Subversion 1.9 for Developers

Essential Concepts 2

Outline

- Branches & tags
- Merging changes
  - Use cases
  - Best practices
- Tracking merges
  - What is merge tracking
  - Tree conflicts and detection
- Other useful features
  - Automatic property setting
  - Pulling in external data

Branches & tags

Branches & tags

Branching and tagging in Subversion:

- No special concept (i.e., not implemented as meta data).
  - There is only the data set and copies there-of.
  - A copy is a tag or branch merely because you, as the user, attach that connotation.
- Extremely cheap (in space and execution time) as data stored consists of:
  - Space (source directory pointer – i.e., no copies of the objects).
  - Time (global revision number).
- Simple, powerful and flexible.
  - You can tag a branch and branch a tag.
- Versioned
  - Commits can be audited.
  - Nothing is historically deleted so mistakes can be easily detected and reverted.

Repository layout

- Repository layout best practice is to keep branches and tags at a single level.

```
/                          vs.      /
branches/                           branches/
tags/                                    component1-branch/
trunk/                                   component4-branch/
                                         project-branch/
                                   tags/
                                   trunk/
```

  - Subversion will handle multiple levels but humans lose track.

- Repository can contain multiple projects utilizing this layout.

```
/
/Project A
         branches/
         tags/
         trunk/
/Project B
         branches/
         tags/
         trunk/
```

What is a tag?

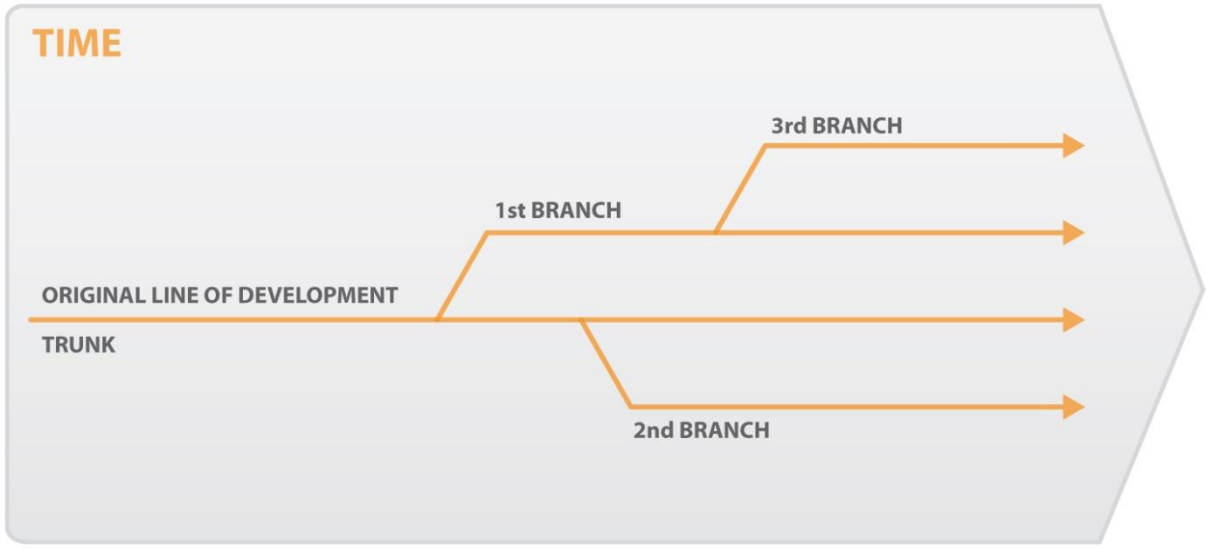
- A tag is:
  - A particular reversion of a tree given a human friendly name (e.g. “release-1.0” instead of r14298).
  - An alternative to revisions for identifying a particular snapshot.
  - Commonly used for milestone builds, and releases.
  - Commonly defined to be immutable; this can be enforced.
- Use `copy` to tag a revision.
  - Simple, common case: tagging a revision.

```
$ svn copy
    http://example.com/svn/calc/trunk \
    http://example.com/svn/calc/tags/release-1.0 \
    -m "Create release-1.0 tag."
```

  - Complex case: tagging a mixed-revision, mixed-path working copy.
- Tags can be:
  - Checked out, switched to, branched from.
  - Renamed, deleted, and restored.

What is a branch?

- A branch:
  - Is an independent line of development, sharing a common history with other lines.
  - Starts as a copy of another line and moves on from there, adding its own history.



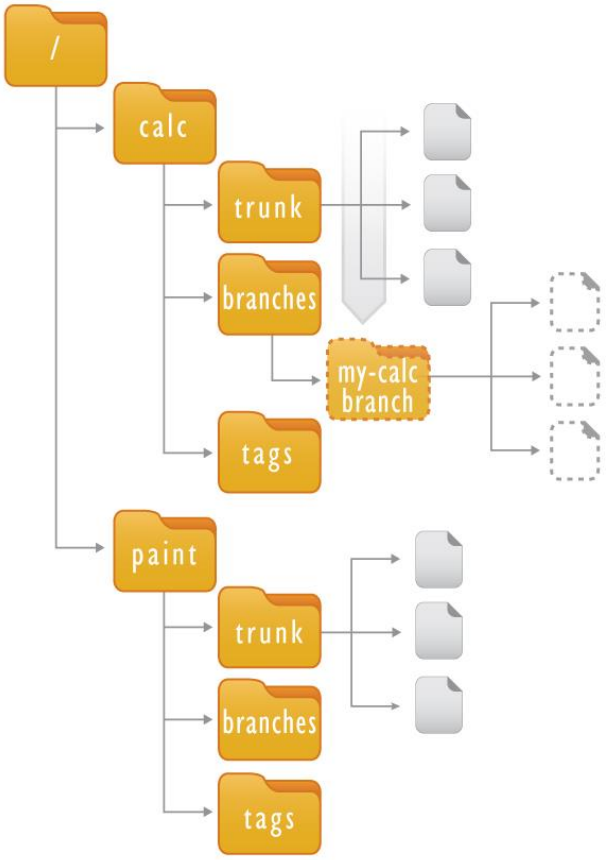
- The trunk is a line of development, just like any branch.
- HEAD refers to the latest revision on the path being referenced.

Create a branch

- Use `copy` to create a branch.

```
$ svn copy
    http://example.com/svn/calc/trunk \
    http://example.com/svn/calc/\
    branches/my-calc branch \
    -m "Create my-calc branch."
```

- Good version control hygiene is to keep the branches under a top-level directory (i.e., branches/).



Work on a branch

To get a local working copy of a branch:

- Check out the branch directly from the repository.

```
$ svn checkout http://example.com/svn/calc/branches/release-1.0
```

- Switch an existing working copy to the branch.

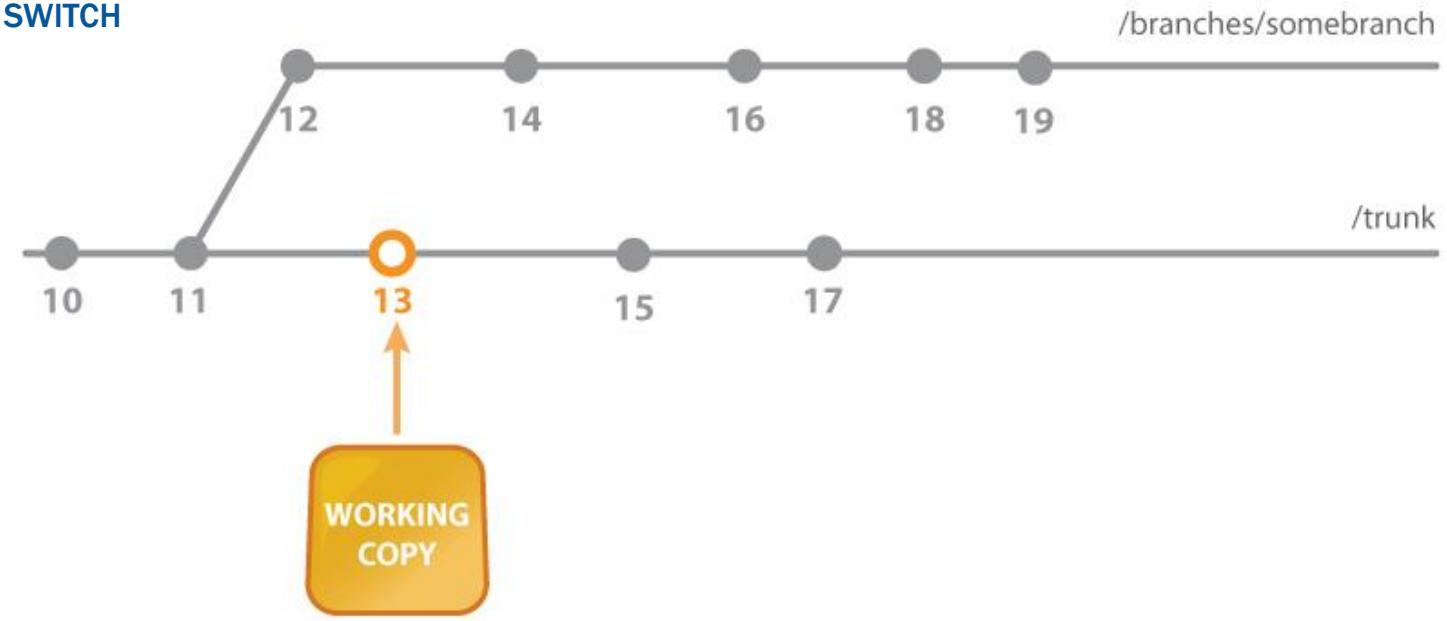
```
$ svn switch http://example.com/svn/calc/branches/release-1.0
```

  - Uncommitted changes are merged with the new branch.
  - Switch can be applied to a branch, directory or file.

Getting around

An **update** is a change in time in relation to the same path (i.e., moving forward or backward in time along a path).

A **switch** is a change in time and space (i.e., moving forward or backward in time on a new path).

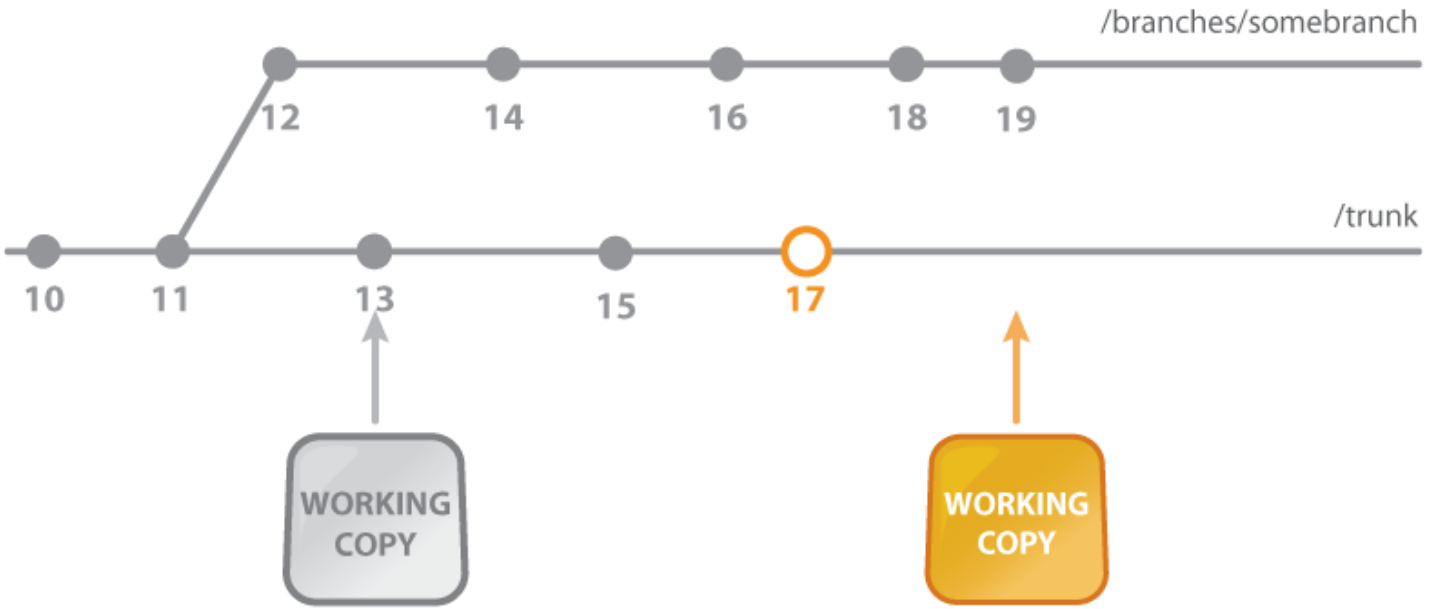


Initial working copy reference point

Getting around (Cont'd)

Move forward in time to latest revision:

```
svn update
```



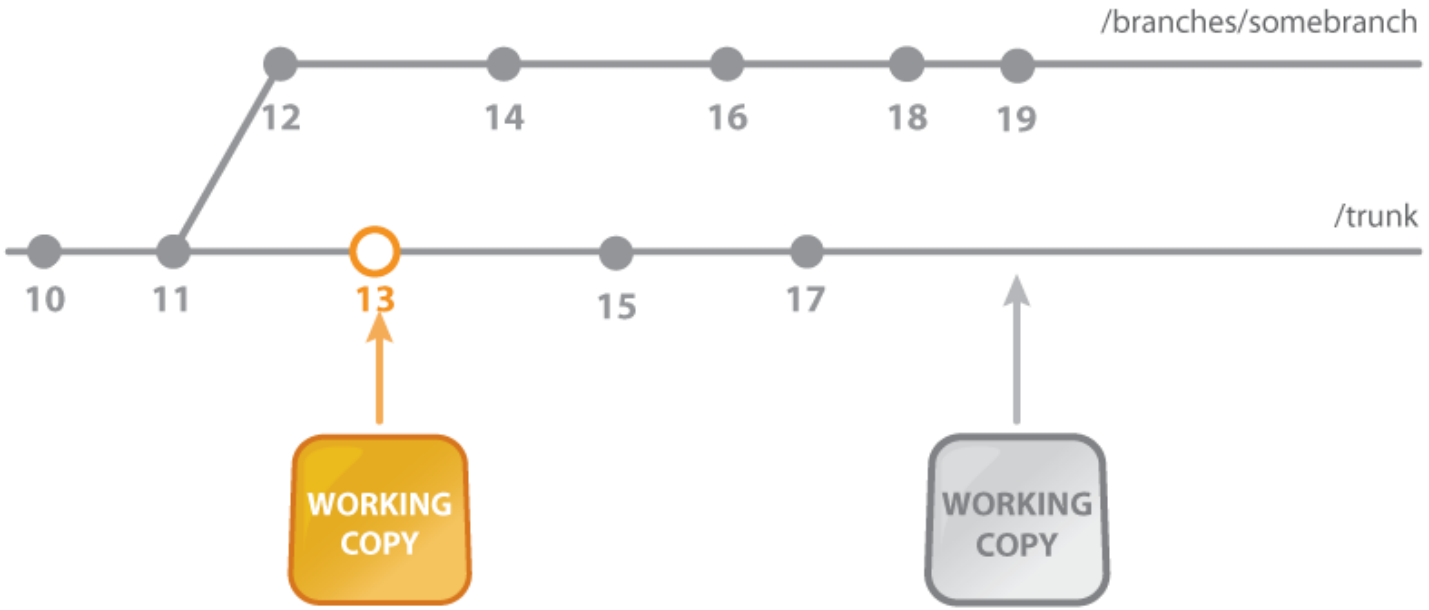
NOTE

File contents on trunk are unchanged since revision 17. However, the revision number is 19 because that is the latest revision in the repository.

Getting around (Cont'd)

Move backward in time:

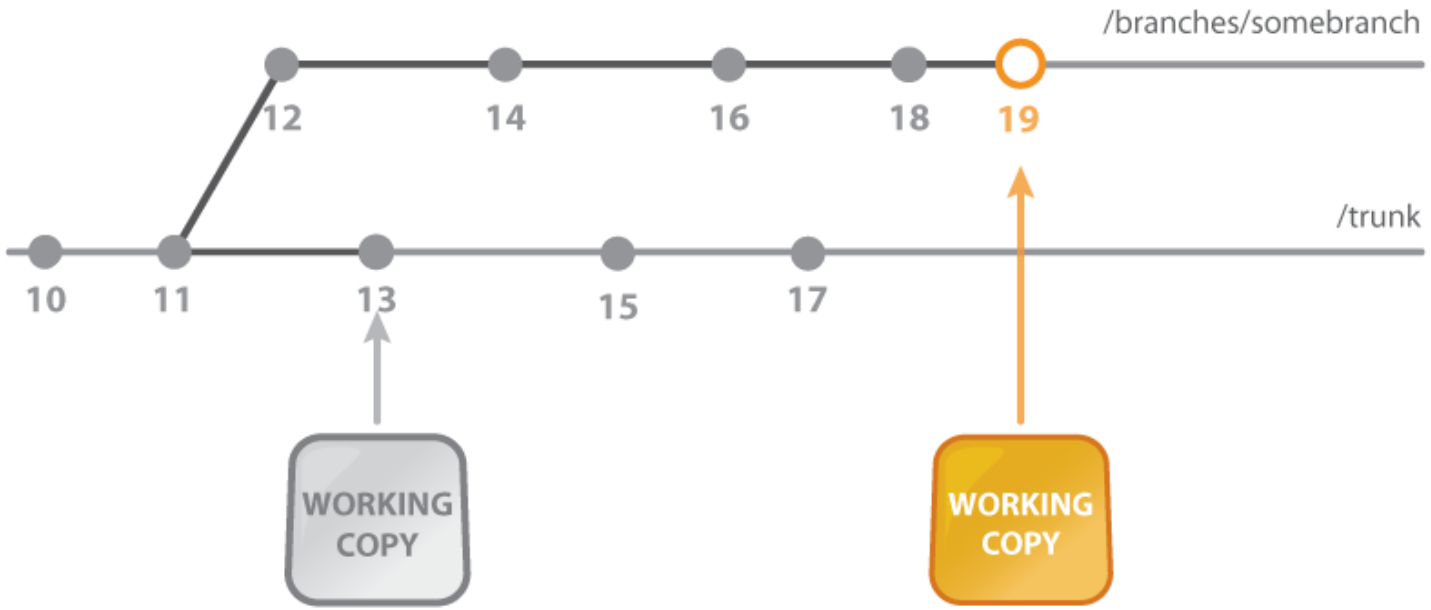
```
svn update -r 13
```



Getting around (Cont'd)

Move in time (forward to the latest) and space:

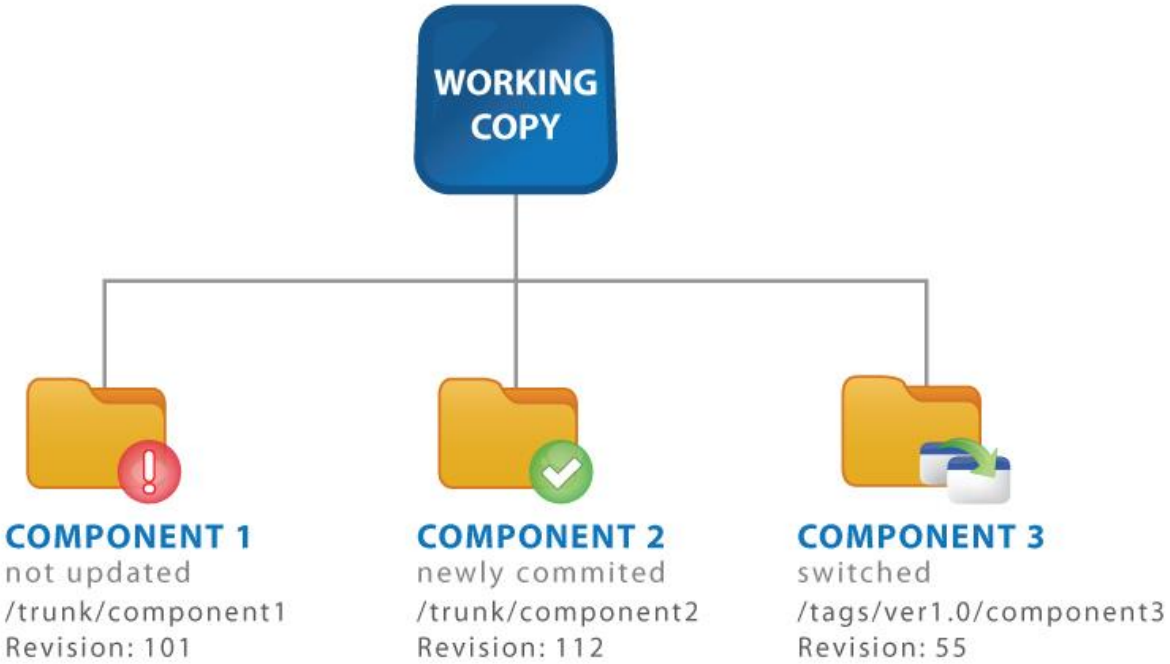
```
svn switch https://example.com/svn/branches/somebranch
```



Mixed revisions & paths

- A working copy can contain:
  - Mixed revisions (see: the ‘Essential Concepts 1’ training course).
  - Mixed paths, e.g. a subtree (directory or file) at a different branch or tag via switch.

- Mixed revisions and/or paths can be a result of selective switches.



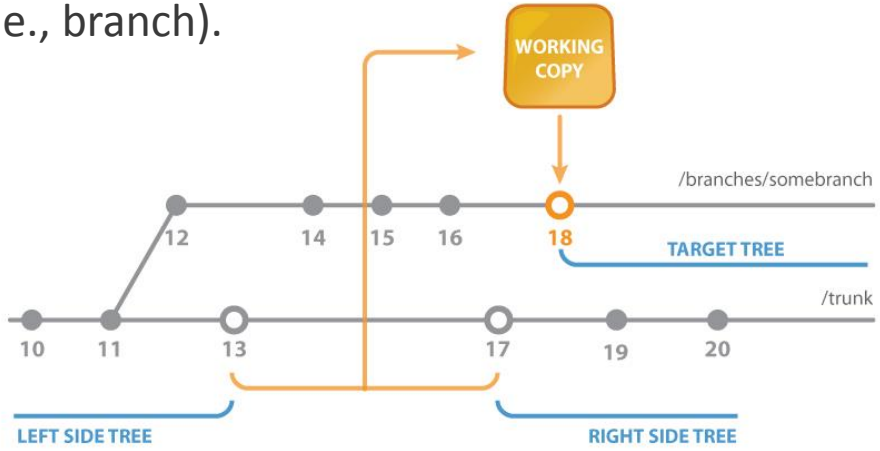
Merging changes

Merging changes

- **Key concept** is Subversion's definition of merge: merge = diff & apply.
  - Compare two trees (from the repository) and apply the differences to a third tree (your working copy).

- The `merge` operation takes three arguments:
  - Initial repository tree (left side of the comparison),
  - Final repository tree (right side of the comparison), and
  - A working copy to accept the differences as local changes (target tree of the merge).

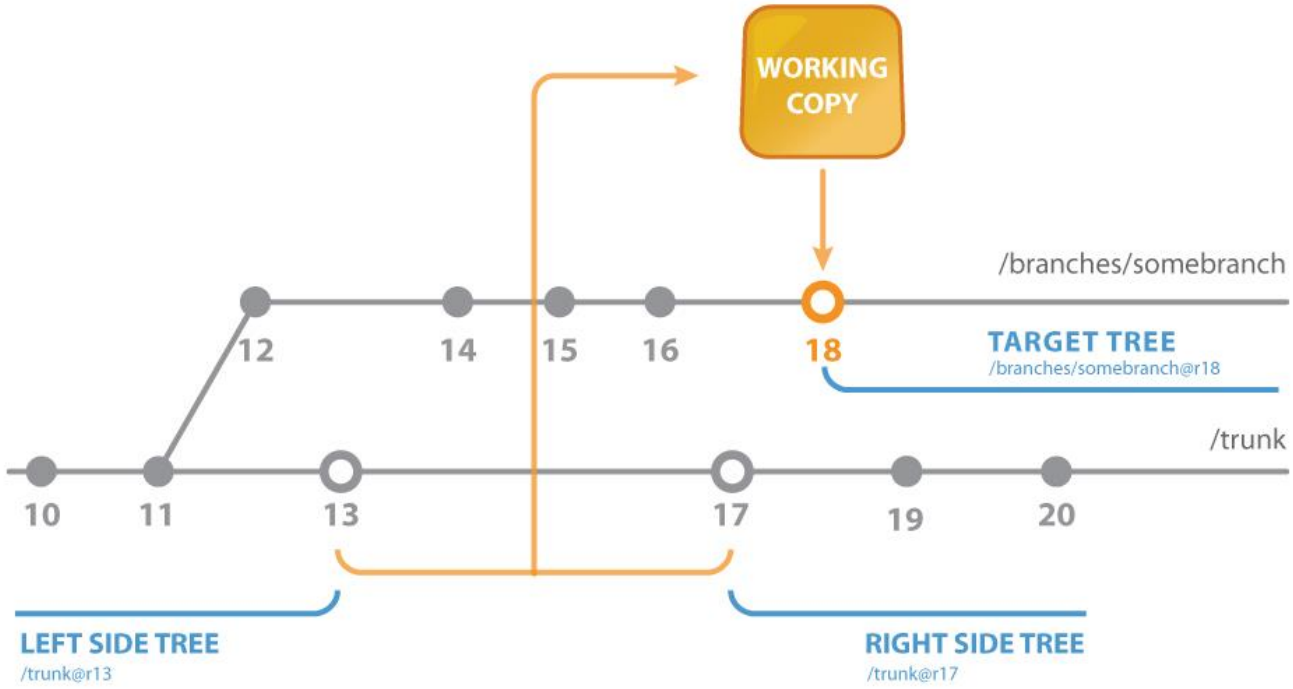
- Merge command will not run if the initial and final repository trees are not related to the final tree (i.e., branch).



Merge a single changeset

Merge revision 17 (e.g., -r 13:17 or -c 17):

```
$ svn merge -r 13:17 https://example.com/svn/trunk .
$ svn commit -m "Merged 13-17 from trunk."
```



Merge a range of changesets

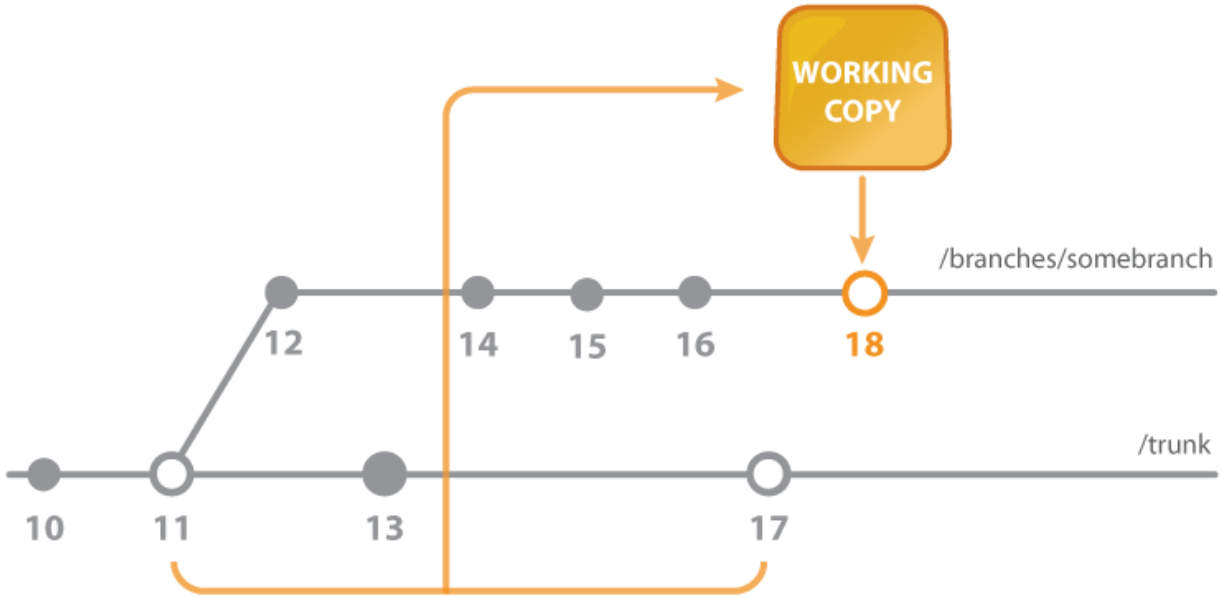
Merge revisions 19 and 20 (e.g., -r 17:20):

```
$ svn merge -r 17:20 https://example.com/svn/trunk .
$ svn commit -m "Merged 17-20 from /trunk."
```

Repeated merges

Merge initial range of revisions:

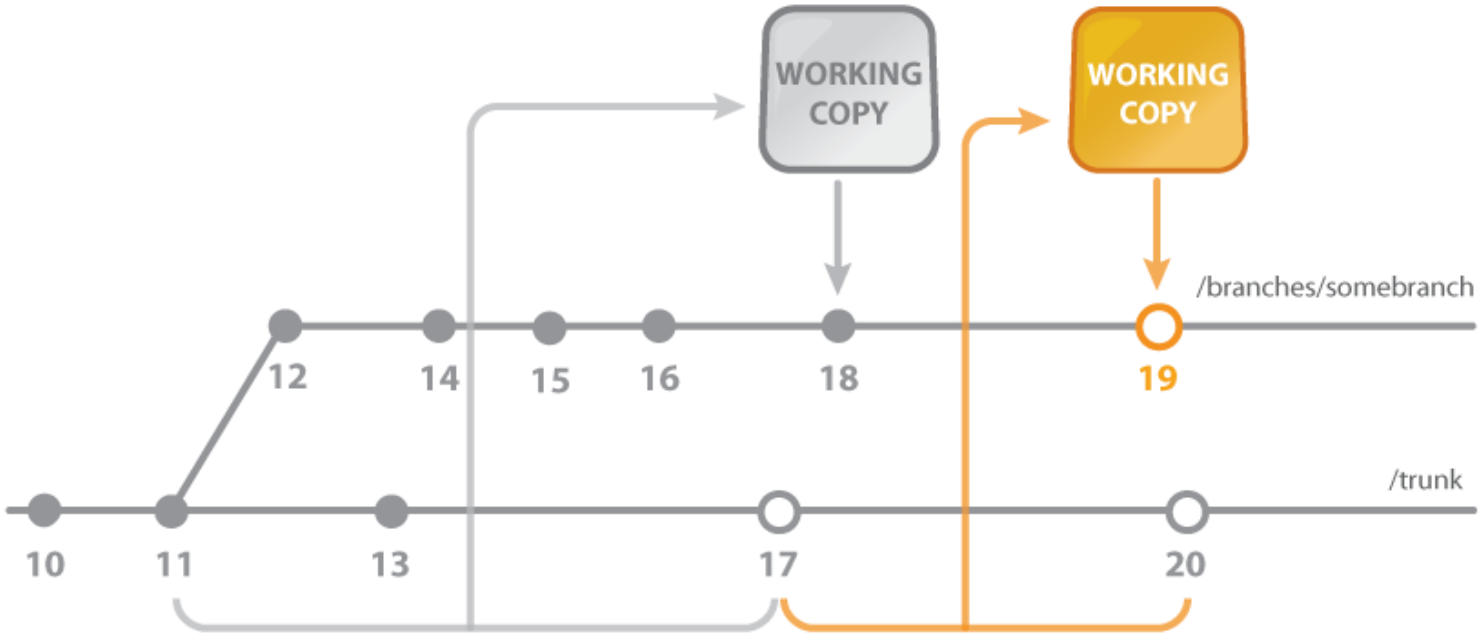
```
$ svn merge https://example.com/svn/trunk .
$ svn commit -m "Merged from trunk."
```



Repeated merges (Cont'd)

Merge subsequent range of revisions:

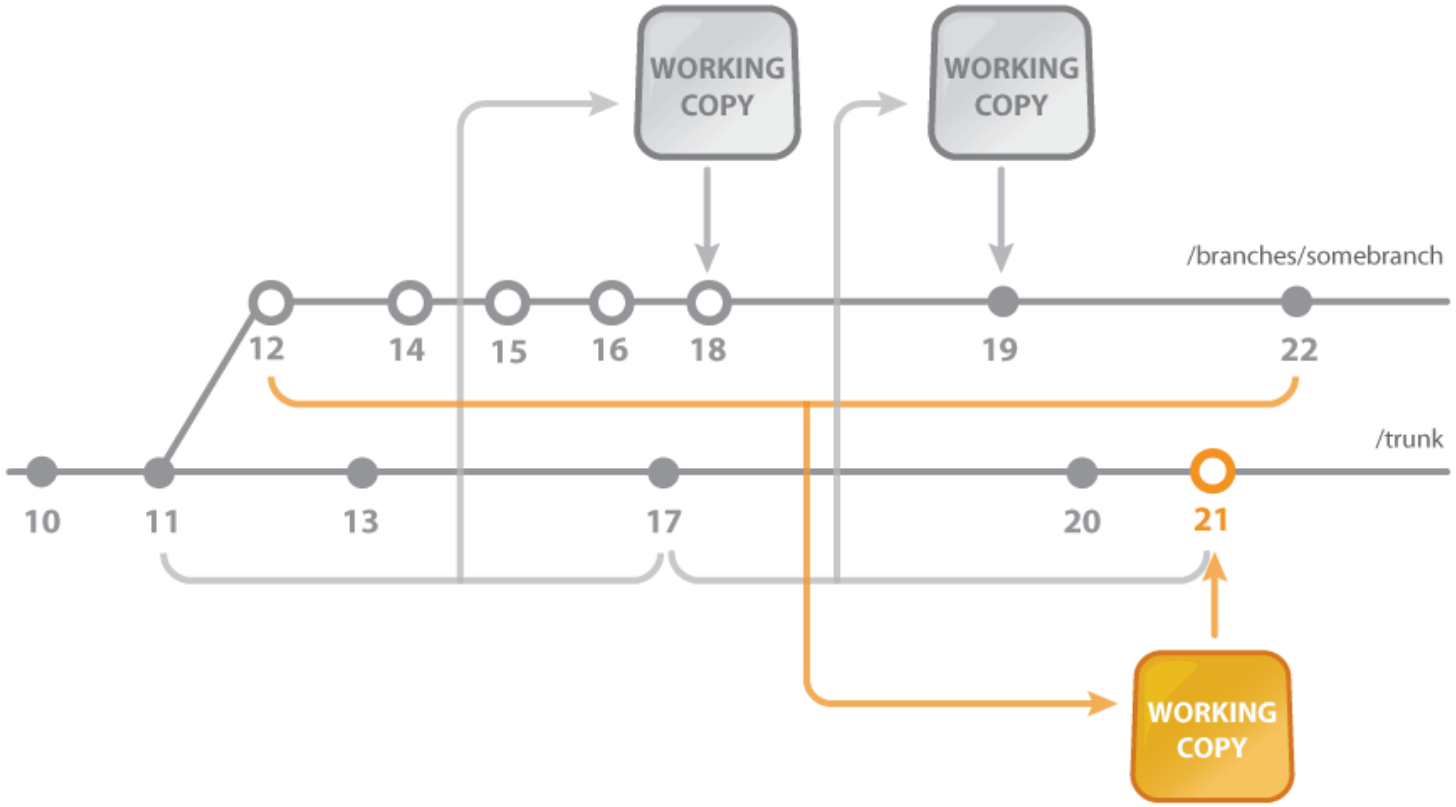
```
$ svn merge https://example.com/svn/trunk .
$ svn commit -m "Merged from trunk."
<time passes and additional work is done on the trunk>
$ svn merge https://example.com/svn/trunk .
$ svn commit -m "Merged from trunk."
```



Child to parent merging

Merge revisions from the child branch (i.e., changesets 14, 15, 16 and 18):

```
$ svn merge https://example.com/svn/branches/somebranch .
$ svn commit -m "Merged 12-22 /branches/somebranch."
```



Rollback (reverse merging)

To roll back changesets, Subversion applies them in reverse:

- An added path in the change set will be deleted.
- A deleted path in the change set will be added.
- An added line in the change set will be deleted.
- A deleted line in the change set will be added.
- A modified line in the change set will be restored.


```
svn merge -r 303:302 http://path_to_repo/trunk
```

Rollback (reverse merging) (Cont'd)

Apply in reverse by specifying the range of revisions in reverse, e.g.

- To revert the change committed in r303:

```
$ svn merge -r 303:302 http://example.com/svn/trunk .
$ svn diff
$ svn commit -m "Revert r303."
```

- To revert r301 through r303:

```
$ svn merge -r 303:300 http://example.com/svn/trunk .
$ svn diff
$ svn commit -m "Revert r301-303."
```

- To resurrect a file deleted in r303:

```
$ svn merge -r 303:302 http://example.com/svn/trunk/foo.c foo.c
$ svn diff
$ svn commit -m "Resurrect foo.c from deletion in r303."
```

Merging best practices

BEST PRACTICES

- Do not have local, uncommitted changes in your working copy prior to a merge.
- Avoid mixed revisions or switched children in your working copy.
- Point your working copy at HEAD for the line of development. (i.e., branch) to which you are merging.
- Identify the source path in your commit message.
- After merging, use status and diff for a sanity check.
- Build and run unit tests before committing.

Merge tracking

What is merge tracking?

- There is no single, commonly agreed upon definition.
- Not all version control systems provide merge tracking and, if they do, they provide it to a varying extent.
- Over the years, CollabNet has worked closely with the Subversion community and selected customers to define and implement merge tracking.
- Identified functionality is being delivered in steps with Version 1.5.0 serving as the starting point, delivering the core functionality, upon which subsequent releases continue to build.
- Prior to 1.5.0 Subversion provided a merge utility, but had no idea when it was being used or had been used.

What is merge tracking in Subversion today?

- Records what revisions (i.e., changesets) have been merged where.
- Prevents duplicate merges.
- Supports manual merges.
- Merges from multiple sources.
- Audits, e.g.
  - What branches contain this exact version of file X?
  - What branches include change C?
  - Is this version of foo.c the 'latest' version? Are there changes out there which are applicable to foo.c, that have not been applied? What are they?

Supported merge use cases

- Repeated merge:
  - Multiple merges from branch A into branch B.
- Cherry picking:
  - Merge one or more individual revisions (i.e., changesets) from branch A into branch B.
- Parent and child branch merges:
  - Merge a parent branch to a child branch and a child branch back to the parent.
- Record manual merge:
  - Allow change sets to be marked as merged without executing a true merge operation.
- Rollback merge:
  - Remove changes previously applied to or merged into the branch.
- Merge auditing:
  - Record and report merge data.

What are tree conflicts?

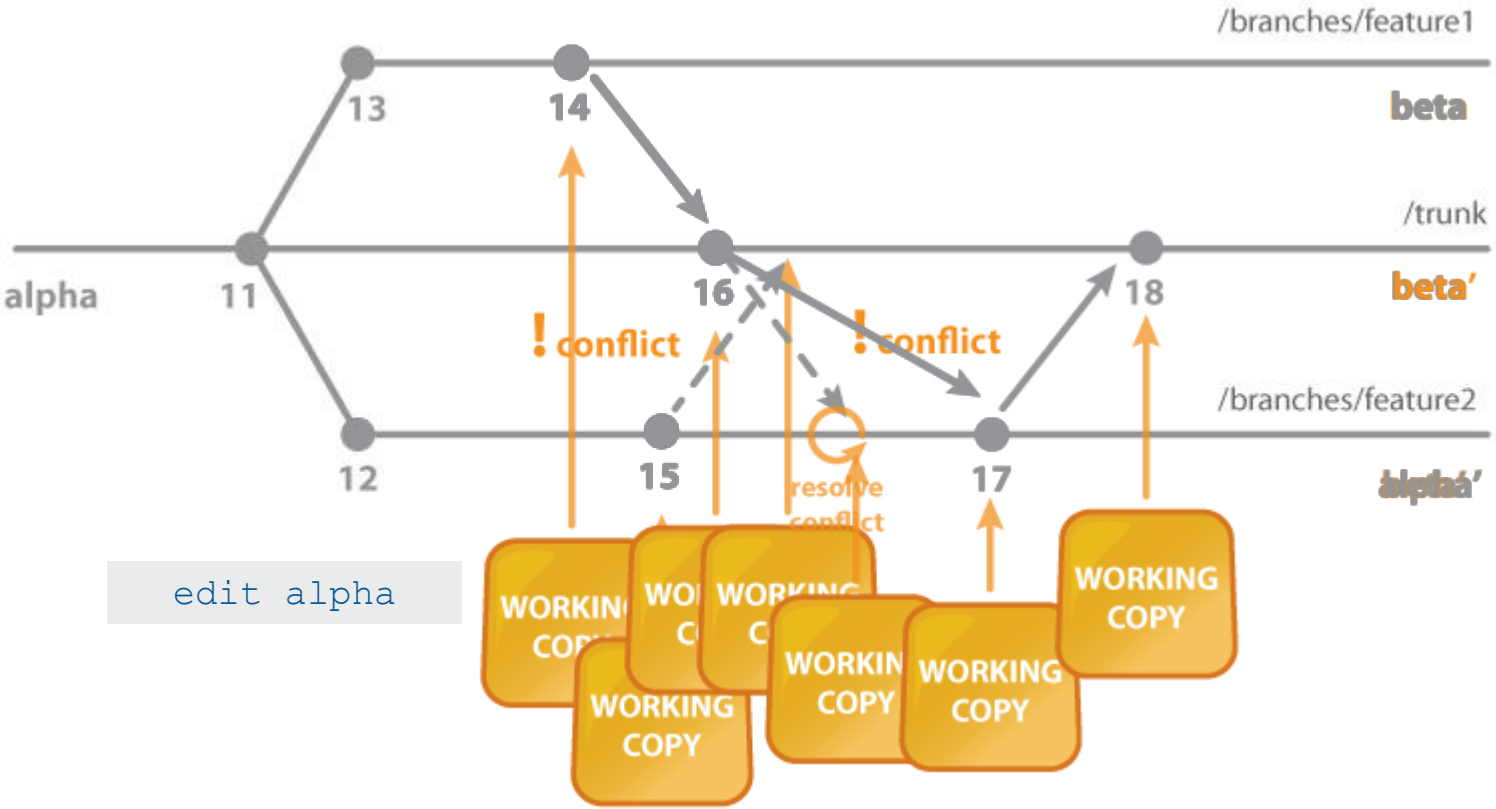
- Tree conflicts occur when an object is deleted by one user, modified by another and then merged.
  - Can happen when merging two branches.
  - Can happen when updating a working copy.
  - Can happen when switching.
  - Can happen when someone renames or moves an object.

- Common use cases:
  - One user deletes and the other modifies.
  - One user modifies and the other renames/moves.
  - One user renames/moves and the other side also renames/moves.
  - One user deletes and the other renames/moves.

Tree conflicts

An example: a rename merged with a modify.

```
svn rename alpha beta
```




```
edit alpha
```

Why is it important to detect tree conflicts?

- They are a common problem with limited visibility, but potentially critical impact.
  - Refactoring is a major and frequent effort for many enterprises.
  - Merging a delete with a modification will happen to every organization – it is just a question of when and how often.
- **Awareness allows you to make the right decisions** – you can't act if you don't know.
- Beyond detection, truMerge is an open source replacement for Subversion's merge operation designed to automate the resolution of tree conflicts (based on predefined resolution policies to the level or use cases logically possible).
  - http://trumerge.open.collab.net/

Other useful features

Automatic property setting

- Versioned properties enable you to store meta data against versioned objects but require that the property **be set** on relevant paths.
- Required (based on your organization's process) properties, if manually applied, can easily be forgotten or the value associated incorrect.
- Upon add or import, Subversion runs a basic heuristic to determine whether a file should have properties set on it based on **either** a set svn:auto-props property or a client/server configuration file.
- Auto props is a configurable option set either on the server or the client via a property or a configuration file.
  - The property svn:auto-props can be set on any folder at or above the folder in which a file is being added (via add or import) which sets pattern and property assignments.
  - There is a single config file on a Subversion server which sets the default values for the settings contained within it.
  - Each user has a config file which overrides the server file, but the svn:auto-props property would be the ultimate definer of patterns and property assignments.

Automatic property setting (Cont'd)

- Set the svn:auto-props property on a folder to automatically map a filename pattern to a versioned property key-value pair.

```
$ svn propget svn:auto-props --show-inherited-props -v calc
Inherited properties on 'calc',
from 'http://svn.example.com/repos':
  svn:auto-props
    *.c = svn:keywords=ID
    *.jpg = svn:needs-lock=*
```

Or

- Configure auto-props in your run-time configuration to do the same.
  - Turn on by setting `enable-auto-props` to `yes`.

```
[miscellany]
enable-auto-props=yes

[auto-props]
*.c = svn:keywords=Id
*.jpg = svn:needs-lock=*
```

- For either, set one line per property to PATTERN = PROPNAME=PROPVALUE.

Automatic keyword expansion

- The property svn:keywords can embed version control meta data into files.
- Typical usage: provide last-modified-date information about a file, e.g. for inclusion in builds.
- The actual expansion is strictly client side and happens at update, checkout, switch and commit.

```
$Date: 2008-06-04 01:31:47 -0400 (Wed, 04 Jun 2008) $
```

Automatic keyword expansion (Cont'd)

- Turn on automatic keyword expansion by:
  - Setting the svn:keywords versioned property on the file with the desired keyword(s) defined as the property's value.
  - Inserting the keyword into the file, between "$" characters (e.g., Id).
    - Keep in mind that keywords are case sensitive.

Keyword	Description
`Date, LastChangedDate`	Date and time the file was last modified
`Rev, Revision, LastChangedRevision`	Revision in which the file was last changed
`Author, LastChangedBy`	Last person to modify the file
`URL, HeadURL`	URL to the file
`Header`	All the other keyword values
`Id`	Compressed summary of all the other keywords
`Your keyword`	A custom string made up of the above keywords

- Example:

```
$ svn propset svn:keywords Id foo.c
```

Custom keywords

- You can also create your own keyword combination of the other keywords using a keyword you define.

Format Code	Definition
%a	The author of the revision given by %r.
%b	The basename of the URL of the file.
%d	Short format of the date of the revision given by %r.
%D	Long format of the date of the revision given by %r
%P	The file's path, relative to the repository root.
%r	The number of the revision which last changed the file.
%R	The URL to the root of the repository.
%u	The URL of the file.
%_	A space (keyword definitions cannot contain a literal space).
%%	A literal '%'.
%H	Equivalent to %P%_%r%_%d%_%a.
%I	Equivalent to %b%_%r%_%d%_%a.

- You define your keyword, and associated value, as the value for the svn:keywords property somewhere at or above this path.
- You place your keyword between dollar signs ($) in the ascii file.

```
$ svn propset svn:keywords "MyKeyword=%r$_%a$_$P" foo.c
```

Externals

- Use the svn:externals property to map a local (i.e., working copy) path to a URL.
  - Can be set on any versioned directory.
  - Value is a multi-line table mapping working copy subdirectories or files to Subversion repository URLs.
- Enables you to construct a working copy out of multiple checkouts, e.g.:
  - Check out different parts of the same repository.
  - Check out different parts of other repositories (directory level only).

```
$ svn propget svn:externals .
     http://doc.collab.net/svn/trunk/notes subversion/notes
-r21 ../../www subversion/web-pages
```

BEST PRACTICE

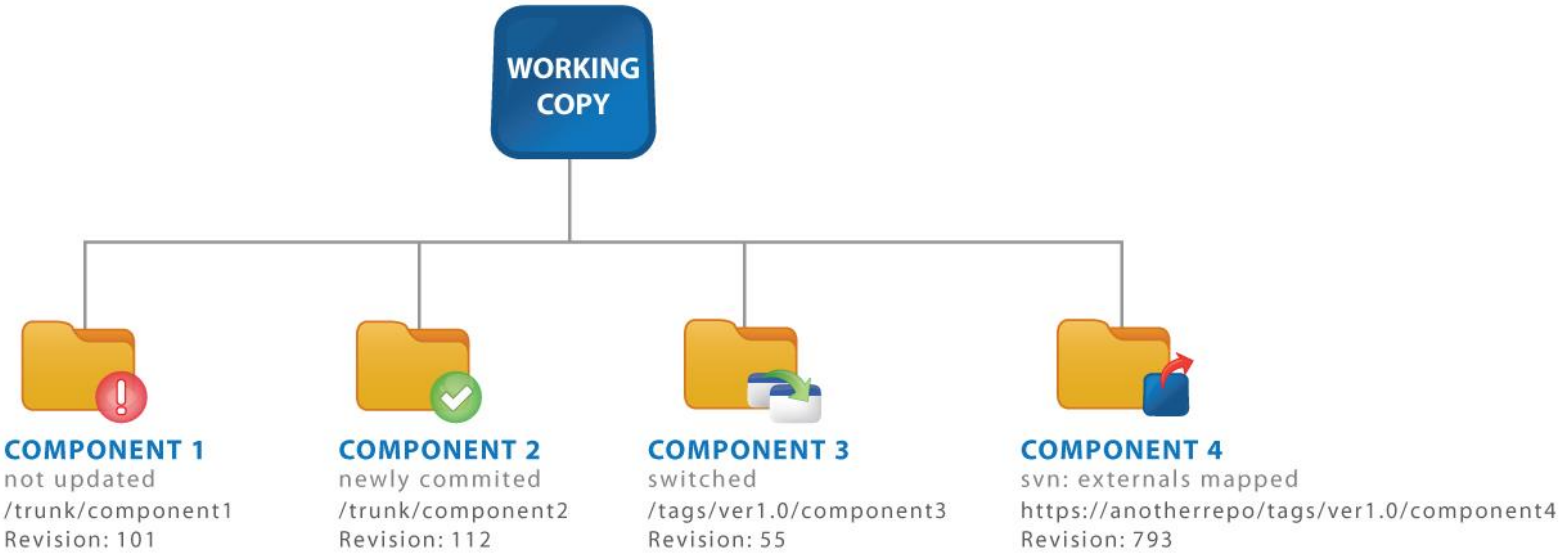
Externals are a way to share code/components.

Externals - limitations

- Externals can point to:
  - Directories - either within the same or a different repository.
  - Files - only within the same repository.
- Subversion operates only partially on disjointed working copies:
  - Paths created via an externals definition are disconnected from the primary working copy (on whose versioned directories the svn:externals property was actually set).
  - Read based operations (e.g., update, checkout, export, and status) will recurse into a part of a working copy that has been created via an external.
  - Write based operations, like commit, will not work if the external involves a *different* repository.
    - For that case, write based operations will require navigating to the externally mapped directory or file.

Mixed revisions & paths

- A working copy can contain:
  - Mixed revisions (see: the ‘Essential Concepts 1’ training course).
  - Mixed paths, e.g. a subtree at a different branch or tag (via switches or externals); or a different repository (via externals).
- This can be a result of the selective use of svn:externals and switches.



Import & export a (sub) tree

- Use `import` to copy an unversioned tree into your repository.
  - Does not require a working copy to execute just the Subversion client.
  - New paths added are treated the same as if they were added individually.
  - Results in a single revision reflecting all the paths added by executing the operation.
  - Note, this does not automatically bring the imported source tree into your working copy – you still need to do a checkout or update.

BEST PRACTICE

Import is the best method to move an existing project into Subversion without history.

- Use `export` to export a clean tree from your repository.
  - Does not require a working copy to execute just the Subversion client.
  - Can be executed on a working copy containing mixed revisions and includes all uncommitted changes to versioned objects (but not unversioned objects).
  - Exporting a tree delivers only a revision of the data set – it excludes the pristine copy and administrative data.
  - Afterwards, Subversion will not know that this tree comes from the repository.

Thank You

CollabNet

About CollabNet

CollabNet is a leading provider of Enterprise Cloud Development and Agile ALM products and services for software-driven organizations. With more than 10,000 global customers, the company provides a suite of platforms and services to address three major trends disrupting the software industry: Agile, DevOps and hybrid cloud development. Its CloudForge™ development-Platform-as-a-Service (dPaaS) enables cloud development through a flexible platform that is team friendly, enterprise ready and integrated to support leading third party tools. The CollabNet TeamForge® ALM, ScrumWorks® Pro project management and SubversionEdge source code management platforms can be deployed separately or together, in the cloud or on-premise. CollabNet complements its technical offerings with industry leading consulting and training services for Agile and cloud development transformations. Many CollabNet customers improve productivity by as much as 70 percent, while reducing costs by 80 percent.

For more information, please visit www.collab.net.

CollabNet, Inc.
8000 Marina Blvd., Suite 600
Brisbane, CA 94005

www.collab.net

+1-650-228-2500
+1-888-778-9793
blogs.collab.net
twitter.com/collabnet
www.facebook.com/collabnet
www.linkedin.com/company/collabnet-inc