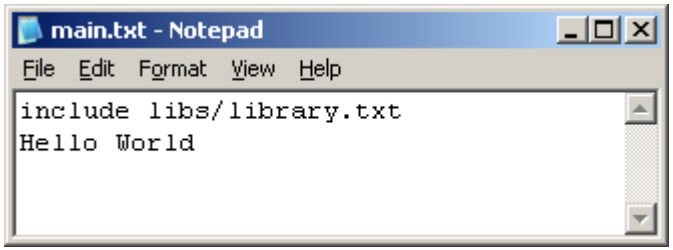
- commit your changes as „initial project“

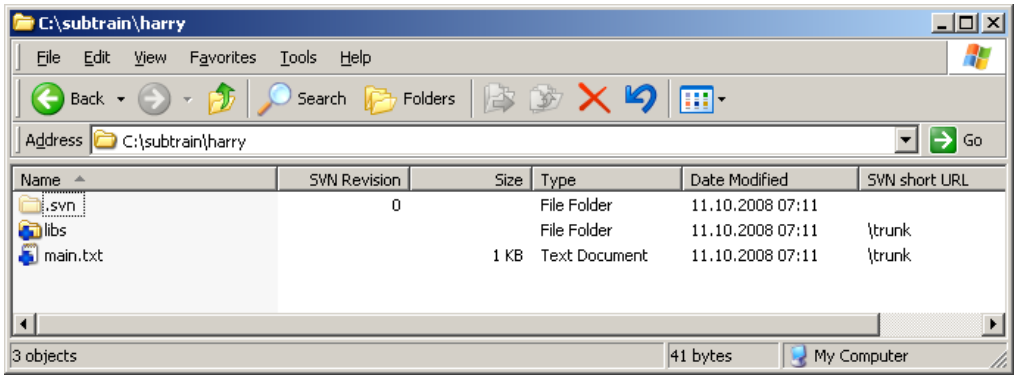
The Working Cycle – Exercise II: restructuring files

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung

- rename main_e.txt into main.txt
- create a folder named libs and add it to SVN
- move library.txt into libs (modify main.txt accordingly!)
- commit your changes as „restructuring project“





The working cycle
-teamwork-

The Working Cycle – Teamwork

1 get content

```
svn checkout
```

R 100

R 106

6 Submit your changes

```
svn commit
```

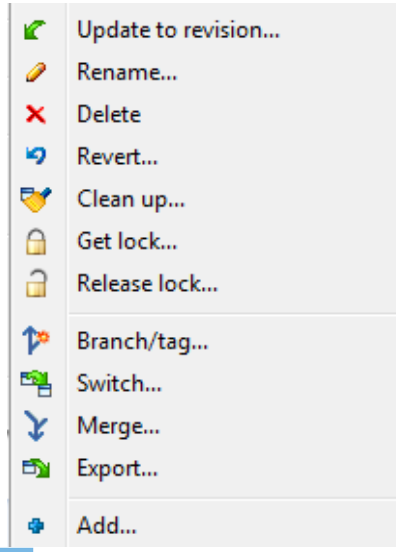
2 Make changes

```
svn add
svn move
svn delete
```

Subversion Repository

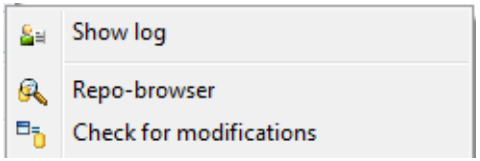
5 Merge your changes
Resolve conflicts

R 105



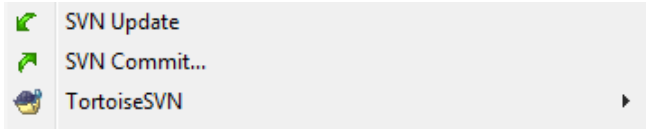
3 See what was changed in the repository in the meantime

```
svn status -u
```



4 Update your local copy

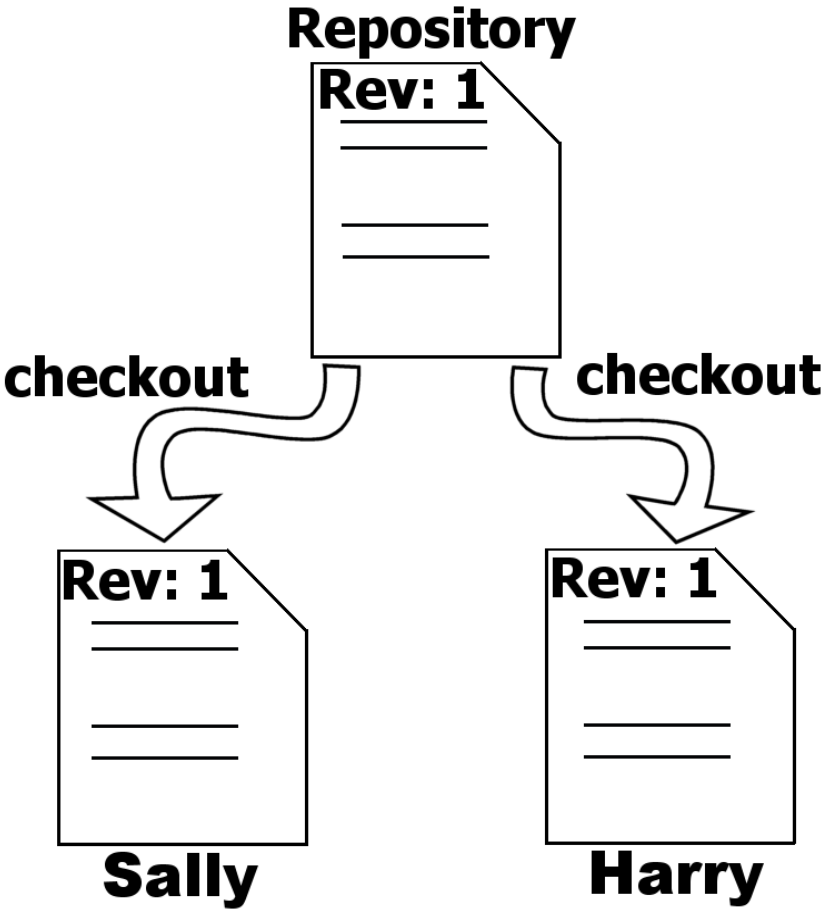
```
svn update
```



Harry and Sally both check out a directory in their repository.

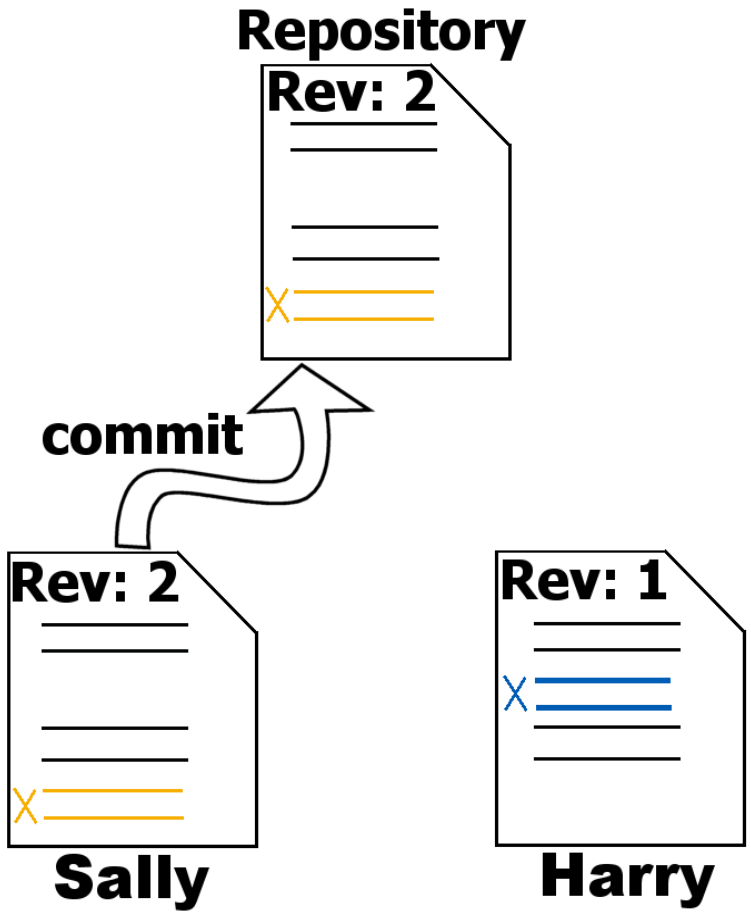
Both get the same revision of this particular directory copied to their machine.

Their **working copy.**



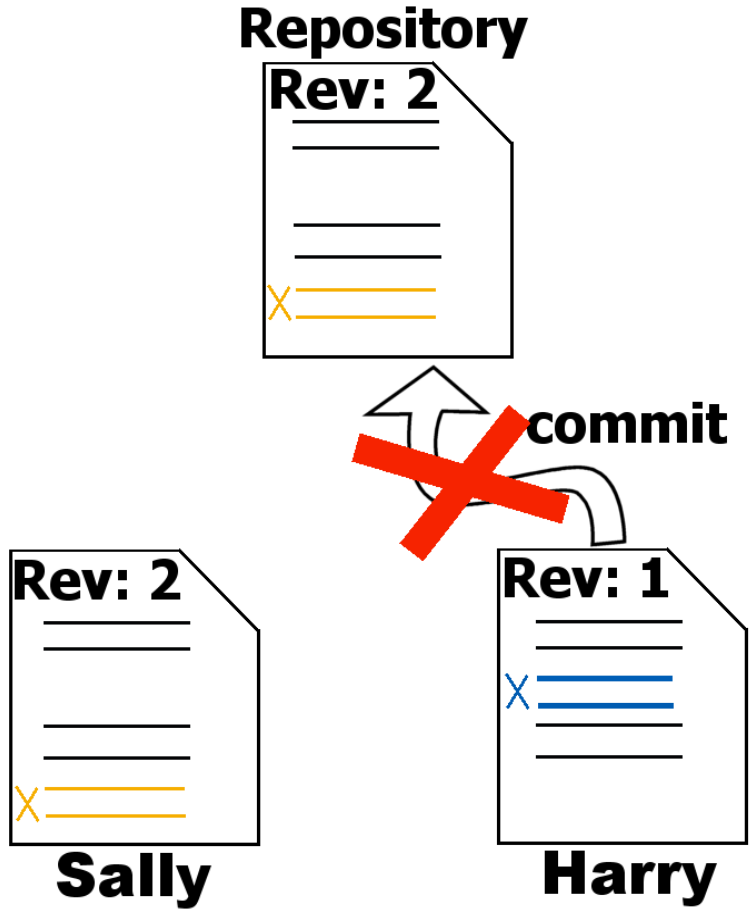
Harry and Sally change different lines in the same file.

Sally commits first and creates revision 2.

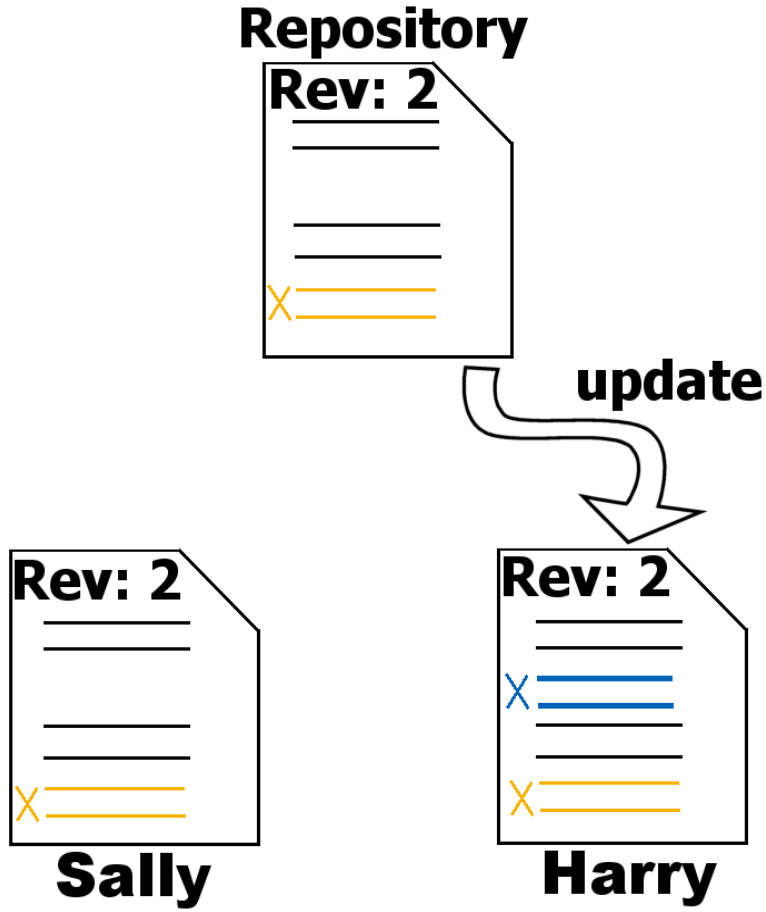


As Harry tries to commit his changes, he receives an error telling him that his working copy is probably out of date.

His commit fails.

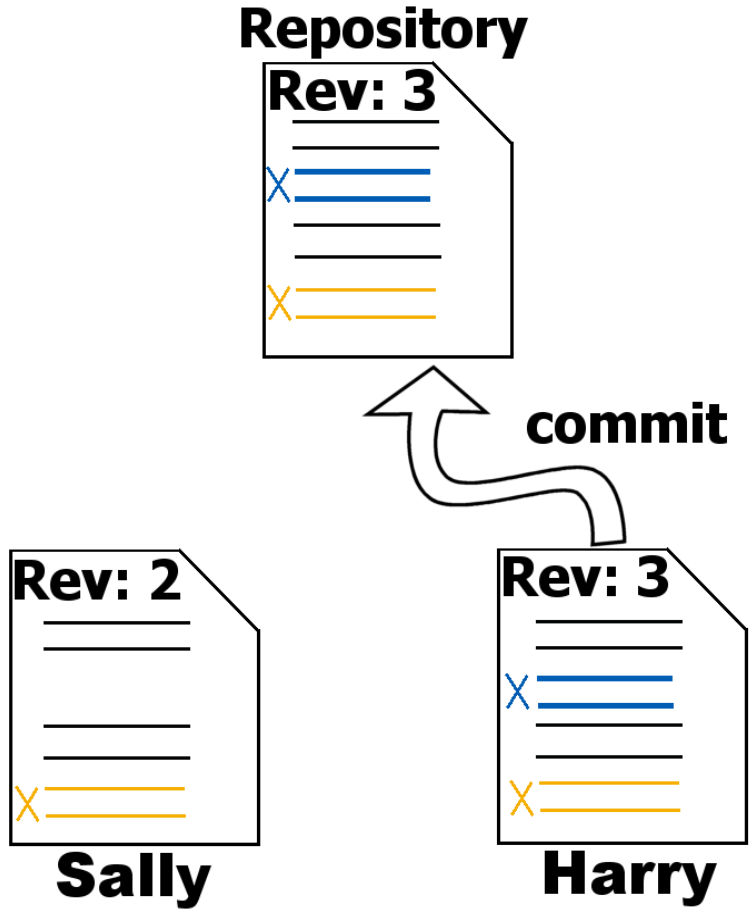


Harry has to update his working copy and SVN integrates Sallies changes into his working copy.



Now Harry is ready to commit his changes.

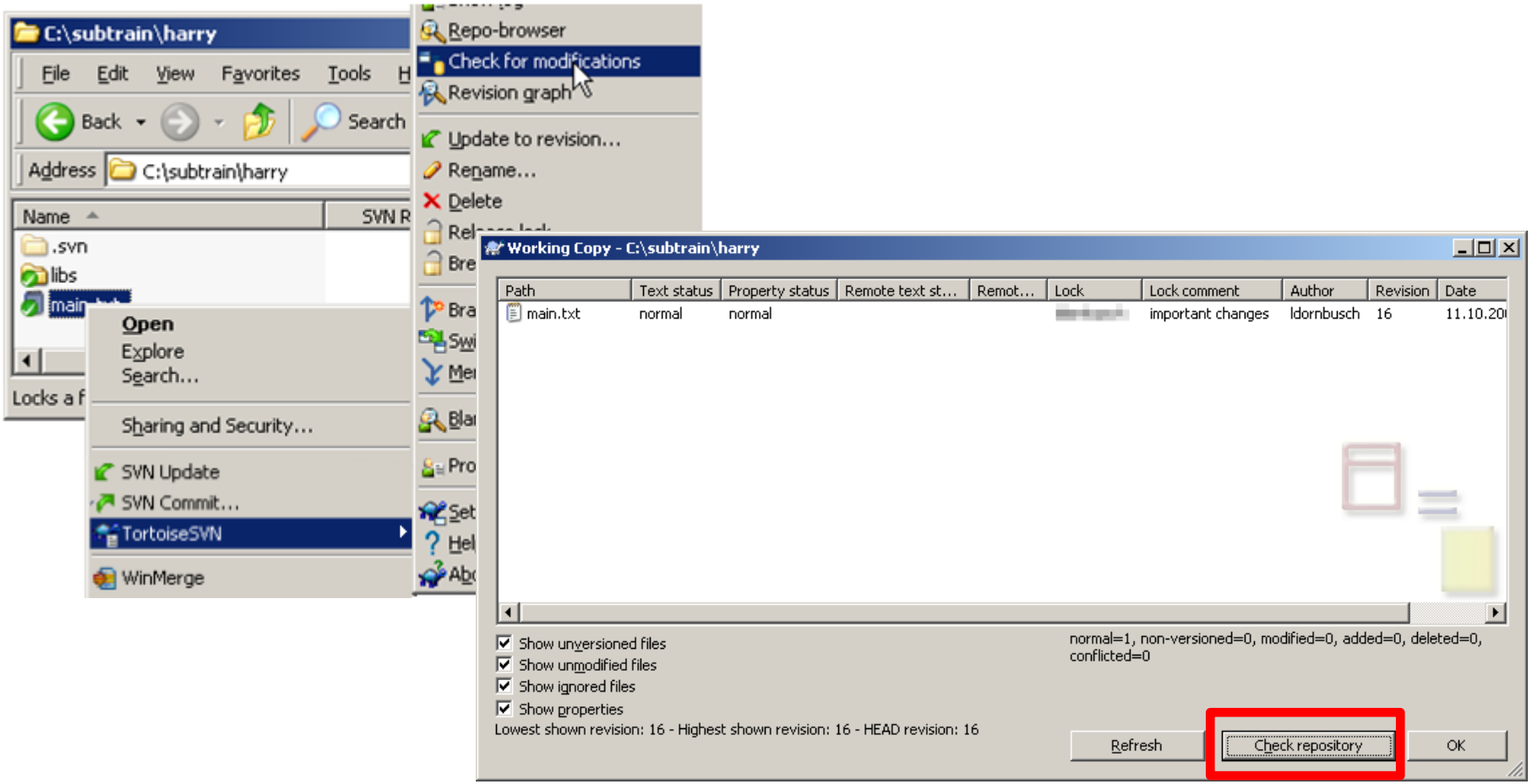
He creates revision 3.



The Working Cycle – Check what other users have changed

Creating safety.
With passion.

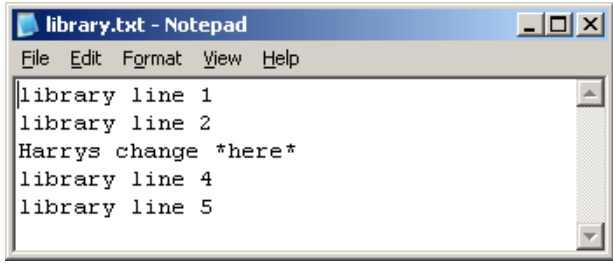
NewTec
System-Entwicklung und Beratung

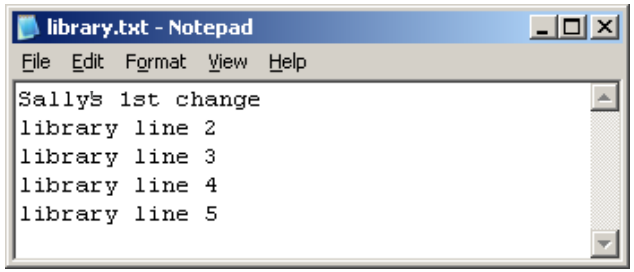



```
svn status -uv c:\subtrain\harry
```

The Working Cycle – Exercise III: teamwork

- check out trunk a second time into a folder called „Sally“
- change the third line in Harry's working copy
- commit Harry's changes as „Harry's 1st changes in our library“
- change the first line in Sally's working copy and try to commit
- update Sally's working copy
- commit Sally's changes as „Sally's 1st changes added“

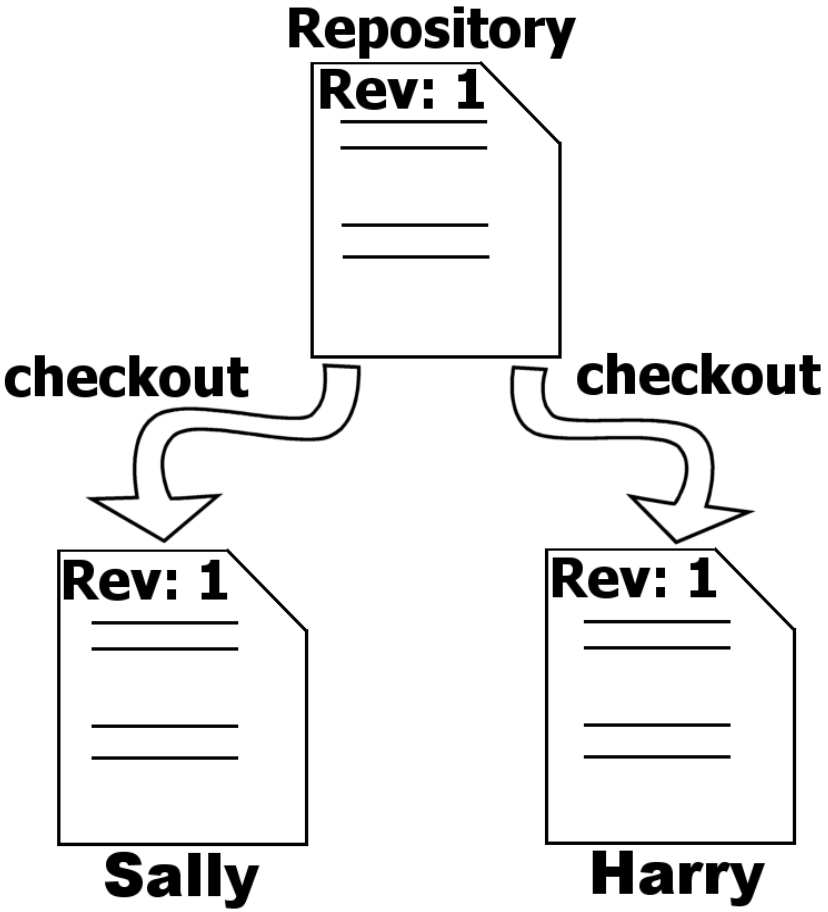




How does a conflict evolve?

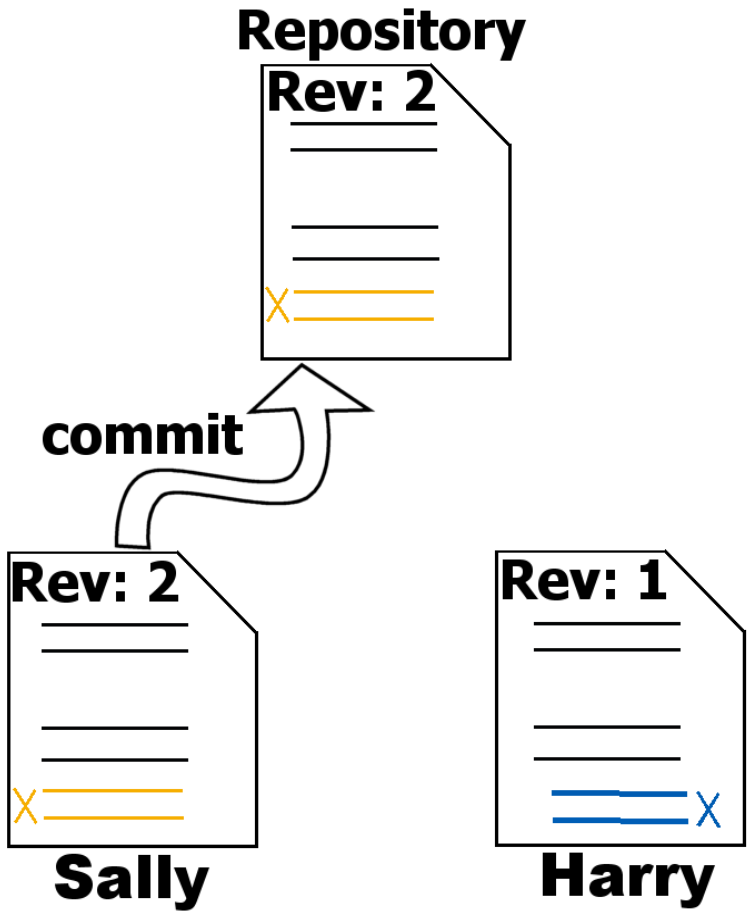
Harry and Sally check out their repo again.

This time they change the same lines in this particular file.

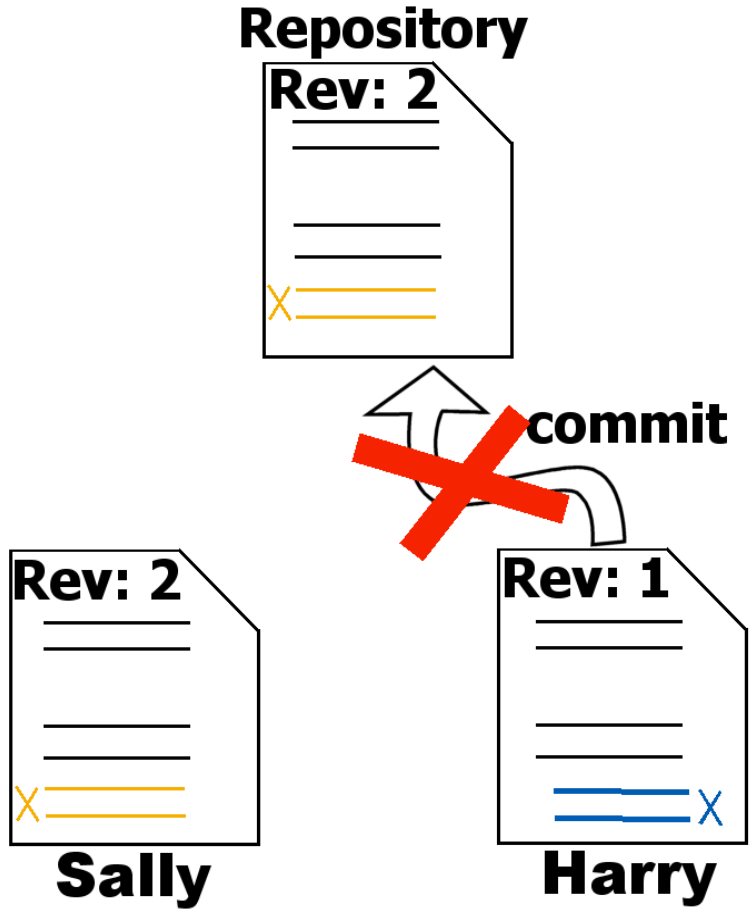


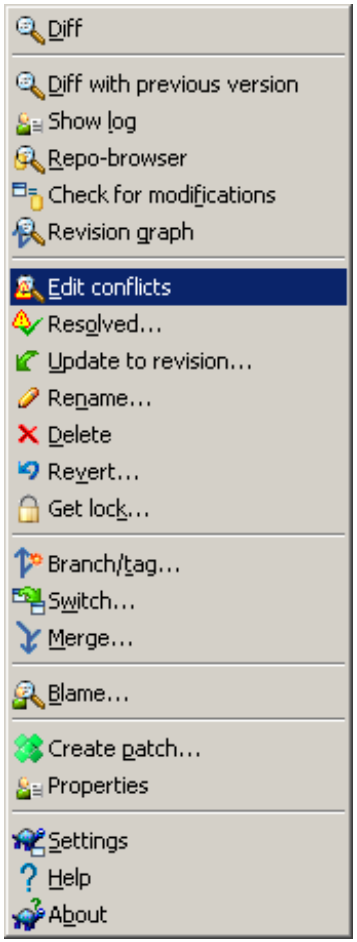
Sally commits her changes and creates a new revision.

Her changes are submitted to the repository



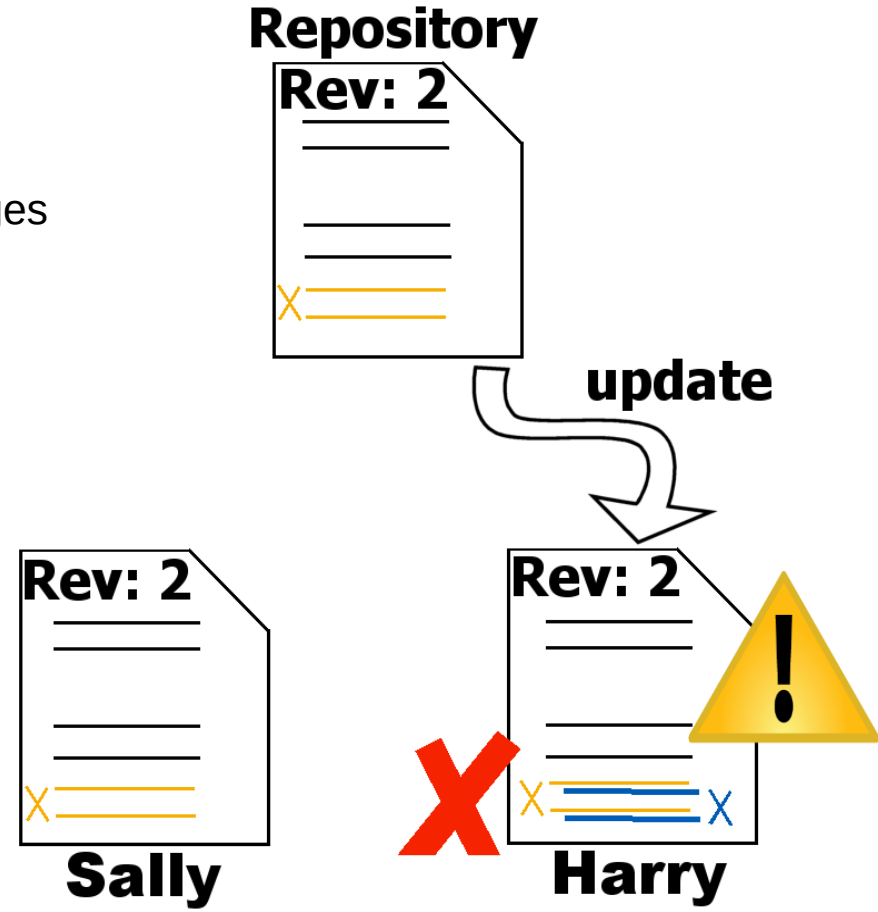
As Harry tries to commit his changes, he receives the well known “out-of-date”-error and an advice to update his working copy.

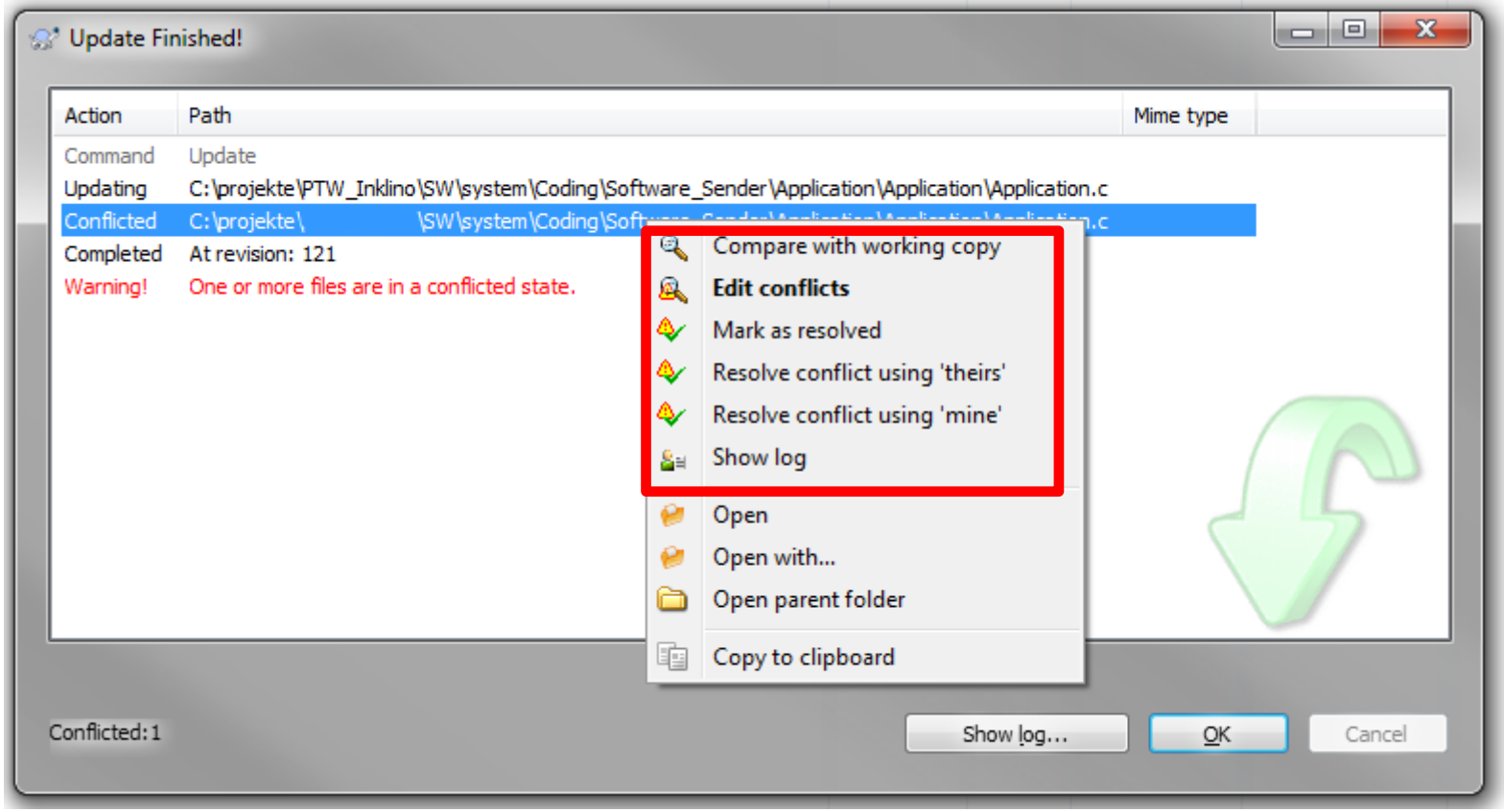




As both Harry and Sally had changed the same lines, SVN cannot decide how to **merge** Harry's changes and the changes from the repository.

So SVN raises a conflict which must be resolved by Harry.

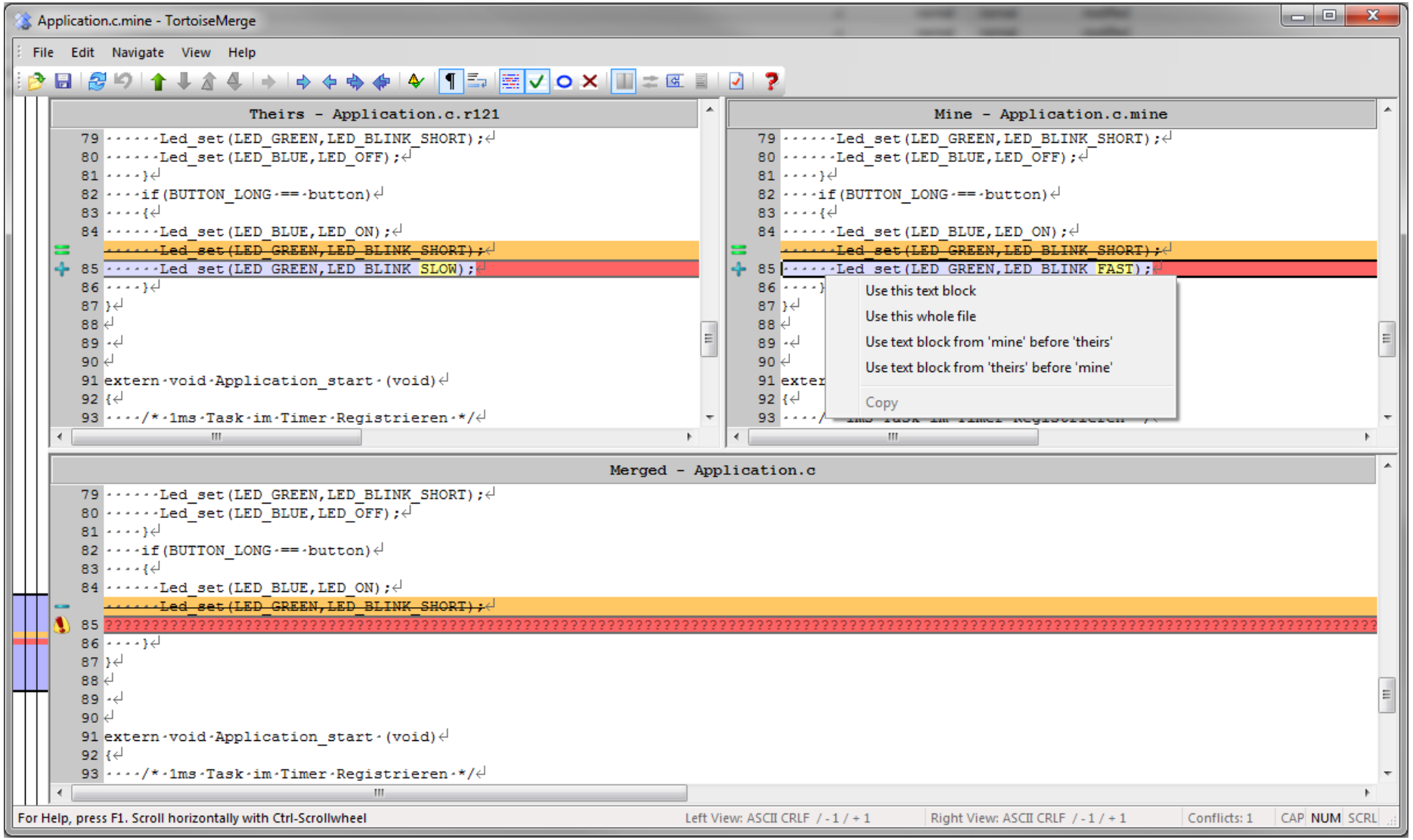


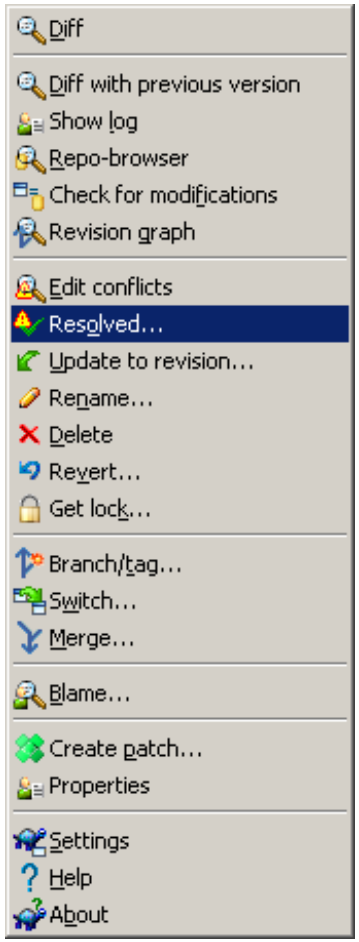

Update Finished!
Action
Path
Mime type
Command
Update
Updating
C:\projekte\PTW_Inklino\SW\system\Coding\Software_Sender\Application\Application\Application.c
Conflicted
C:\projekte\
\SW\system\Coding\Soft
Completed
At revision: 121
Warning!
One or more files are in a conflicted state.
Compare with working copy
Edit conflicts
Mark as resolved
Resolve conflict using 'theirs'
Resolve conflict using 'mine'
Show log
Open
Open with...
Open parent folder
Copy to clipboard
Conflicted: 1
Show log...
OK
Cancel

Teamwork – Conflicts 4/6b

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung

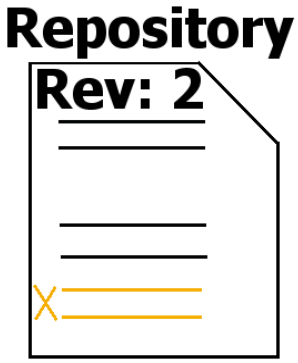


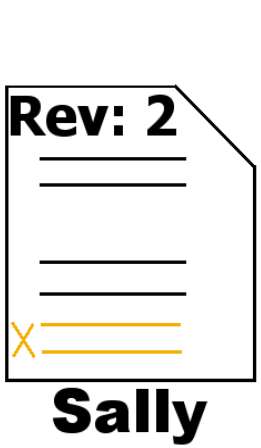


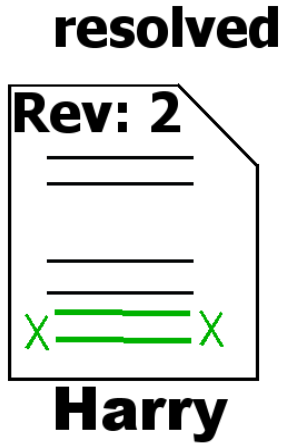
Harry cannot commit a file which is in a conflicted state.

Harry has to resolve the conflict/s and then notifies SVN via

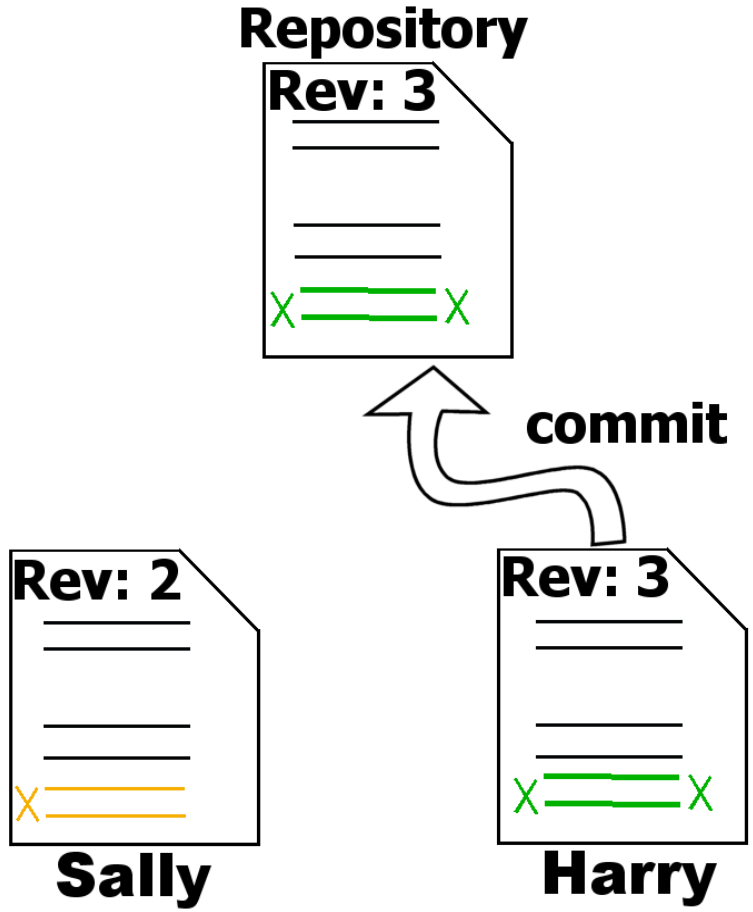
```
svn resolved
```





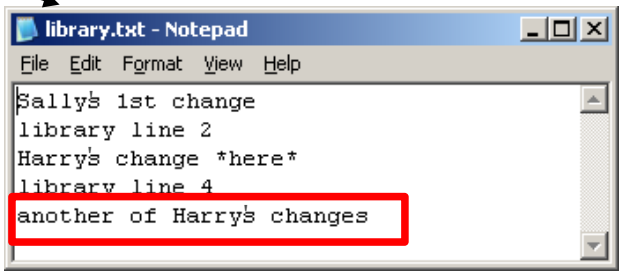


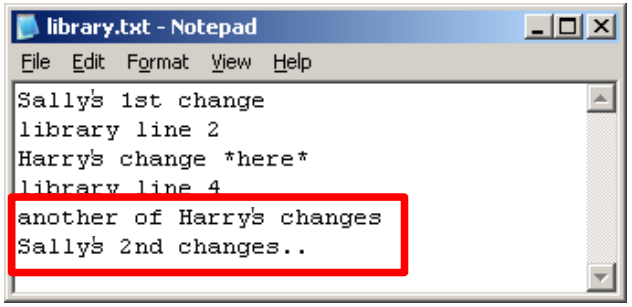
Harry commits his changes and creates a new revision.

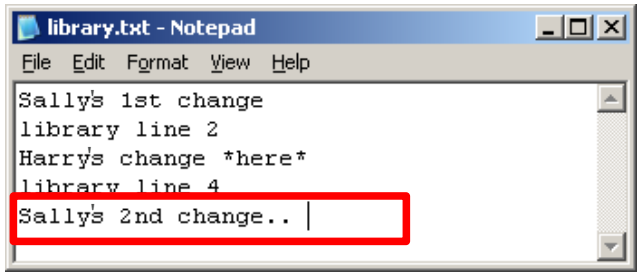


The Working Cycle – Exercise IV: conflicts

- update Harry's working copy to get Sally's last changes
- change the fifth (last) line in Harry's working copy
- commit Harry's changes as „Harry's 2nd changes in our library“
- change the fifth (last) in Sally's working copy and try to commit
- update Sally's working copy
- edit and resolve the conflict:
- commit Sally's changes as „Sally's changes added“







Locking

- **The default model of Subversion is the “copy-edit-merge” model.**

  This means:

  - Check out a working copy
  - Edit content
  - Merge changes from server
  - Commit changes

- **But there are situations in which this model might cause problems:**

  - Binary files
  - files which are frequently updated by many users

- **You can't use the "copy-modify-merge" model with files like Word or Excel (this applies for other types too) because you can't merge those files.**
- **So you have to find another way. There are two possible solutions:**

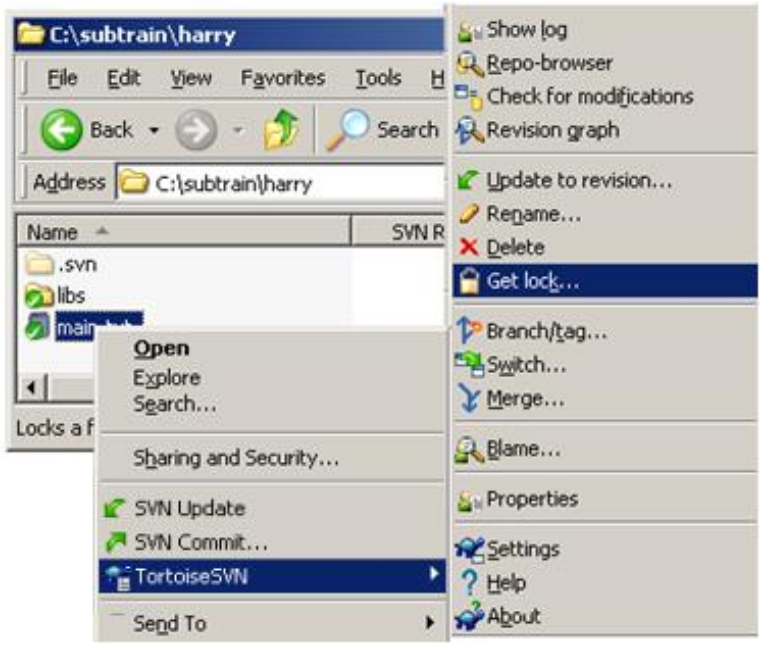
  - You *can* lock the file before you start to edit a file of the above type (called optimistic locking).
    You do not have to lock a file.

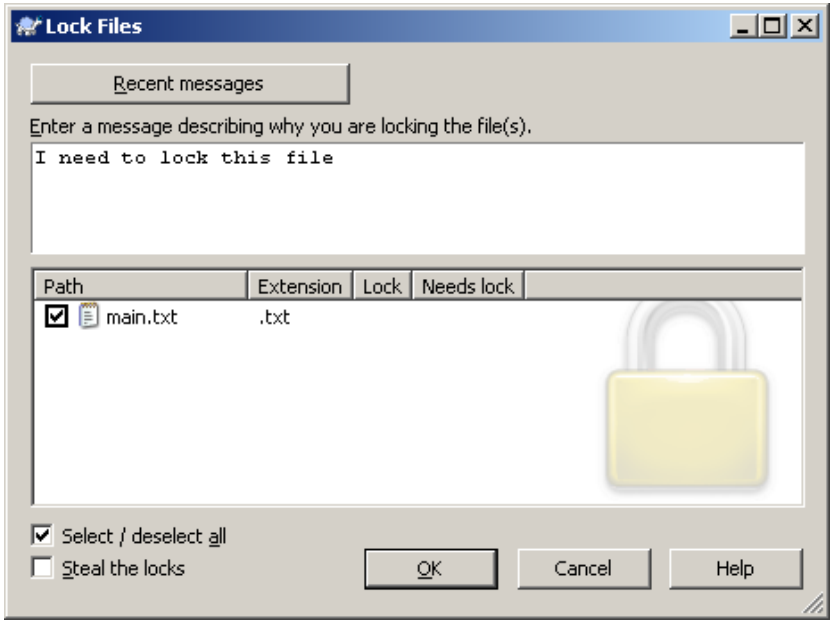
  - The second one is to use the "pessimistic-locking" alternative.

    You *have* to lock the file, otherwise it will be write-protected

Locking – Optimistic Locking: How to lock a file

- To lock a file just type the **`svn lock`** command with a particular destination.
- Use a descriptive lock message to tell your co-workers why you need this lock

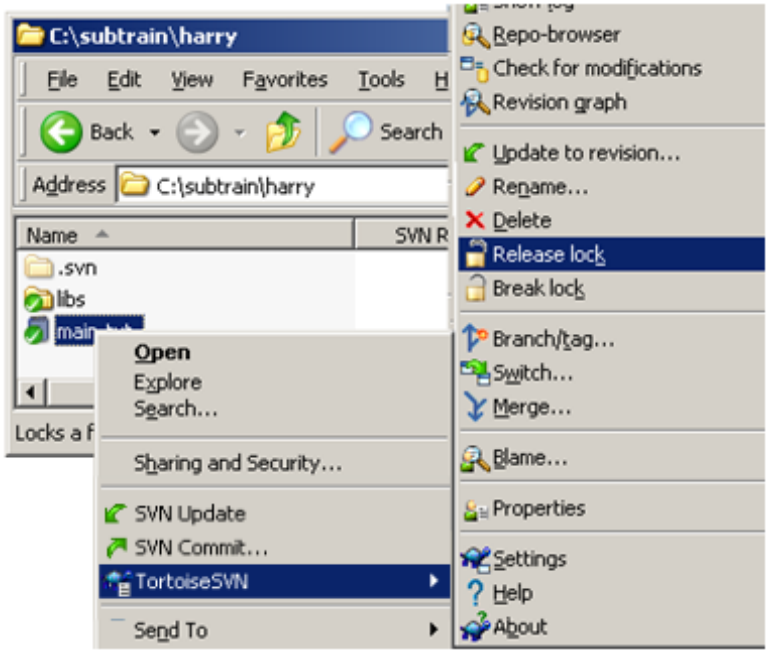





```
svn lock -m "I need to lock this file" c:\subtrain\harry\main.txt
```

Locking – Optimistic Locking: Unlock a file

- To unlock a file just type the **`svn unlock`** command with a particular destination.
- Locks will be released automatically if you commit your changes.



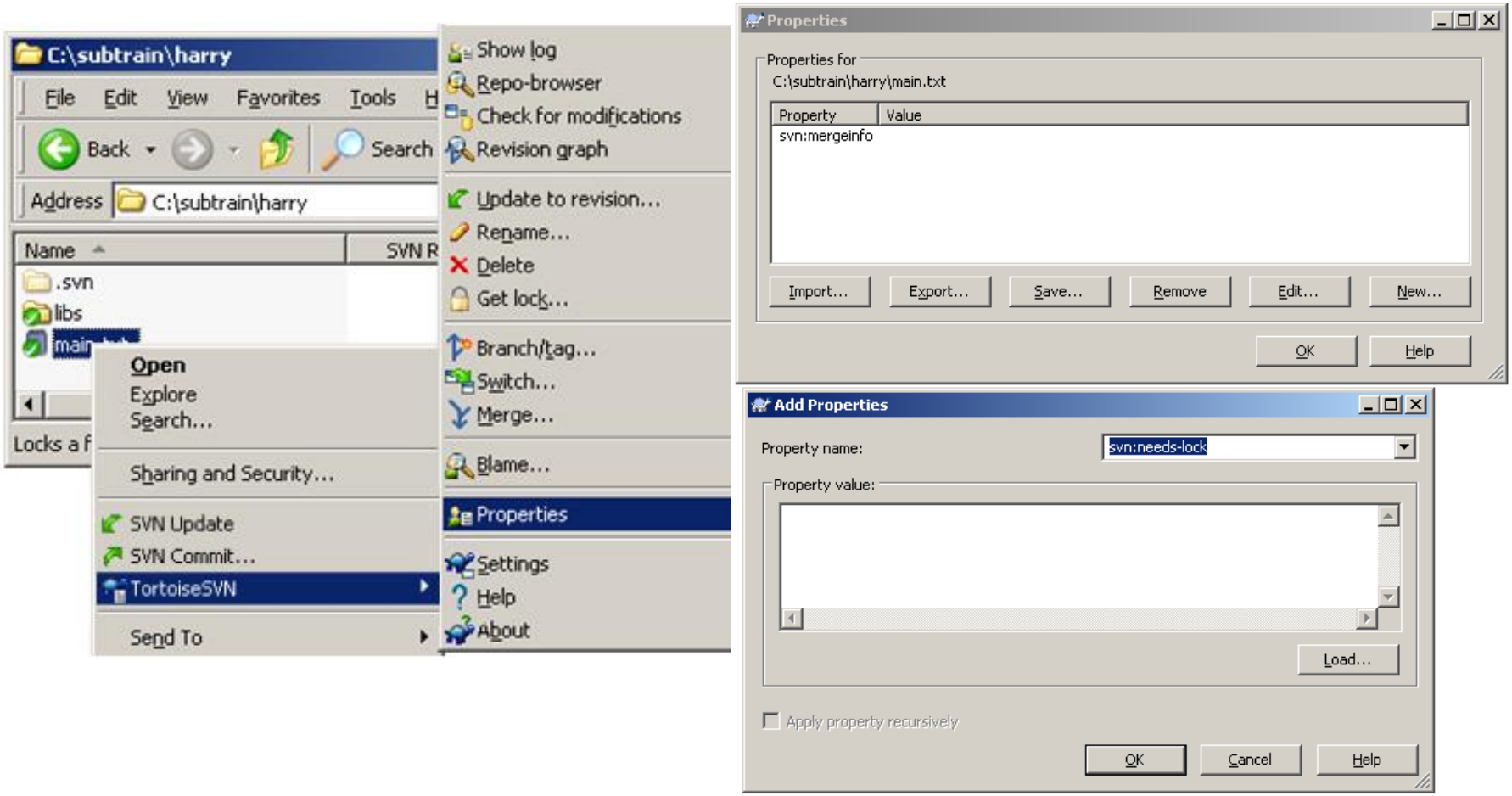

```
svn unlock c:\subtrain\harry\main.txt
```

- **Pessimistic-Locking:**
  - You can put a special property to a file **`svn:needs-lock`** with any value.
  - The result is if you check out a file with the property set, the file will be marked as read-only.
  - Advantage:
    - The application you use with this type of file will give you a reminder about the read-only state.
  - Disadvantage:
    - The system isn't flawless, either. It's possible that even when a file has the property, the read-only reminder won't always work. Sometimes applications misbehave and “hijack” the read-only file, silently allowing users to edit and save the file anyway. Unfortunately there's not much Subversion can do about this.

Locking – Pessimistic Locking

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung

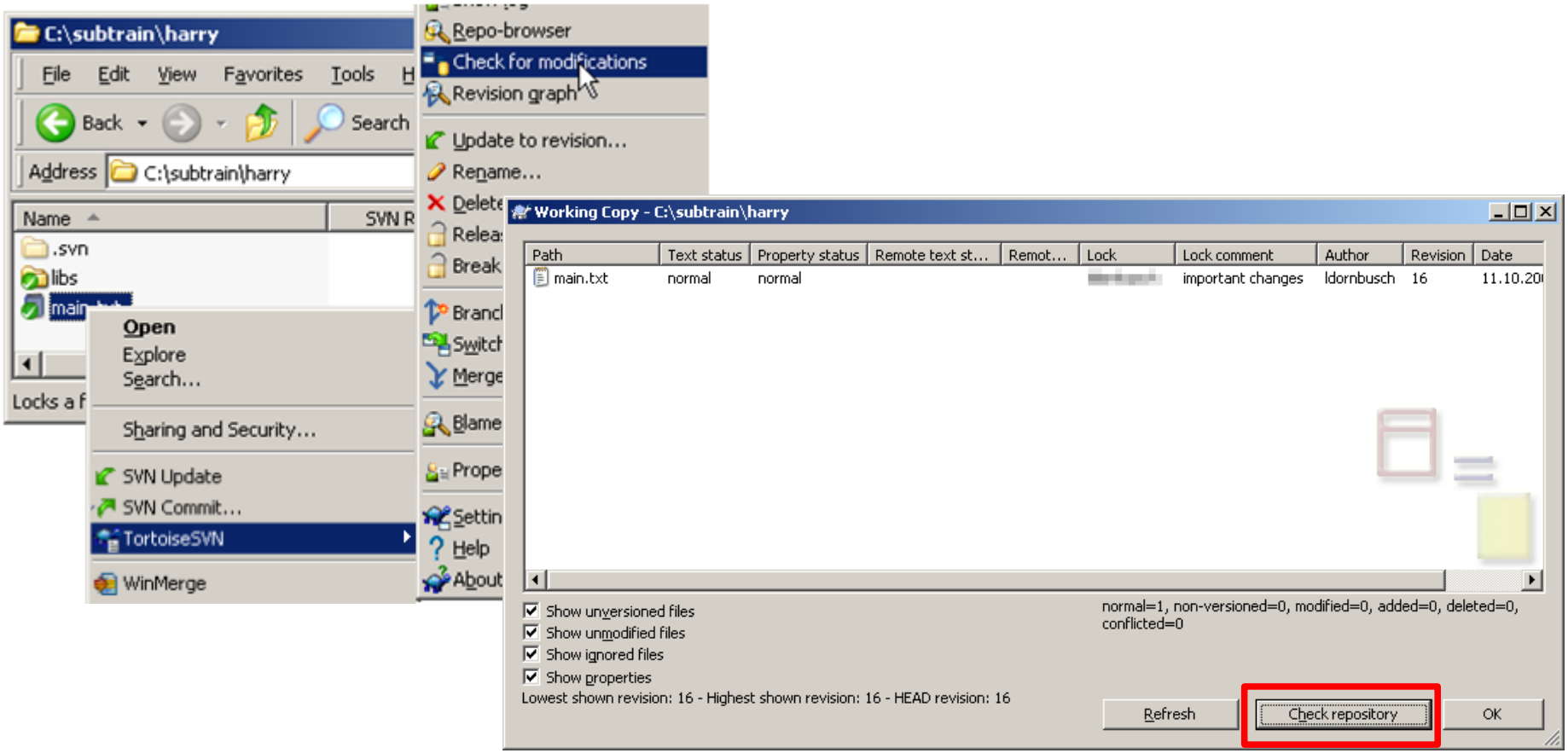



```
svn propset svn:needs-lock T c:\subtrain\harry\main.txt
```

Locking – Take a look on the server…

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung




```
svn status -uv c:\subtrain\harry
```

Locking – Exercise V: optimistic locking

- lock the main.txt in Harry's working copy
- change the main.txt in Sally's working copy
- try to commit Sally's changes
- change main.txt in Harry's working copy
- commit changes as „changes to locked file“
- revert Sally's changes
- update Sally's working copy

Locking – Exercise VI: pessimistic locking

- **set property svn:needs-lock on main.txt in Sally's working copy**
- **commit as "enabling pessimistic locking"**
- **update Harry's working copy**
- **try to change main.txt in Harry's working copy**
- **get lock on main.txt in Sally's working copy**
- **change Sally's main.txt**
- **try to get lock on main.txt in Harry's working copy**
- **commit Sally's working copy as „new changes to locked file“**
- **remove svn:needs-lock and commit as „disabling pessimistic locking“**

SVN properties

- Subversion provides interfaces for adding, modifying, and removing **versioned metadata** on each of your versioned directories and files.
- We refer to this metadata as properties, which exist as two-column tables that map property names to values attached to each item in your working copy
- The names and values of the properties can be whatever you want them to be, with the constraint that the names must contain only ASCII characters
- Properties are versioned just like the textual contents of your files. You can modify, commit, and revert property changes
- Every property-change must be committed just as the resources that exist in your working copy

SVN poperties

svn:eol-style

svn:executable

svn:externals

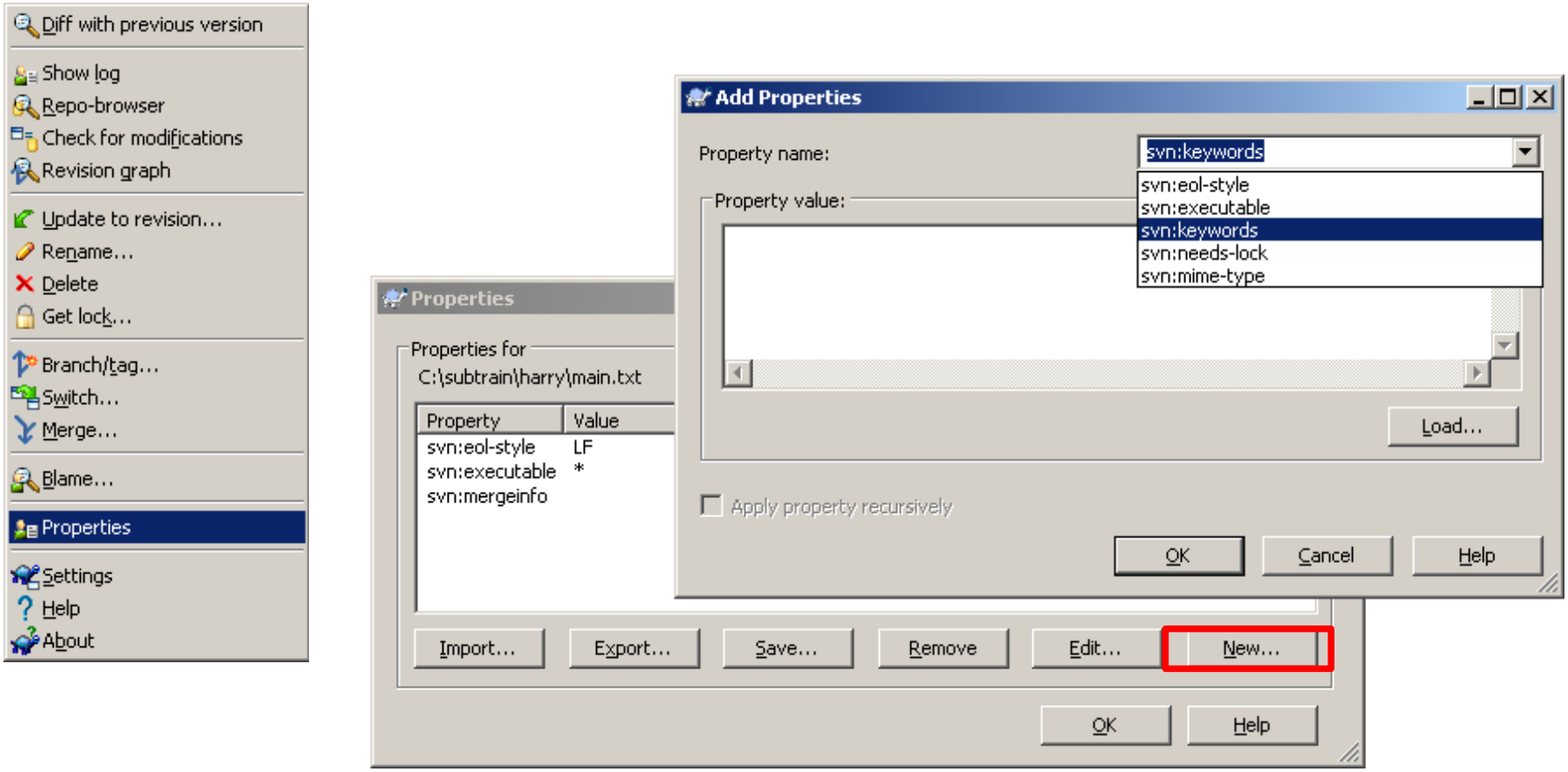
svn:ignore

svn:keywords

svn:needs-lock

svn:mime-type

svn:merge-info


Diff with previous version
Show log
Repo-browser
Check for modifications
Revision graph
Update to revision...
Rename...
Delete
Get lock...
Branch/tag...
Switch...
Merge...
Blame...
Properties
Settings
Help
About
Add Properties
Property name:
svn:keywords
svn:eol-style
svn:executable
svn:keywords
svn:needs-lock
svn:mime-type
Property value:
Load...
Apply property recursively
OK
Cancel
Help
Properties
Properties for
C:\subtrain\harry\main.txt
Property
Value
svn:eol-style
LF
svn:executable
*
svn:mergeinfo
Import...
Export...
Save...
Remove
Edit...
New...
OK
Help

- **svn:ignore**

  - Tell Subversion which files and subdirectories to ignore
  - Equivalent to CVS's .cvsignore file

  - You can define a special file/directory or patterns defining the files/directories which will be ignored.

  - You can suppress the exclusion if you use the --no-ignore flag for the svn status command.

Special SVN Properties: keywords

Creating safety. With passion.

NewTec System-Entwicklung und Beratung

- **svn:keywords**
  - Keyword substitution in files.
    The only available keyword substitutions are:
    - LastChangedDate, LastChangedRevision
    - LastChangedBy, HeadURL, Id, Header

  - Only keywords listed in the svn:keywords property value are replaced in the file

```
$LastChangedDate: 2008-07-22 21:42:37 -0700 (Sun, 22 Jul 2008) $
$LastChangedRevision: 14 $
$LastChangedBy: sally $
$HeadURL: file:///c:/subtrain/repo/trunk/main.txt $
$Id: main.txt 14 2008-07-28 21:30:43Z sally $
$Header: file:///c:/subtrain/repo/trunk/main.txt 148
     2008-07-28 21:30:43Z sally $
```

What is the format of timestamps in Subversion?

Subversion will always store timestamps in

UTC(Coordinated Universal Time)

```
$Id: calc.c 148 2002-07-28 21:30:43Z sally $
```

On Wikipedia:

The UTC time zone is sometimes denoted by the letter **Z** – a reference to the equivalent nautical time zone (GMT), which has been denoted by a **Z** since about 1950. The letter also refers to the "zone description" of zero hours, which has been used since 1920 (see time zone history). Since the NATO phonetic alphabet and amateur radio word for Z is "Zulu", UTC is sometimes known as **Zulu time**.

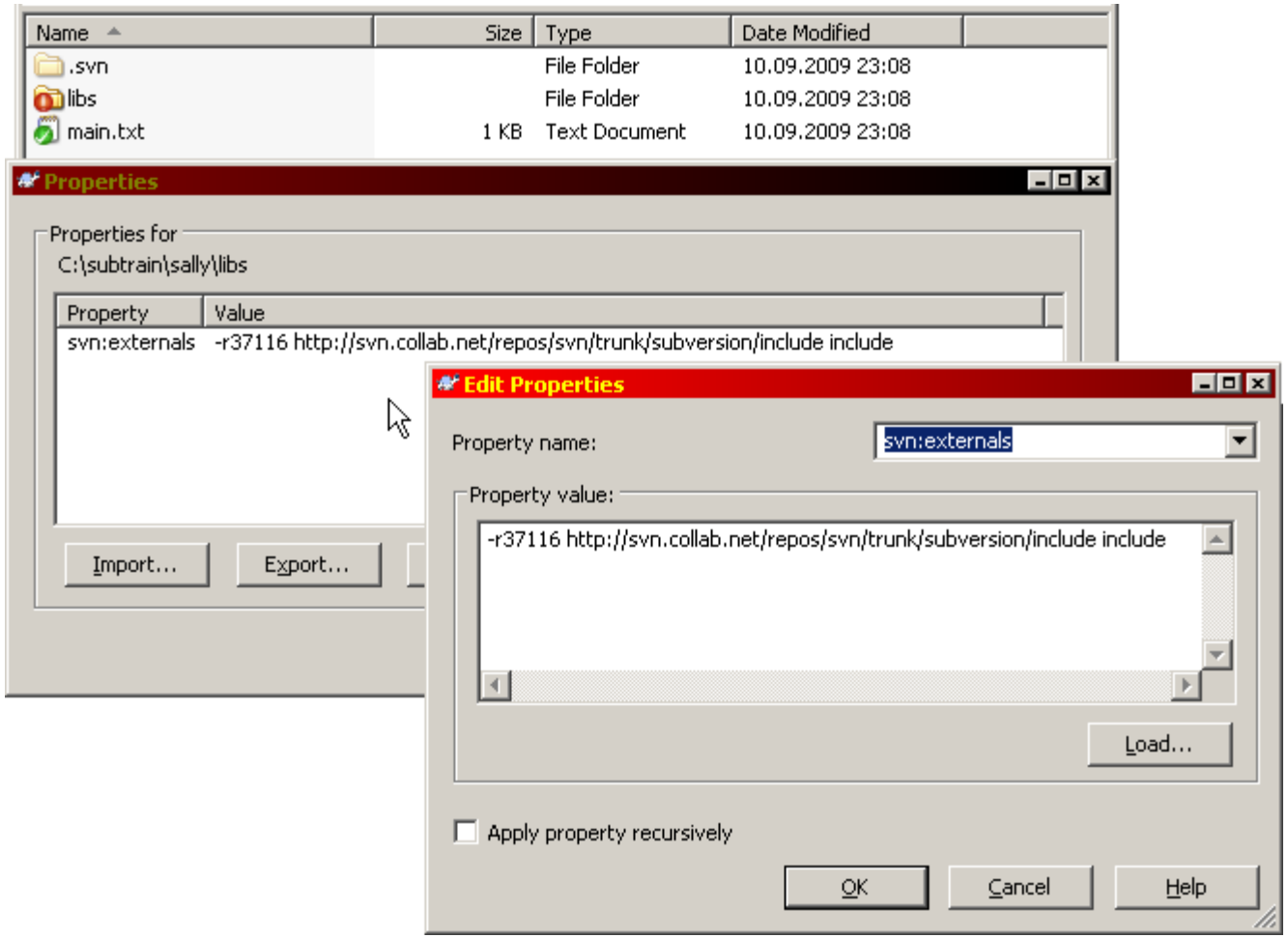
http://en.wikipedia.org/wiki/Coordinated_Universal_Time#Time_zones

- **svn:mime-type**
  - SVN assumes a text file if svn:mime-type is not set or is set and matches “text/*”, otherwise it is a binary file.
  - If a file is a text file SVN can use diff and patch to update changes from the repository.
  - Useful for setting the MIME type of a file for web browsing.
  - Set MIME type to binary if you would like to prevent SVN from automatic merging

- **svn:externals**
  - If you use a library (source form) as part of your development which is used in more than one project, you can make a kind of a link to the repository for that library. This avoids redundant copies of the library's source code.

- Advantages:
  - Just a single point of development for 3rd party library or libraries at all.

- Disadvantages:
  - No automatic commits etc. in directories which are created by svn:externals.
  - If you create a branch the externalized content will **NOT** be branched accordingly.
  - If you create a tag the externalized content will **NOT** be frozen.
    For example: you create a tag in revision 10. Six weeks later you export it to deliver it to a customer. As the externalized content was **NOT** frozen you get the latest (HEAD revision e.g. rev 24) externalized code and have therefore exported a source-code bundle which does not look the same as six weeks ago.



- **svn:eol-style**
  - Native
    - Convert all CR, CRLF and LF's to the system's native eol
  - CR
    - Convert all CRLF and LF's to CR's
  - CRLF
    - Convert all CR and LF's to CRLF's
  - LF
    - Convert all CR and CRLF's to LF's

- Property not set, this is the default when a file is added.
  Do not change end of lines upon check in or update.

Managing versions

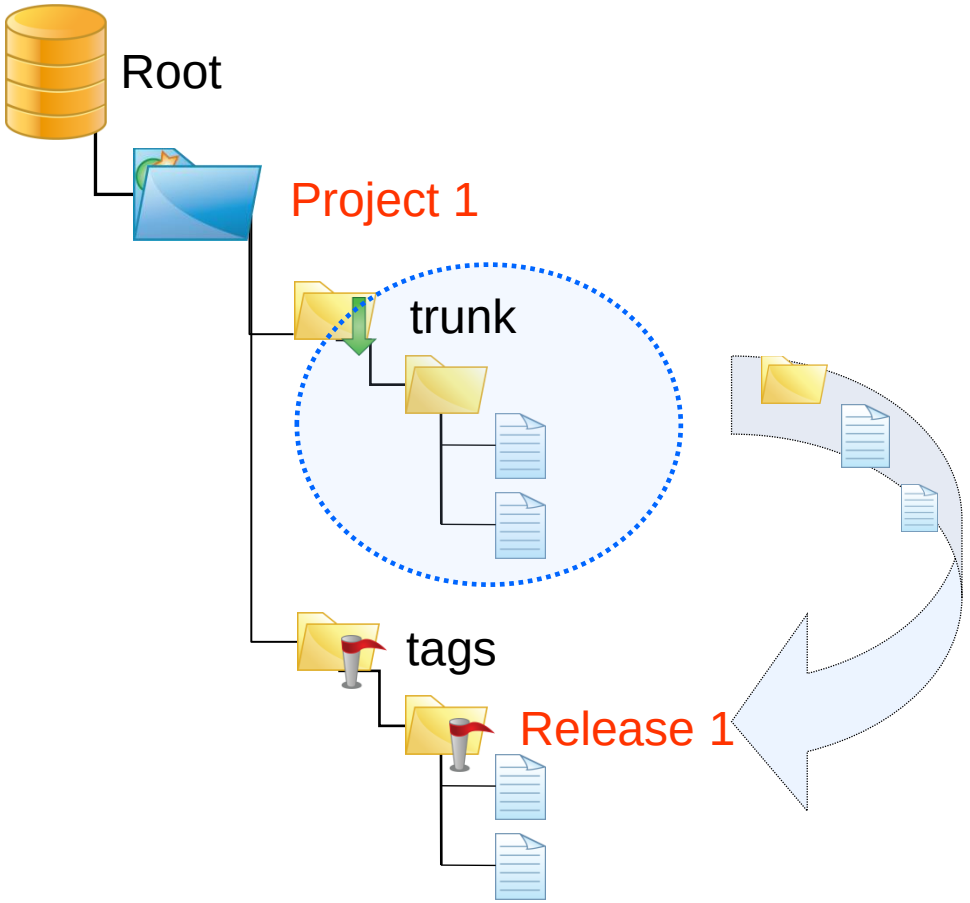
- **Why do we need tags?**
  - Mark a release state of a product.
  - Mark a snapshot of the current development.

- **Typical Release names:**
  - Release V1.0, Release V1.1, PRODUCT P1.0 etc.

- **A Tag name should be unique to mark all components of the given product (source code and documentation) and is used to reproduce the state of the tag in the future.**

- **If we take a look at the architecture chapter we really don‘t need a special tag. We only need to write down the revision number.**
- **But human beings have other requirements.**
- **They like to have self-explanatory tag names which are more handy.**

- **But the question is:**
  - Where to put the tag information in the repository?



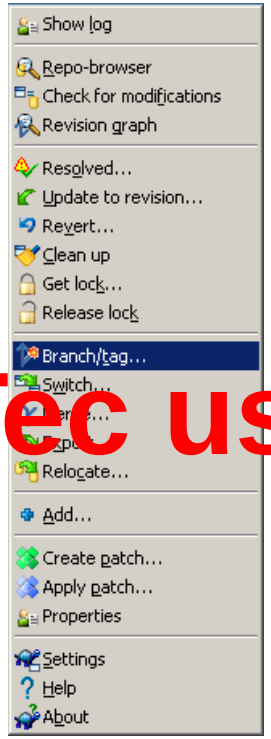
To create a release tag just copy …

…anyway you have the revision number …

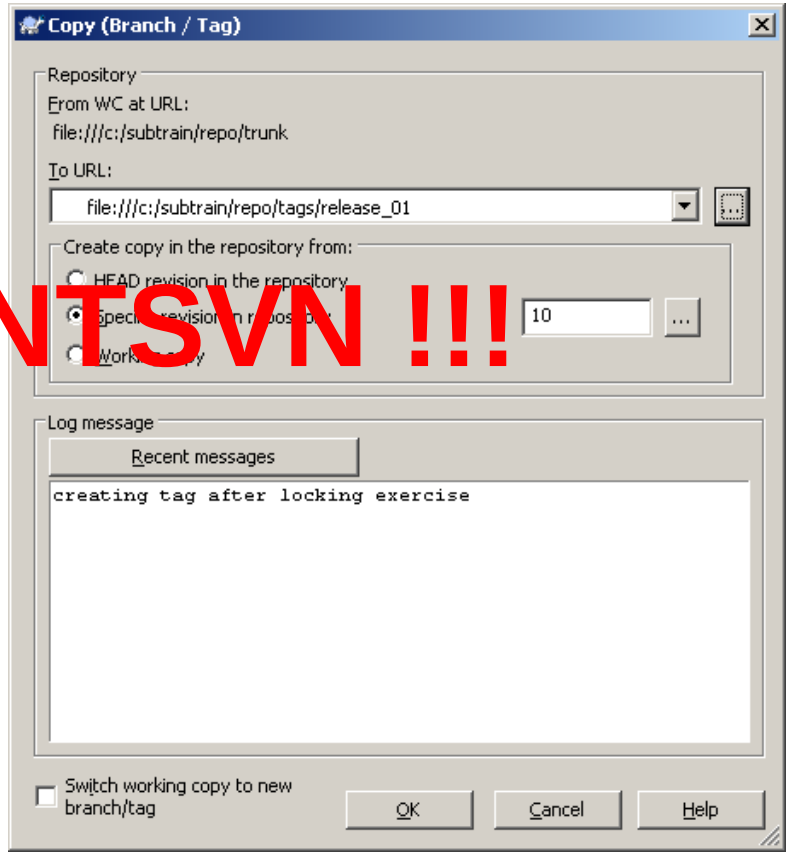
Exercise!

- create a tag named „**/tags/release_01**“ with commit message

„**creating tag after locking exercise**“



At NewTec use NTSVN !!!



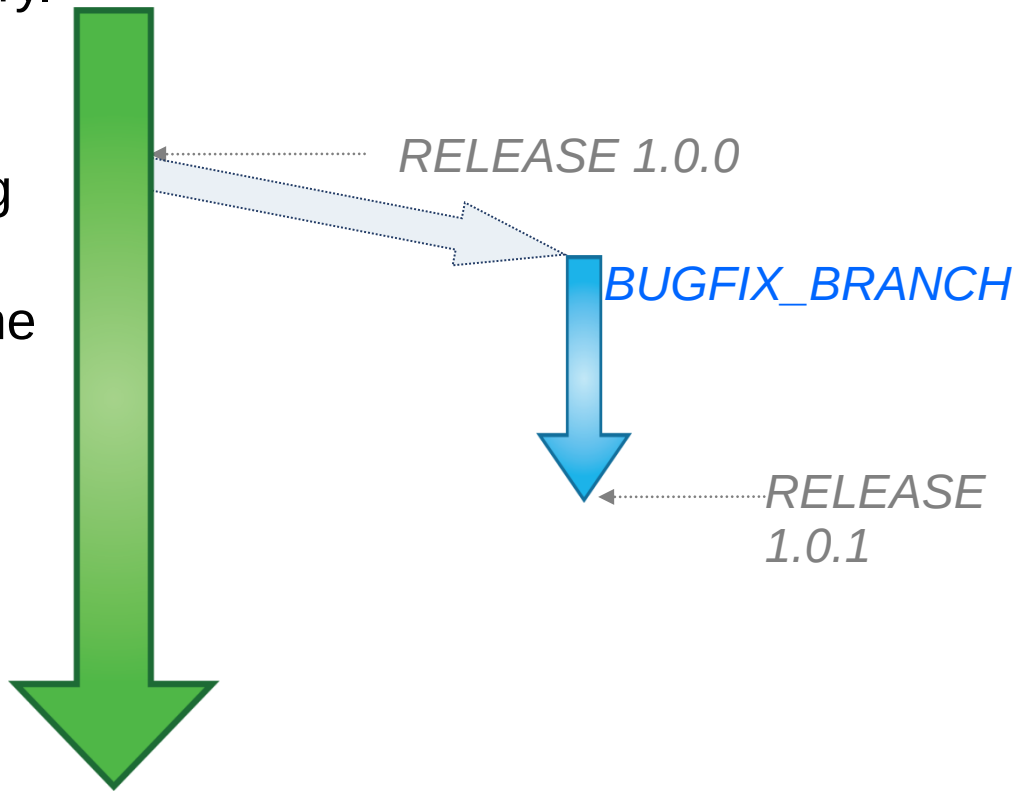

```
svn copy –m “creating release 01“
file:///c:/subtrain/repo/trunk
file:///c:/subtrain/repo/tags/release_01
```

- **Assume the following situation:**
  - You have implemented a piece of software and it has been delivered to your customer.
  - Before you delivered the product you have created a tag, let's say „Release 1.0.0“
  - Your current development crew is working on Release 1.1.0 with new features.

- **And now Murphy's Law caught you:**
  - Your customer calls you and reports that he has found a bug in your software.

- The development has continued after the release of RELEASE 1.0.0
- You want to fix the bug to satisfy your customer!
- In your current development you have enhanced many of the product's functions but you don't want to deliver a product with more features and you haven't finished testing yet.
- How to solve this situation?

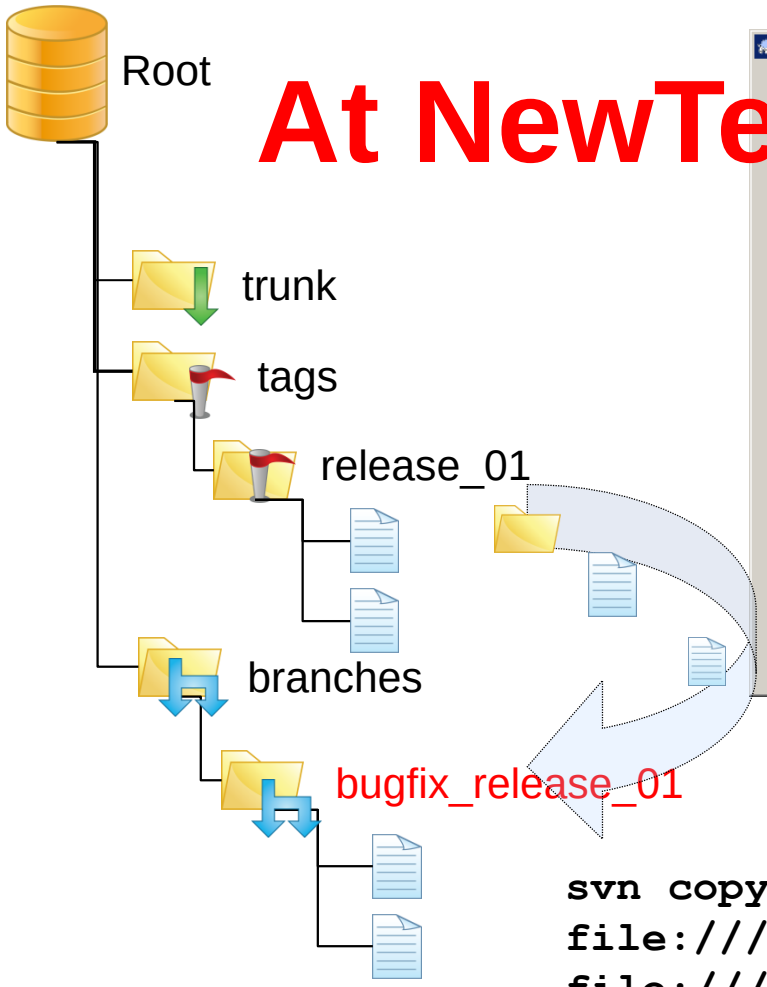
- Based on the tag you‘ve created during the delivery you can check out the exact state of the delivery.
- You create a **Branch** to fix the bug in the software.
- After you have fixed the bug you can tag the Branch and deliver another version to the customer.
- Your customer is satisfied that you fixed the bug so fast.
- You haven‘t disturbed the current development.



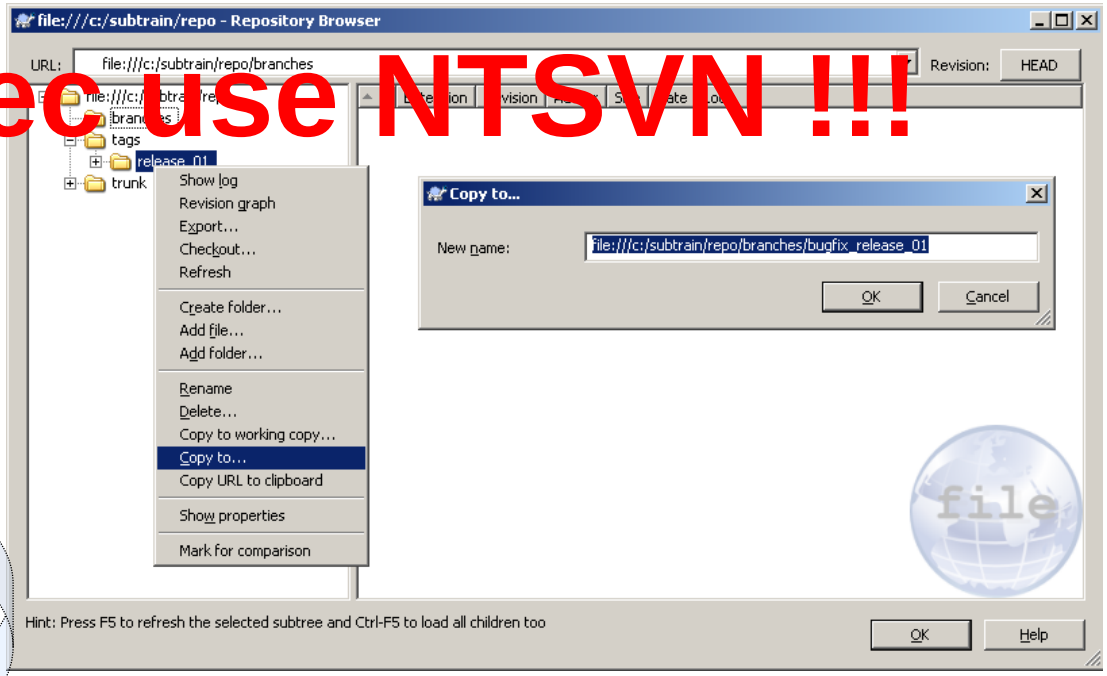
Branches - Creating branches by copy in repository

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung



At NewTec use NTSVN !!!

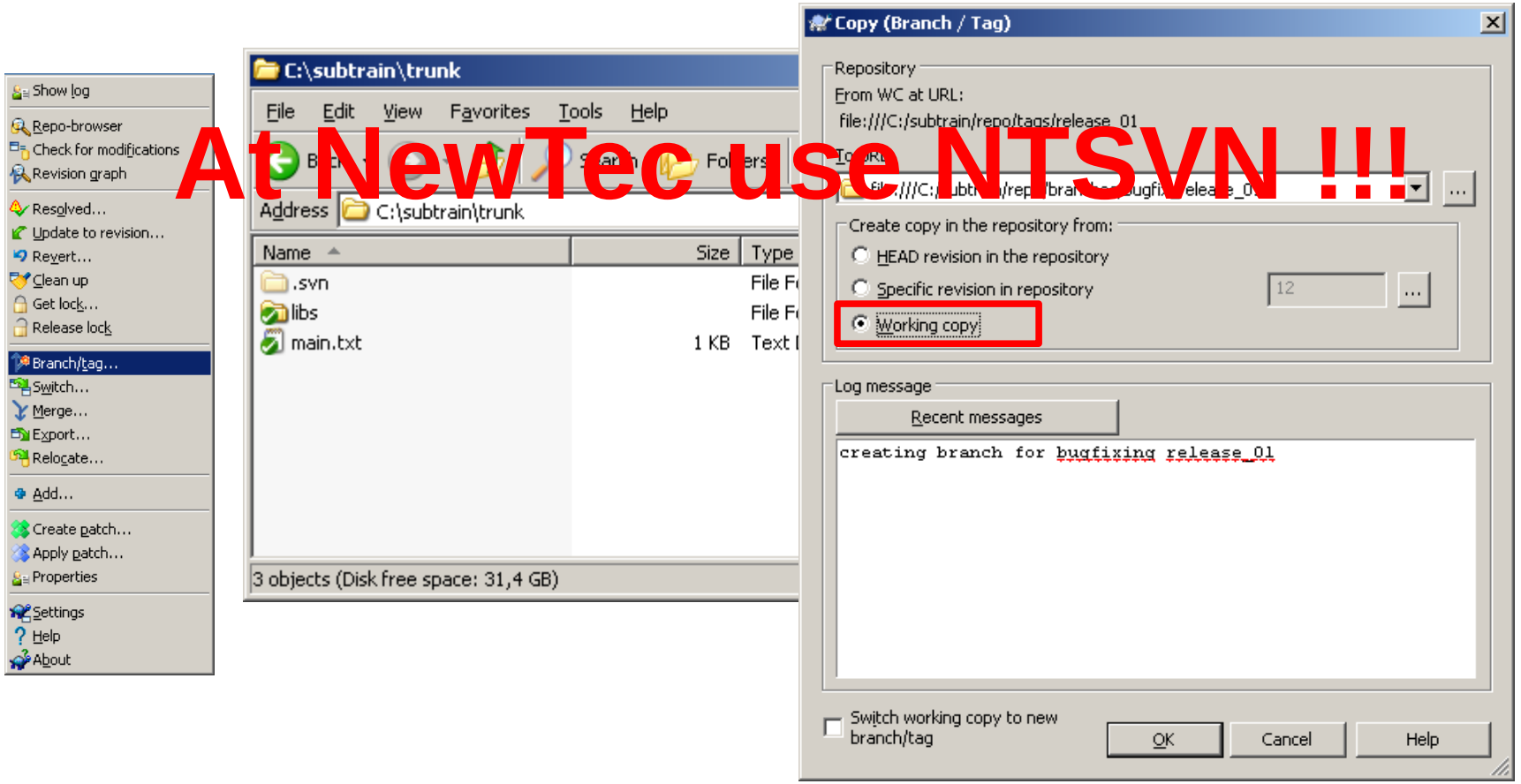



```
svn copy –m "creating branch for bugfixing.."
file:///c:/subtrain/repo/tags/release_01
file:///c:/subtrain/repo/branches/bugfix_release_01
```

Branches - Creating branches from your working copy

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung



At NewTec use NTSVN !!!

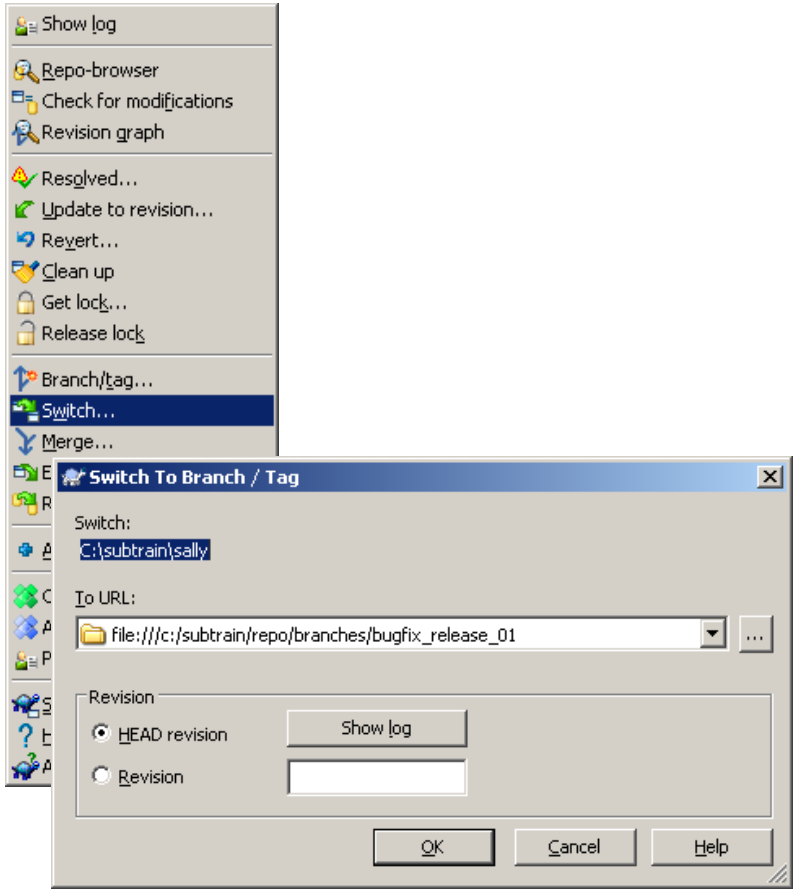
```
svn copy
c:\subtrain\harry\
file:///c:/subtrain/repo/branches/bugfix_release_01
```

- If you want to work on a different branch, you can „switch“ your working copy instead of a new checkout.

```
svn switch url path
```

1 2 3

1) The Subversion “switch”-command.
2) The url where your working copy should point to.
3) The path of your working copy.




```
svn switch
file:///c:/subtrain/repo/branches\bugfix_release_01
c:\subtrain\sally
```

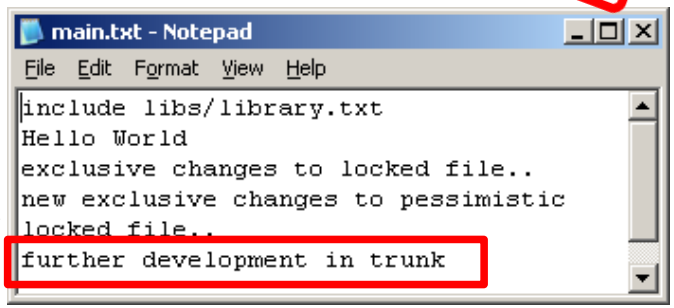
Exercise!

- create a branch named „**`/branches/bugfix_release_01`**“ with commit message

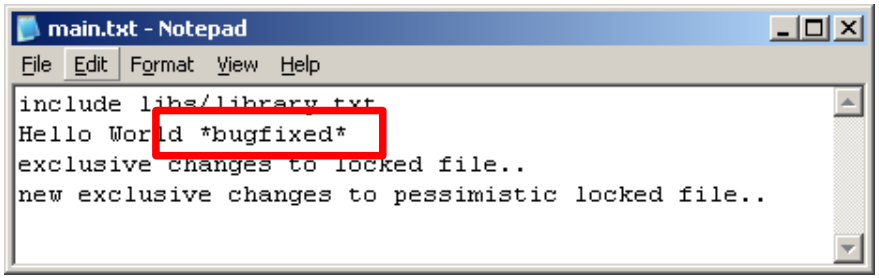
  „**`creating branch for bugfixing`**“

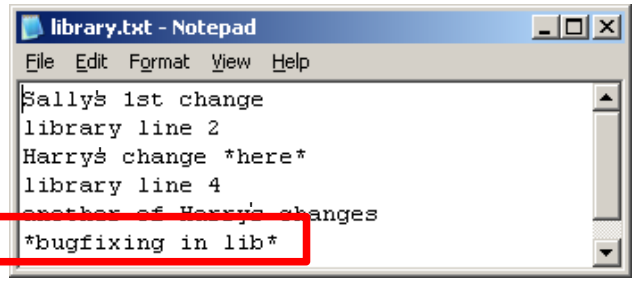
- Switch Sally's working copy to this new branch

- add a line in Harry's main.txt and commit it as „making progress in trunk“



- change Sally's main.txt and libs/library.txt:

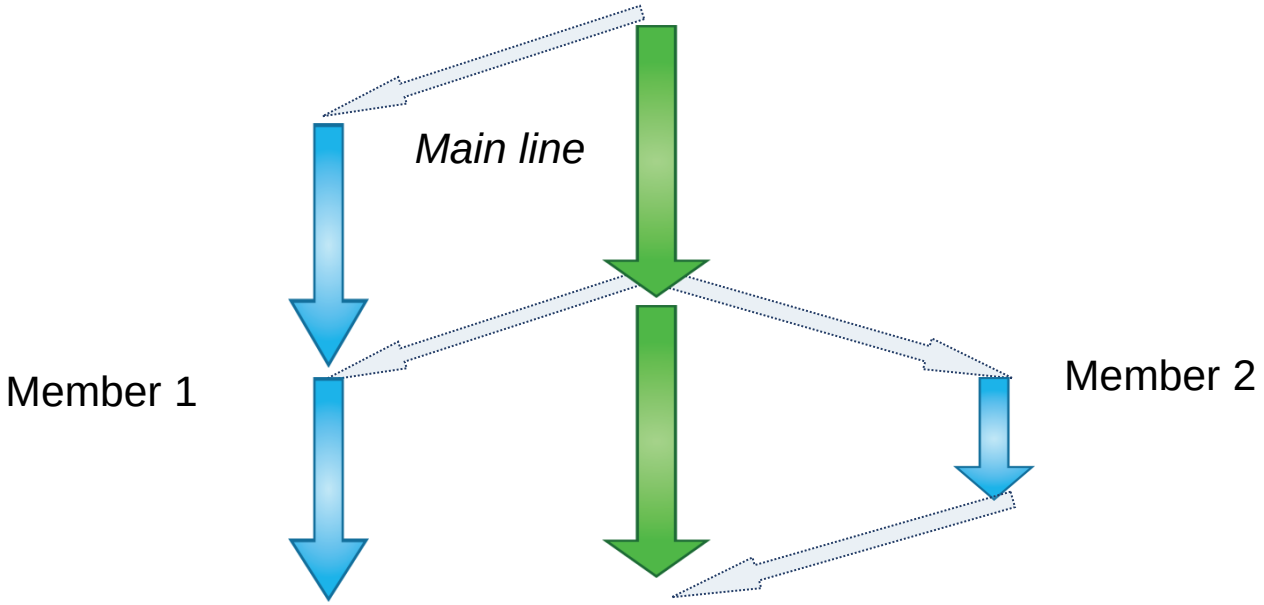




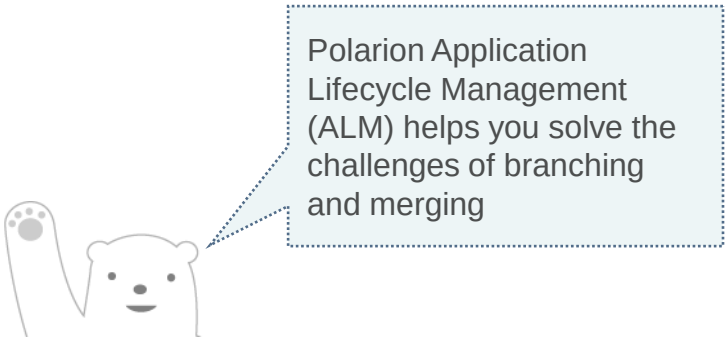
- commit Sally's changes as „bugfixing on branch“

Branching strategies

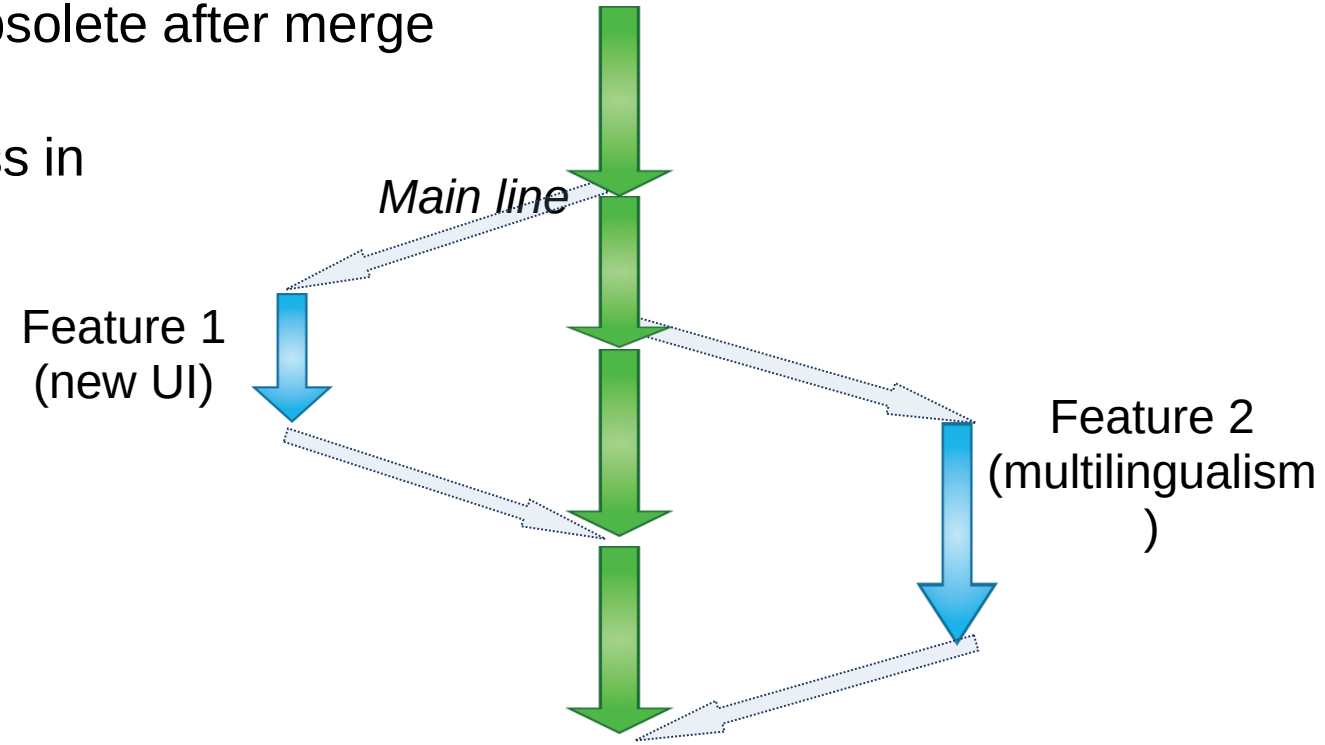
- Separation of team members can be realized with branches.
- One branch per team member or several members on a branch - the decision is based on the size of the teams.



- **Advantages using branches for team work:**
  - No changes during development on the main line needed => Code stability.
  - Every team member can work in their own environment.
- **Disadvantages:**
  - Sometimes the mainline and the branch will diverge if the branch lives too long.
  - large merge effort for maintainer of trunk

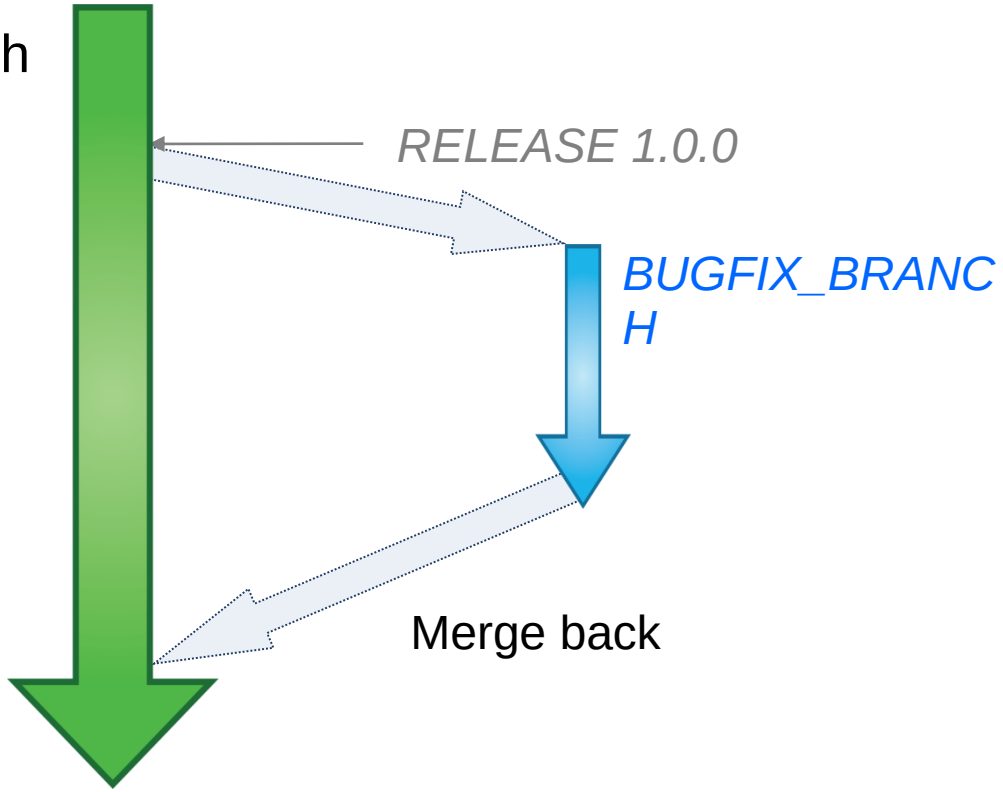


- Separation by features (one branch each).
- Branch will be obsolete after merge
- trunk will progress in atomic features



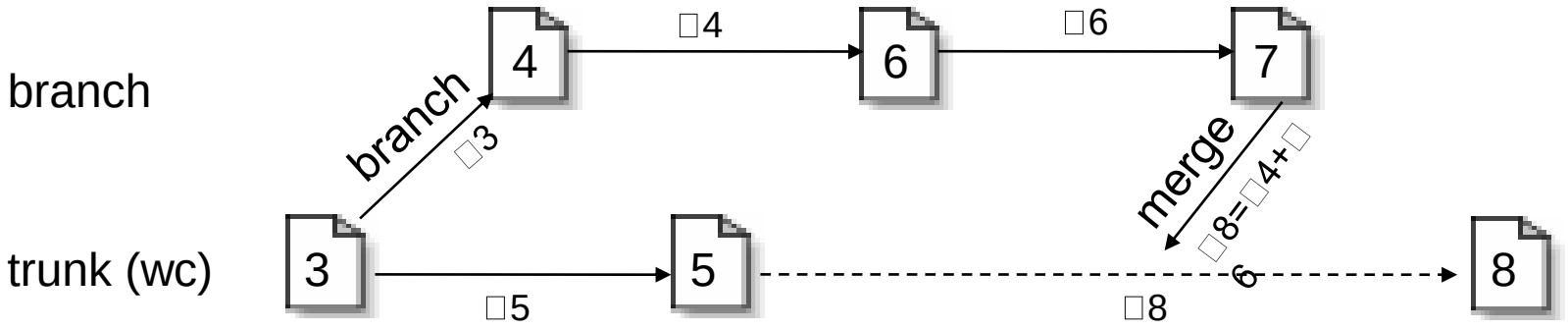
Merging

- What’s with the bug you've fixed on the bug-fix-branch?
- What about your current development?
- You have to merge the changes made in the branch back to the main line.



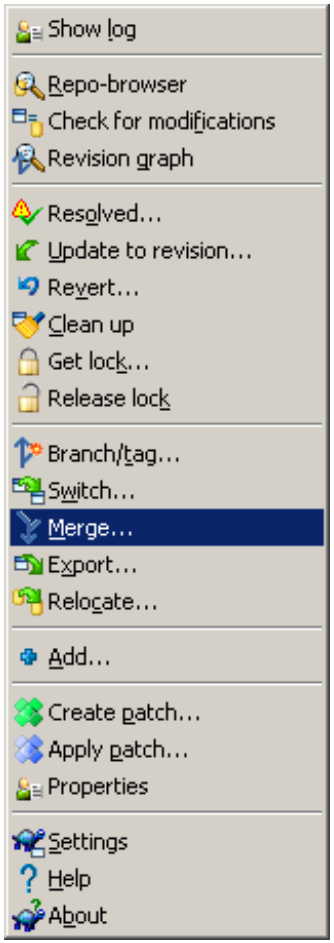
Merging – Merging means: applying a „patch“ to your wc

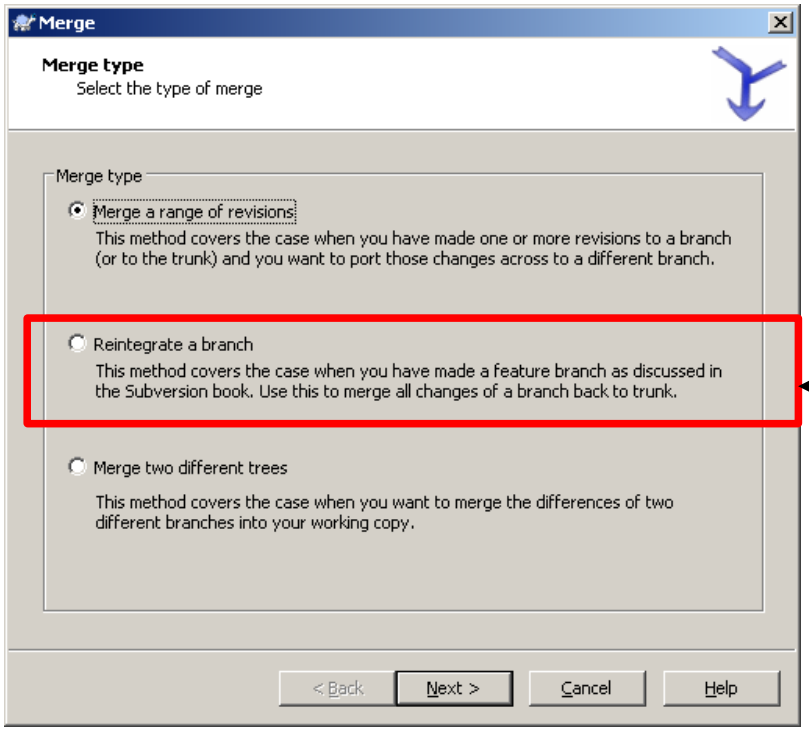
- a merge will always apply a “patch” to a working copy
- a patch is always a difference between two versions/trees
- in this sample below we create the patch between revision 4 and 7



- If you need to create a branch, you should do it from a completely committed working copy. This prevents you from becoming confused.

- If you merge check out a clean copy into another directory.
  - Otherwise you can't go back using “svn revert”.
  - After you've merged commit the changes and provide a log message with information on which revision/branch you have merged (merge tracking).
  - You can first test the merge using the --dry-run flag of the merge command**.**





reintegrate branch

- Switch Sally's working copy back to /trunk
- merge the appropriate revision(s) from /branches/bugfix_release_01/ into Sally's working copy
- check if the merge add the bugfix into the trunk
- commit Sally's changes as:

„merged bugfix from branches/bugfix_release_01 rev13:14 into trunk“

- as this you have some kind of human readable merge tracking
- control the merge tracking with TortoiseSVN‘s „show log“

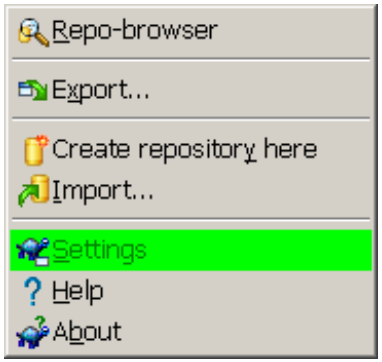
Subversion Configuration

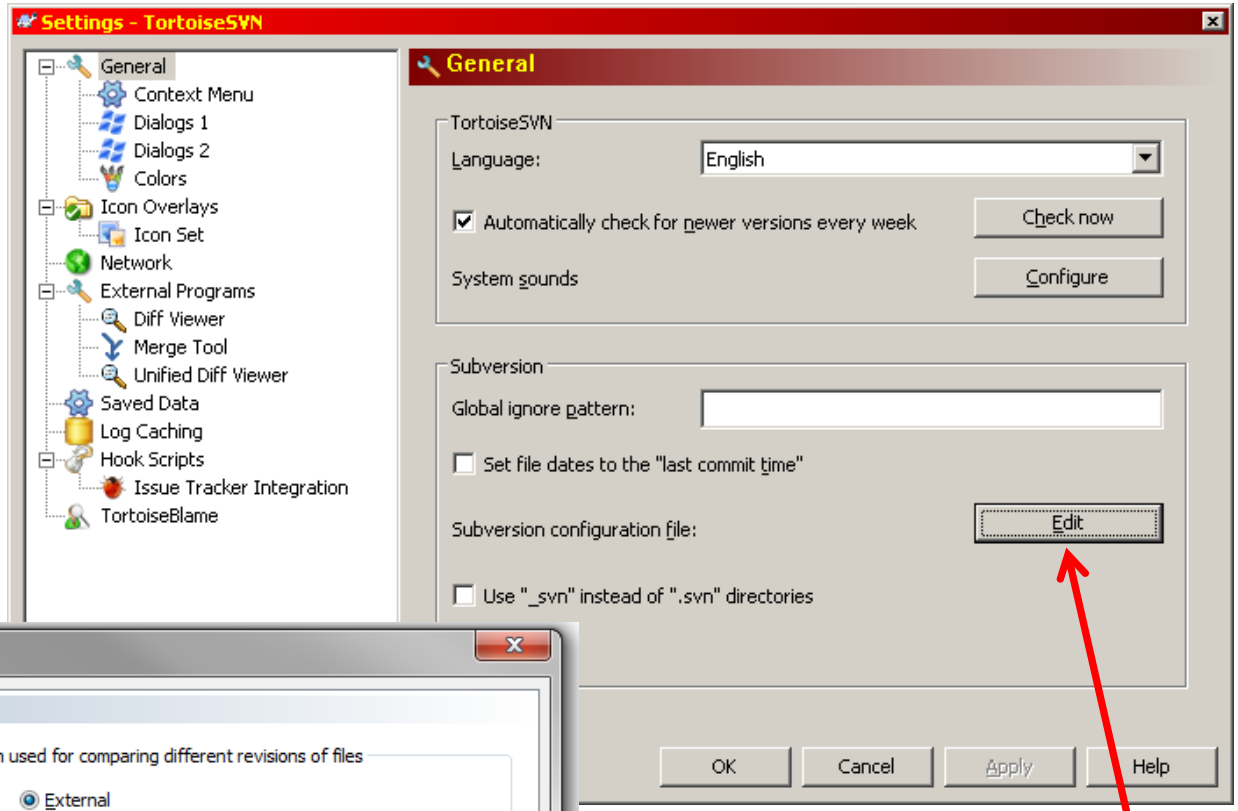
- **Under Linux you will find the configuration under:**
  - /home/**Username**/.subversion/config
- **Under Windows you will find the configuration under:**
  - %APPDATA%/subversion/
  - In the config file [**`config.txt`**] you can change the behaviour of parts of Subversion.
  - in the server configuration file [**`servers`**] you can setup proxy-server settings and ssl authorities

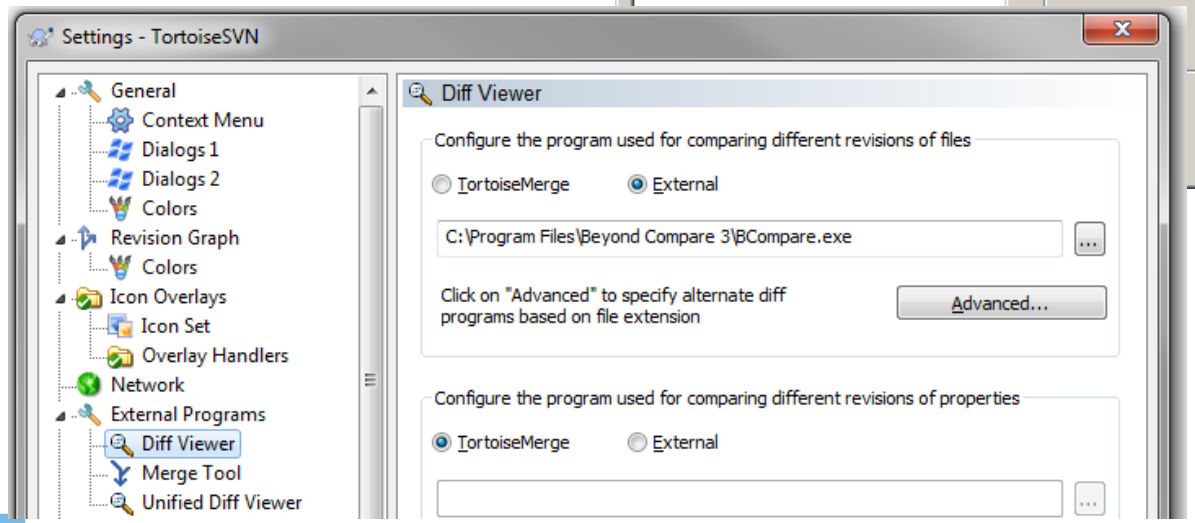
Subversion Configuration

Creating safety.
With passion.

NewTec
System-Entwicklung und Beratung







Edit configuration and replace with content from:
http://svn/nt/ntsvn/trunk/client/system/config/config

- **Auto-properties can be useful if you like to set Word/Excel files to read-only.**
  - You have to turn on enable-auto-props which means set it to „yes“.
  - After that you can set svn:needs-lock for particular file types.

Subversion Configuration auto-properties

```
[auto-props]
### The format of the entries is:
###   file-name-pattern = propname[=value][;propname[=value]...]
### The file-name-pattern can contain wildcards (such as '*' and
### '?').  All entries which match will be applied to the file.
### Note that auto-props functionality must be enabled, which
### is typically done by setting the 'enable-auto-props' option.
# *.c = svn:eol-style=native
# *.cpp = svn:eol-style=native
# *.h = svn:eol-style=native
# *.dsp = svn:eol-style=CRLF
# *.dsw = svn:eol-style=CRLF
# *.sh = svn:eol-style=native;svn:executable
# *.txt = svn:eol-style=native
# *.png = svn:mime-type=image/png
# *.jpg = svn:mime-type=image/jpeg
# Makefile = svn:eol-style=native
```

You can put whatever file extension into global-ignores to globally ignore files of a particular type.

```
[miscellany]
### Set global-ignores to a set of whitespace-delimited globs
### which Subversion will ignore in its 'status' output.
# global-ignores = *.o *.lo *.la #*# .*.rej *.rej .*~ *~ .#* .DS_Store
```

- Sometimes you need to have the changed time to be set on the commit-time instead of the created local time. This is often used in relationship with Make-files.
- You can simply uncomment the line with “use-commit-times = yes“.
- After that every check out will set the time stamp to commit time and not to the check out time.

```
### Set use-commit-times to make check out/update/switch/revert
### put last-committed timestamps on every file touched.
# use-commit-times = yes
```

Additional Links and SVN info sources

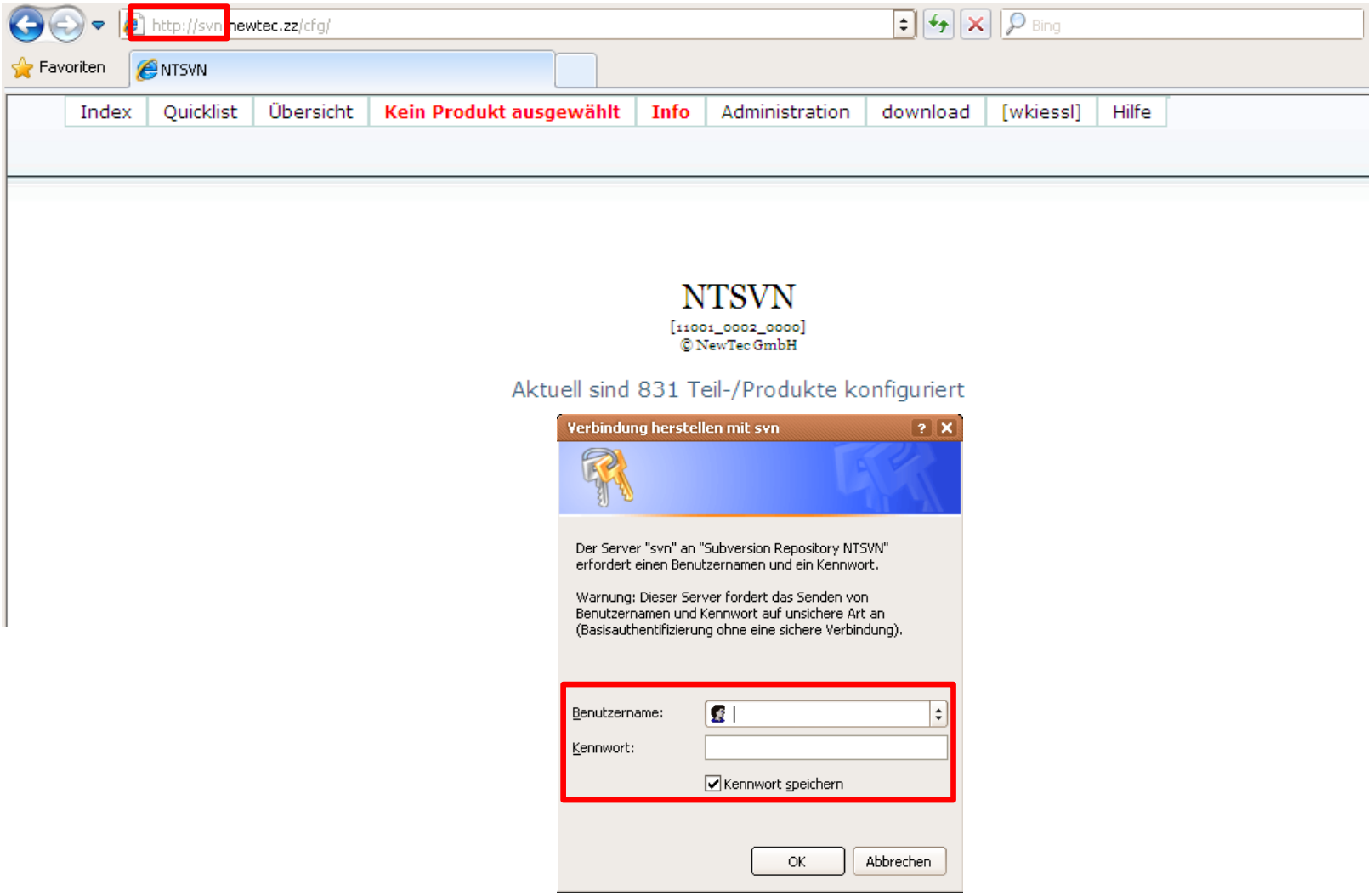
- **Subtrain online:**
  - http://subtrain.tigris.org
- **Book: Version Control with Subversion:**
  - http://svnbook.red-bean.com/
- **Subversion Developer Portal and downloads:**
  - http://subversion.apache.org/
- **Subversion Wiki:**
  - http://www.subversionary.org
- **Subversion Free Tools:**
  - http://www.polarion.org
- **Subversion Forums:**
  - http://www.svnforum.org/
- **Subversion News**
  - http://svn.haxx.se/

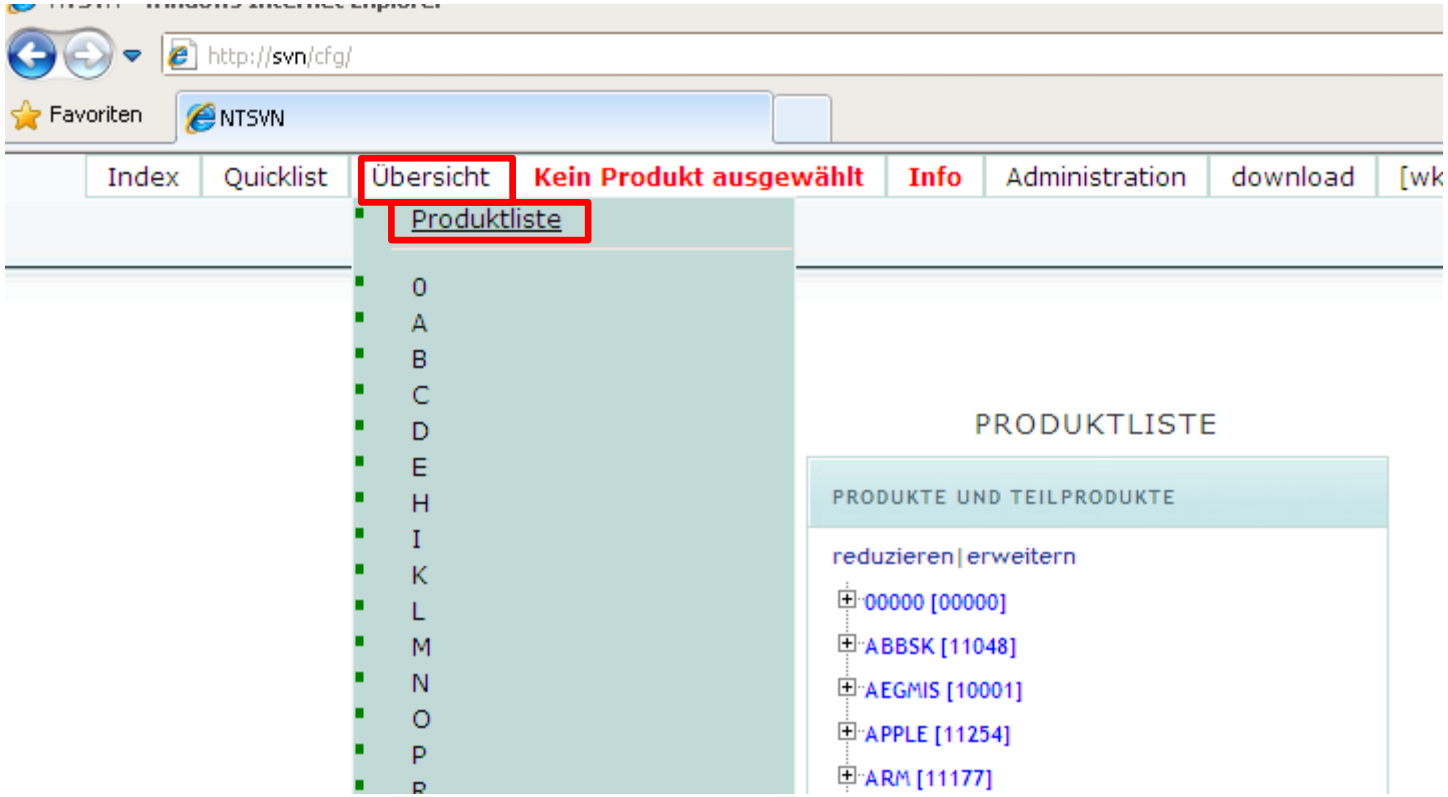
NTSVN

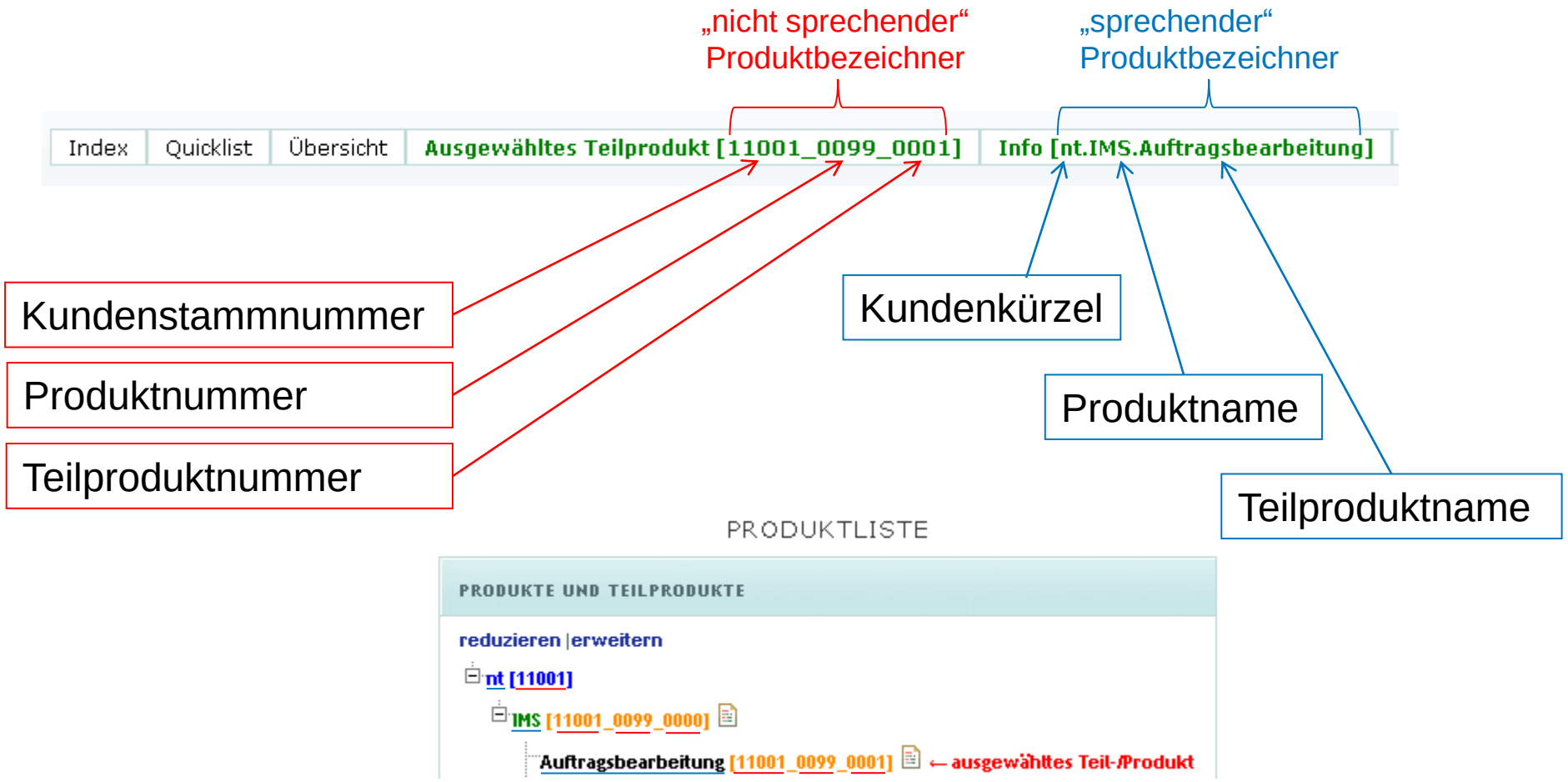
NTSVN stellt unser Konfigurationsmanagmentsystem dar.

Es unterteilt sich

- in das Konfigurationssystem Subversion und
- in die Weboberfläche NTSVN, welches die Verwaltung übernimmt und den
- SVN Client „TortoiseSVN“ .


http://svn.newtec.zz/cfg/
Bing
Favoriten
NTSVN
Index
Quicklist
Übersicht
Kein Produkt ausgewählt
Info
Administration
download
[wkiessl]
Hilfe
NTSVN
[11001_0002_0000]
© NewTec GmbH
Aktuell sind 831 Teil-/Produkte konfiguriert
Verbindung herstellen mit svn
Der Server "svn" an "Subversion Repository NTSVN" erfordert einen Benutzernamen und ein Kennwort.
Warnung: Dieser Server fordert das Senden von Benutzernamen und Kennwort auf unsichere Art an (Basisauthentifizierung ohne eine sichere Verbindung).
Benutzername:
Kennwort:
Kennwort speichern
OK
Abbrechen

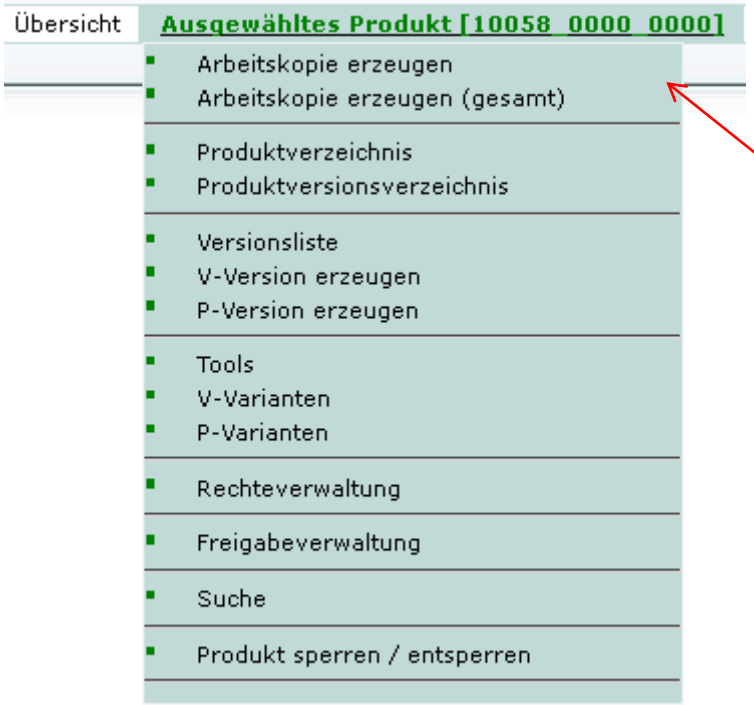

http://svn/cfg/
Favoriten
NTSVN
Index
Quicklist
Übersicht
Kein Produkt ausgewählt
Info
Administration
download
Produktliste
0
A
B
C
D
E
H
I
K
L
M
N
O
P
PRODUKTLISTE
PRODUKTE UND TEILPRODUKTE
reduzieren | erweitern
00000 [00000]
ABBSK [11048]
AEGMIS [10001]
APPLE [11254]
ARM [11177]


„nicht sprechender“
Produktbezeichner
„sprechender“
Produktbezeichner
Index
Quicklist
Übersicht
Ausgewähltes Teilprodukt [11001_0099_0001]
Info [nt.IMS.Auftragsbearbeitung]
Kundenstammnummer
Produktnummer
Teilproduktnummer
Kundenkürzel
Produktname
Teilproduktname
PRODUKTLISTE
PRODUKTE UND TEILPRODUKTE
reduzieren | erweitern
nt [11001]
IMS [11001_0099_0000]
Auftragsbearbeitung [11001_0099_0001]
← ausgewähltes Teil-/Produkt

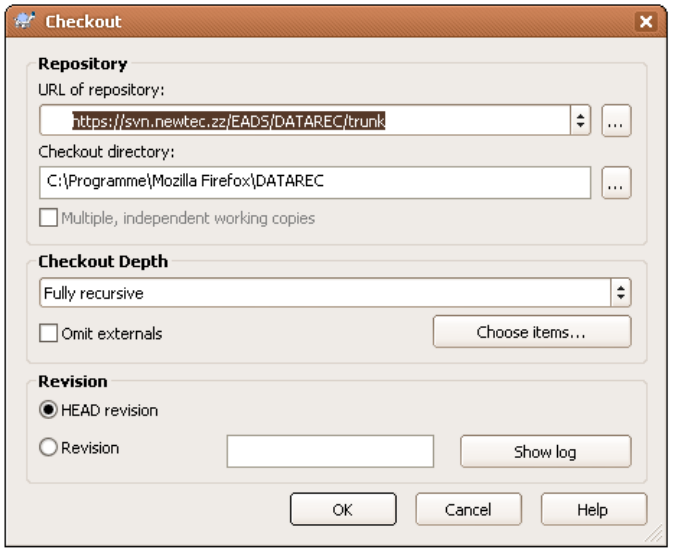
Alle weiteren Aktionen sind dann durch das aktualisierte Menü möglich

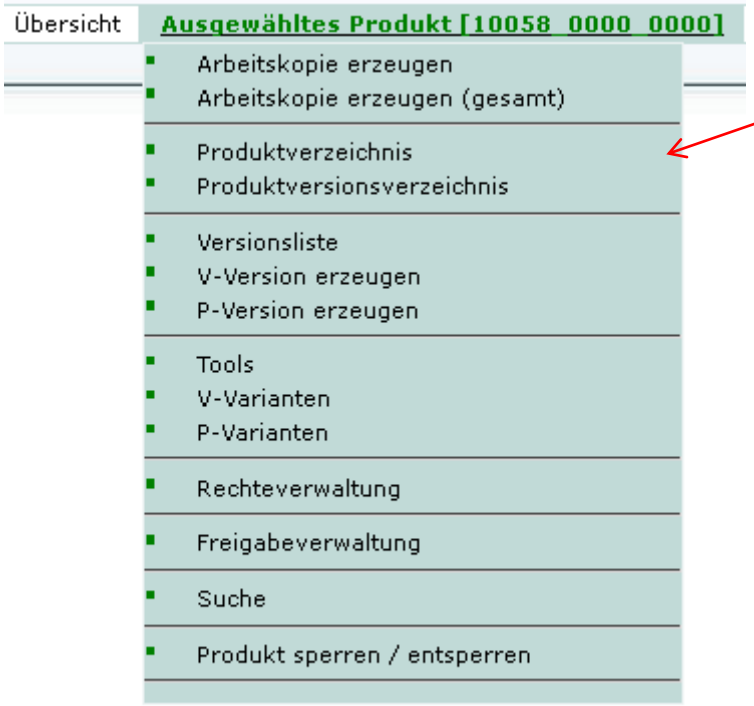
Teil-/Produkt kann durch Klick ausgewählt werden

NTSVN
Arbeitskopie erzeugen



Den aktuellen Produktstand des Produktes als neue Arbeitskopie auschecken.
(gesamt) bedeutet hier, dass alle Teilprodukte ebenfalls mit ausgecheckt werden





Link in das Produkt bzw.
Produktversionsverzeichnis

Aktuelle Revision: 4824

/trunk/system

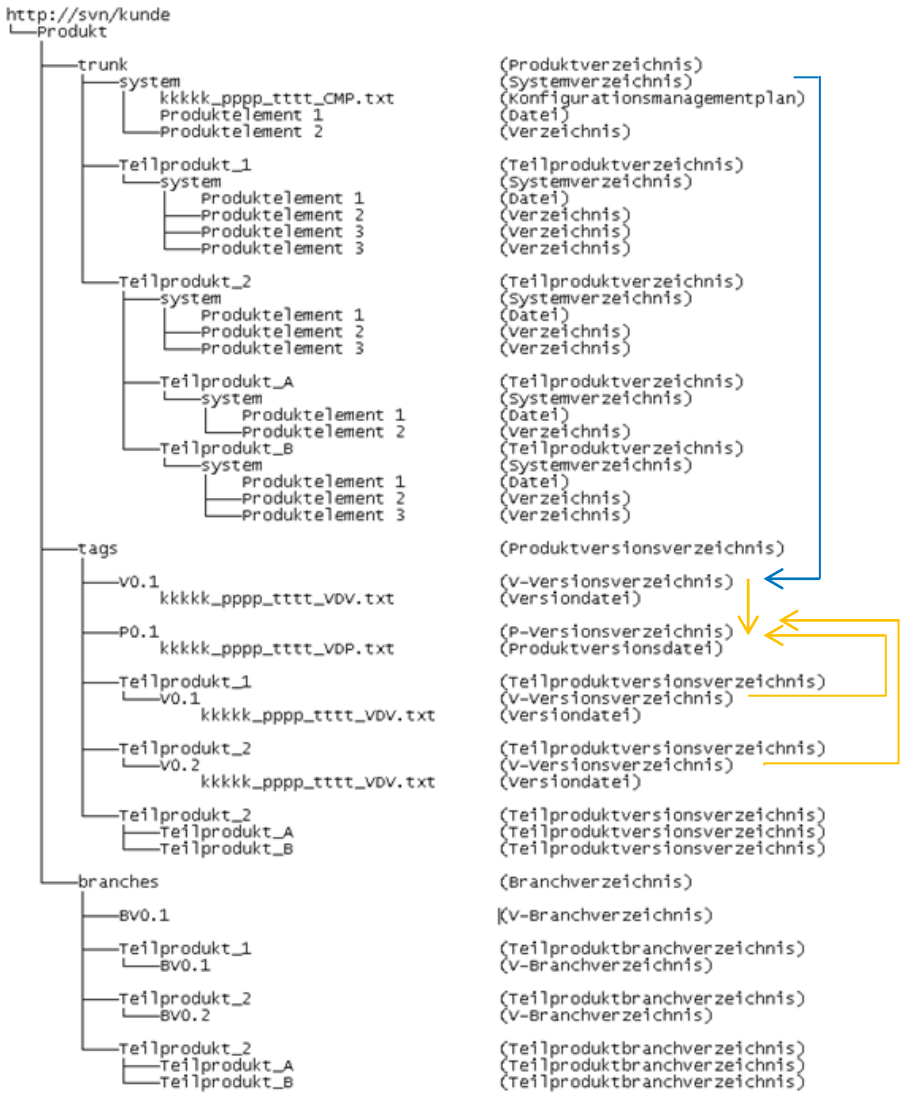
- [Parent Directory]
- Design/
- Extern/
- Implementation/
- Integration/
- Planning/
- Requirements/
- Reviews/
- Testing/
- 10058_0000_0000_CMP.txt
- 10058_0000_0000_Pruefprotokoll.pdf
- MakeCD.bat
- xcopy.exe

Zeigt alle bereits erzeugten Versionen an

VERSIONSLISTE

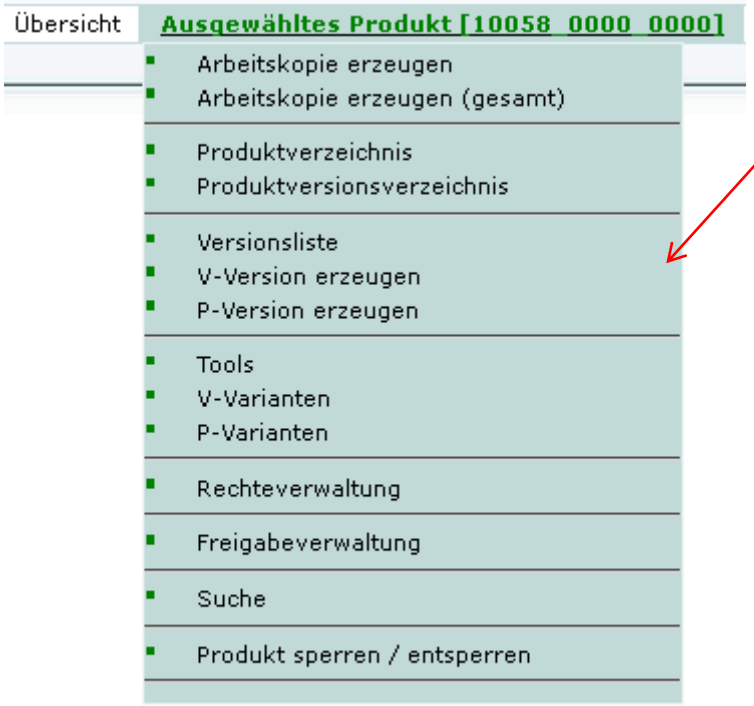
VERSIONEN	DATUM	BESCHREIBUNG	EXTERNALS
P2.0	2013-02-19	Stand zur Abnahme Phase C	"../V2.0" "system" "../Bedien-SW/P1.4" "Bedien-SW/" "../Frontend/P1.3" "Frontend/" "../Recorder/P2.0" "Recorder/" "../Studie/P1.0" "Studie/"
V2.0	2013-02-19	Stand Abnahme Phase C	ø
P1.7	2013-01-16	Stand mit funktionierendem FE_Replay (FPGA A 150MHz Speichertakt)	"../Bedien-SW/GUI/P1.4" "Bedien-SW/GUI/" "../Bedien-SW/simDataGen/V3.1" "Bedien-SW/simDataGen/system" "../Frontend/Logik/V1.2" "Frontend/Logik/system" "../Recorder/API/V1.10" "Recorder/API/system" "../Recorder/FPGA/V1.12" "Recorder/FPGA/system" "../Recorder/SW/P1.13" "Recorder/SW/" "../Recorder/Sync-Board/V1.1" "Recorder/Sync-Board/system"

NTSVN
Unterschied zwischen V und P-Version

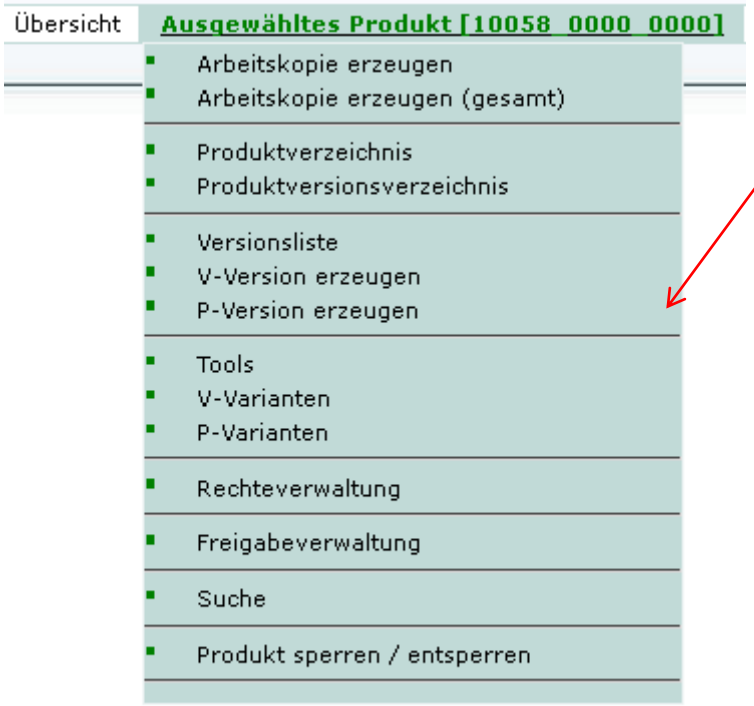


Eine V-Version ist eine umbenannte Kopie eines Systemverzeichnisses in das V-Versionsverzeichnis

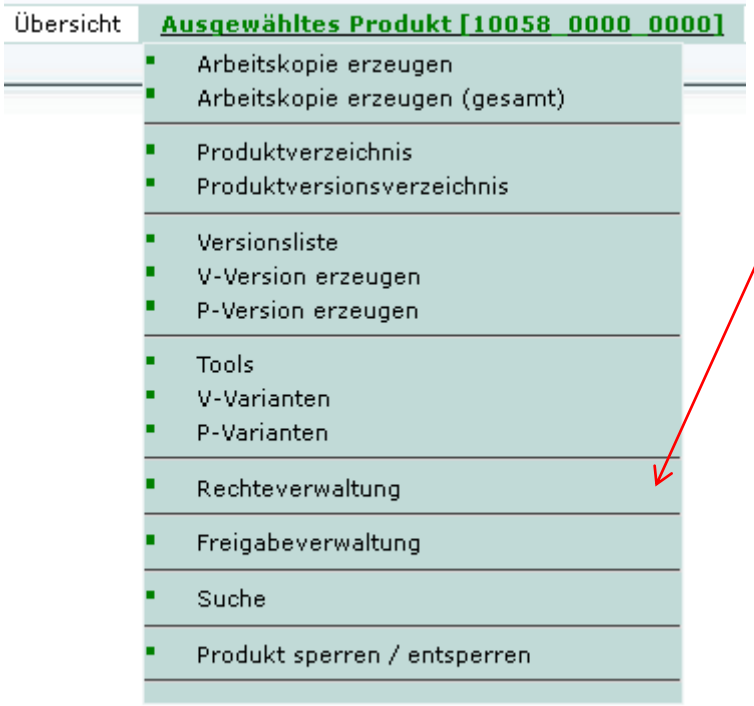
Eine P-Version ist eine Gruppierung von V- und P-Versionen



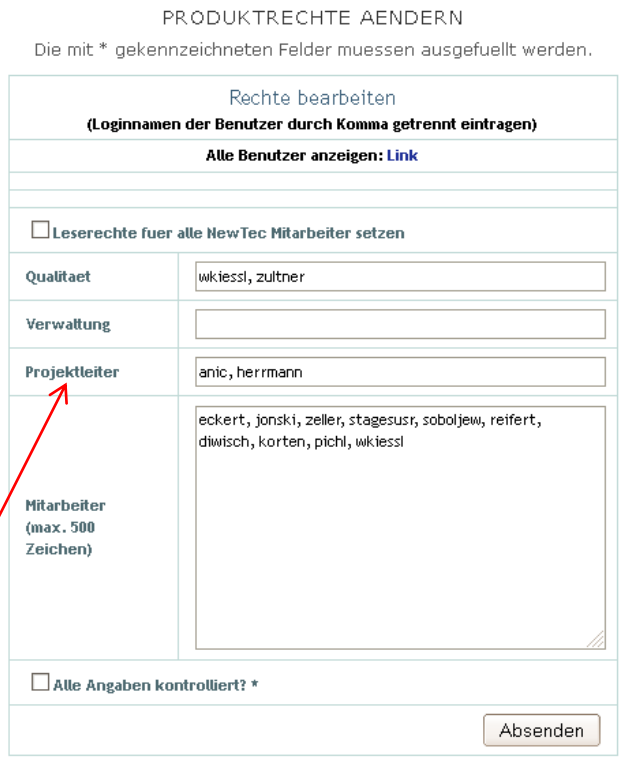
Erzeugt eine neue V-Version



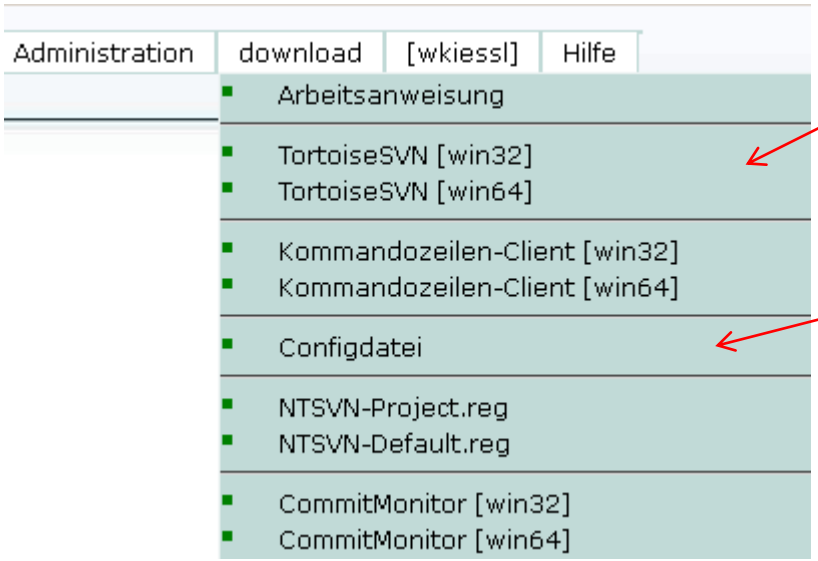
Erzeugt eine neue P-Version



Zugriffsrechte einstellen

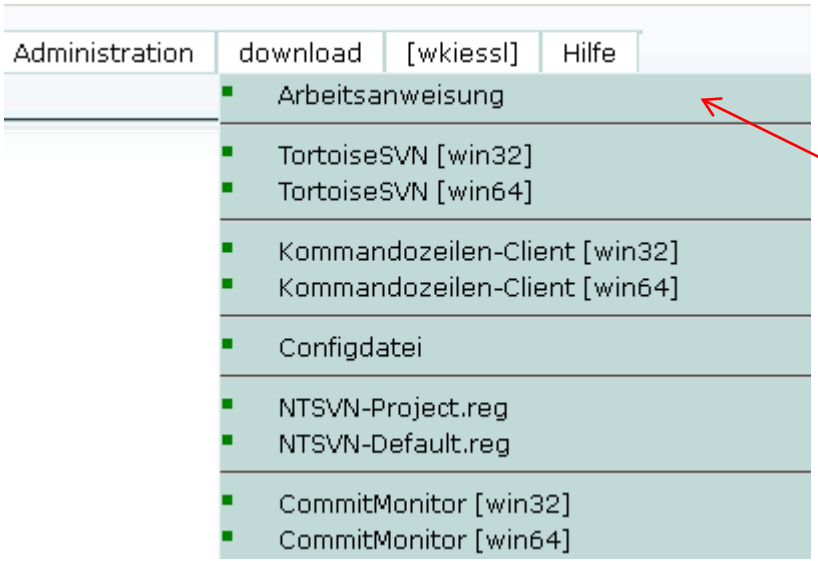


Alle eingetragenen Projektleiter haben das Recht Mitarbeiter hinzuzufügen bzw. zu entfernen



Den aktuell freigegebenen Client downloaden und installieren

NewTec spezifische Configdatei downloaden und im Verzeichnis **%appdata%\subversion** mit dem Namen **config** speichern



Download der aktuellen Arbeitsanweisung

Da diese Arbeitseinweisung ebenfalls einer Lenkung unterliegt, wird man auf Stages weitergeleitet. Nach einem erfolgreichen Login kann die Arbeitsanweisung heruntergeladen werden.

Any Questions?

SVN Command reference

add
blame (praise, annotate, ann)
cat
changelist (cl)
checkout (co)
cleanup
commit (ci)
copy (cp)
delete (del, remove, rm)
diff (di)
export
help (?, h)
import
info
list (ls)
lock
log
merge
mergeinfo
mkdir
move (mv, rename, ren)
propdel (pdel, pd)
propedit (pedit, pe)
propget (pget, pg)
proplist (plist, pl)
propset (pset, ps)
resolve
resolved
revert
status (stat, st)
switch (sw)
unlock
update (up)