# Updating Commits:

Considering an iteration that is verifying a monitored repository rep1, the steps needed to update its commits in the database are the following:

1. Retrieve the previous local snapshot from disk 🡺 previousSnapshot
2. Retrieve current snapshot from repository 🡺 currentSnapshot
3. Identify new local commits, comparing current and previous snapshots 🡺 newCommits = currentSnapshot \ previousSnapshot
   1. If previousSnapshot doesn’t exist, then newCommits = currentSnapshot
4. Retrieve commit count for the system from database 🡺 C
5. If C is not 0, retrieve all commits from database that do not exist in at least one of the repositories related to rep1, this is, commits that were not found either in rep1 or in at least one of its partners. All other commits in the database are considered to be already synchronized 🡺 commitsNotFoundInSomeReps
6. If C is not 0, retrieve all commits from database that exist in newCommits 🡺 newCommitsInDatabase
7. If C is not 0, identify which of newCommits commits do not exist in newCommitsInDatabase 🡺 commitsToInsert = newCommits \ newCommitsInDatabase
   1. If C is 0 all commits should be sent to database 🡺 commitsToInsert = newCommits
8. For each one of commitsToInsert commits, update the list of repositories where it is found, according to the discussion in section 2
9. Insert commits from commitsToInsert in the database
10. Identify commits that were deleted locally since previous run: commitsToDelete = previousSnapshot \ currentSnapshot
    1. If previousSnapshot doesn’t exist, then commitsToDelete = null
11. If commitsToDelete is not null, delete commits from commitsToDelete in the database
12. Save current snapshot to disk
13. Identify local commits that were in non-tracked branches, that now are in tracked branches 🡺 nowTrackedCommits
14. Update commits from nowTrackedCommits, changing tracked attribute to “true”
15. Remove deleted commits from commitsNotFoundInSomeReps 🡺 commitsToUpdate = commitsNotFoundInSomeReps \ commitsToDelete
16. For each one of commitsToUpdate commits, update the list of repositories where it is found, according to the discussion in section 2
17. Update commits from commitsToUpdate that had the “foundIn” list changed in the database

# Updating the list of repositories where a commit is found

Considering a single commit X, the following table shows what can be concluded regarding its existence in the repositories related to the monitored repository. The ahead lists contain commits that exist in the monitored repository and do not exist in its partner (one list per partner). The behind list contain commits do not exist in the monitored repository but exist in any of its partners (one list per partner).

Lines marked with 'Not valid' denote impossible situations, once that a single commit X cannot be both ahead AND behind (it exists or does not exist locally). This is, if X exists locally, its status will be whether ahead (if it also exists remotely) of synchronized. It will never be behind, because it exists locally. The same reasoning applies where X does not exist locally: it will be whether synchronized or behind, but never ahead.

Table 1 – Commit existence based on behind and ahead lists

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ahead List | | Behind List | | Existence | | |
| rep A | rep B | rep A | rep B | Local | rep A | rep B |
| F | F | F | F | see conclusion 1 below | | |
| F | F | F | T | F | F | T |
| F | F | T | F | F | T | F |
| F | F | T | T | F | T | T |
| F | T | F | F | T | T | F |
| F | T | F | T | Not valid | | |
| F | T | T | F | Not valid | | |
| F | T | T | T | Not valid | | |
| T | F | F | F | T | F | T |
| T | F | F | T | Not valid | | |
| T | F | T | F | Not valid | | |
| T | F | T | T | Not valid | | |
| T | T | F | F | T | F | F |
| T | T | F | T | Not valid | | |
| T | T | T | F | Not valid | | |
| T | T | T | T | Not valid | | |

**Conclusions:**

1. If commit is not in any list then it either:
   1. Does not exist locally: then it does not exist in any of the repositories in the push / pull lists either;
   2. Exists locally: then its existence will be based on whether it belongs to a tracked or to a non-tracked branch:
      1. If it belongs to a tracked branch, it means that it also exists in all repositories in the push / pull lists;
      2. If it belongs to a non-tracked branch, it means that it exists only locally;
2. If commit is in at least one behind list, then it does not exist locally and exists in all repositories that have at least one behind list containing it;
3. If commit is in at least one ahead list, then it exists locally and in in all repositories that DO NOT have an ahead list containing it.

**Problem:** Commits from non-tracked branches do not appear in ahead or behind lists. Hence, they will erroneously map to conclusion 1.

**Possible solutions:**

1. It would be necessary to create a clone of each repository in the push / pull lists, instead of using the working clone, then create the commit map for it and use set theory to find out which of the commits are missing in each repository. This can led to performance and memory problems;
2. For each commit not found in the local set, get its info from the topology