Understanding merge types in Teamwork Cloud

On this page

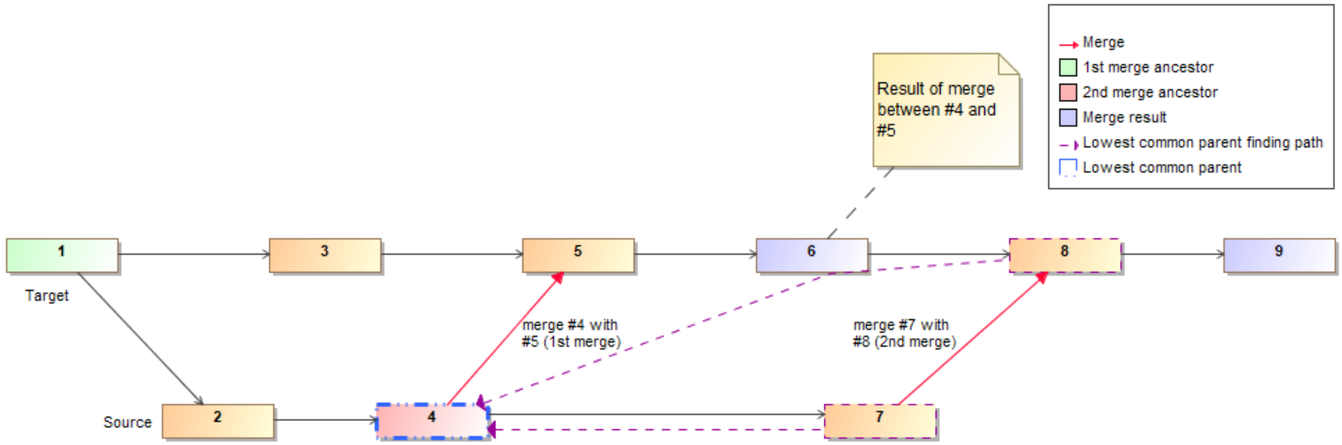
The 3-way merge is the default method used to merge selected Teamwork Cloud projects; the ancestor version is calculated automatically depending on the selected ancestor calculation algorithm. Once the correct ancestor for the two selected project versions is calculated, the **Merge** dialog opens.

You need to start the model merge from the **Tools** (**Tools** > **Project Merge**) menu to be able to change the ancestor selection (**Lowest Common Parent** or **Branch Point**) for the model merge. By default, **Lowest Common Parent** is considered an ancestor of the selected project versions.

Lowest Common Parent (Default)

If **Lowest Common Parent** is selected, the lowest common version for both the Target and Source is considered an ancestor. The lowest common parent of two revisions of the Source and Target in a directed acyclic graph is the nearest revision that has both the Target and Source as descendants. The algorithm considers both the direct parent path and the merged path when traversing and searching for the lowest common parent. In other words, during the repeated merge, the Contributor from the Source that was participating in the last merge is considered to be the common ancestor.

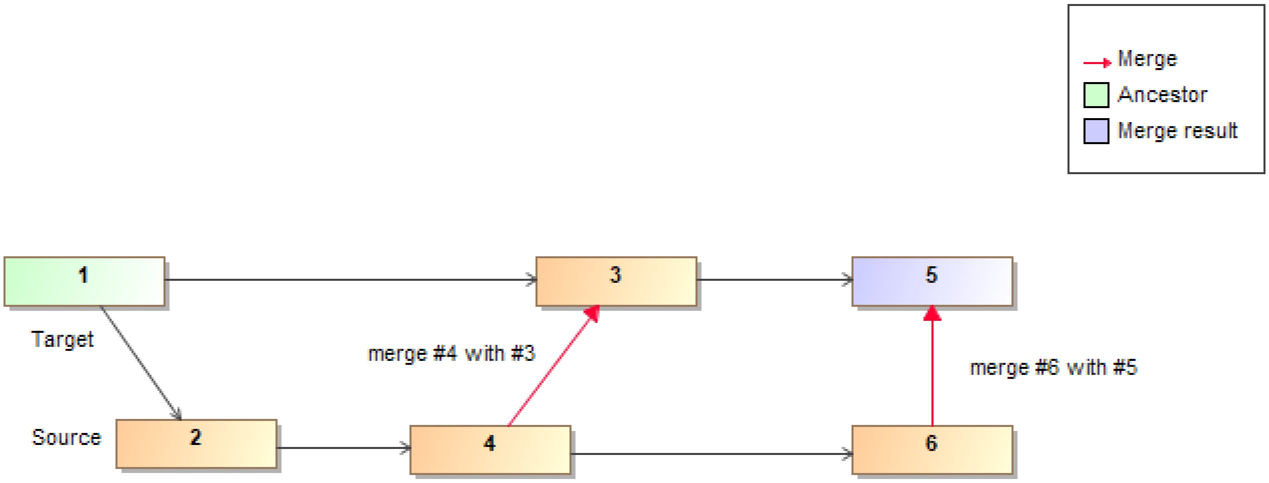
As shown in the image below, version #1 is considered to be a common parent during the first merge between versions #4 and #5. During the second merge between version #7 and #8, we traverse graphs back until we find the common parent for #7 and #8, which, in this case, is #4.


→ Merge
🟩 1st merge ancestor
🟥 2nd merge ancestor
🟦 Merge result
⇢ Lowest common parent finding path
□ Lowest common parent
Result of merge between #4 and #5
1
3
5
6
8
9
Target
Source
2
4
7
merge #4 with #5 (1st merge)
merge #7 with #8 (2nd merge)

Visual representation of the model merge when Lowest Common Parent is considered an ancestor

Branch Point

If **Branch Point** is selected, the version from which the branches separate is considered an ancestor. As shown in the image below, version #1 is considered an ancestor in the first (version #4 is merged with version #3) and subsequent merges (version #6 is merged with version #5).



Visual representation of the model merge when Branch Point is considered an ancestor

Both ancestor calculation algorithms work in an identical way during the very first merge of the selected branches because the same ancestor is taken during the very first merge, no matter if the Branch Point or the Lowest Common Parent is specified. However, it is crucial to understand the importance of the ancestor selection during subsequent merges because the changes may differ greatly from those you expect to see (see examples below).

Repetitive merge

Lowest Common Parent

During the repeated merge, the Lowest Common Parent algorithm always chooses the latest merged Source version as the ancestor. Consequently, you do not need to review historical decisions made during previous merges.

Example #1

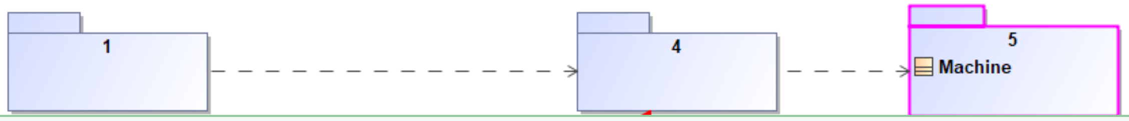
You create a Class *Machine* in the Source branch and merge it with the Target (merge version #3 with version #4). Then continue your development process in the Source and rename *Machine* to *Washing Machine*. When you merge these two branches for the second time (merge version #6 with version #5), the algorithm recognizes that *Machine* was already created in the Target and Source, and, therefore, the only change that needs to be merged is renaming the Class *Machine* to *Washing Machine*.

On the other hand, the **Lowest Common Parent** calculation algorithm may cause confusion, since rejections from the Source branch are considered to be the inverted changes in the Target branch. For example, the created Class that was previously rejected in the Source will appear as a deletion in the Target. As a result, it may seem that some changes that are shown have never been made in the project.

Example #2

Br

Du
bef
any



Example #3

Ho

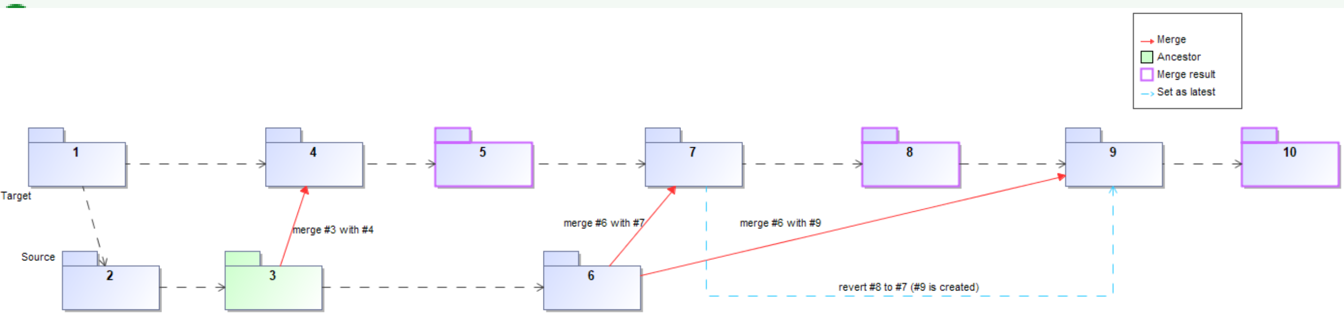
Example #4

Set as Latest

Lowest Common Parent

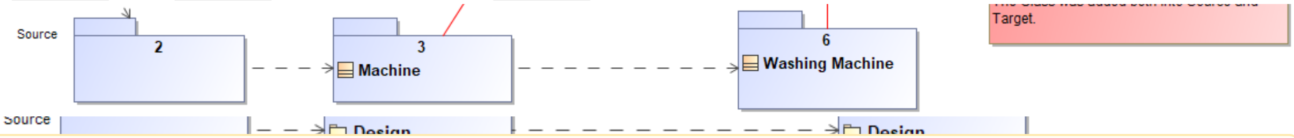
[illegible]

If an earlier version of the project is set to the latest version in the Target branch, e.g., version #8 is reverted to version #7 (version #9 is created), the ancestor is calculated taking the version that was set as the latest (version #7). If there is a preceding merge (version #3 is merged with version #4), the Source version from the last merge (version #3) is considered the ancestor; however, if no merges precede the version that was set as the latest, the version where the branch splits off the *trunk* (version #1) is considered the ancestor.



Br

If a
the
(ve



The default filter hides all the equivalent changes from Target and Source. Therefore, the creation of the *Requirements* package is hidden by default.