ADVANCED PROGRAMING PRACTICES

Revision Control Systems

REVISION CONTROL SYSTEMS

What is revision control?

- Revision control in general is any kind of structured practice that tracks and provides control over changes to files, particularly source code.
- As teams design, develop and deploy software, it is common for multiple versions of the same software to be deployed in different sites and for the software developers to be working simultaneously on updates.
- This requires a solution to keep track of the different versions, and to manage the incorporation of new changes in the appropriate versions.
- Many different applications have been designed over the years to provide a software solution to this problem.

Why?

- Why use a revision control system?
  - To have a common repository for all project files available and updated remotely.
  - To make sure that concurrent changes to the same file are properly handled.
  - To allow branching and merging of versions as well-managed operations.
  - To help handle merging operations when the same part(s) of a file has been changed by more than one programmer.
  - To avoid the proliferation of different disjoint copies of files/projects.
  - To make sure that everybody in a team is always using the correct version of project files.
  - To ensure a proper rollback sequence in the event that some changes need to be undone.
  - To easily compare the differences between different file version.
  - To easily access previous project version.

Goals

- Maximizing productivity by automating code integration tasks.
- Reducing the cost related to confusion and mistakes.
- Maximizing software integrity, traceability, and programmer accountability.
- Assisting developers in providing cost-efficient coordinated changes to software products and components.
- Accurately recording the composition of versioned software products evolving into many revisions and variants.
- Reconstructing previously recorded software components versions and configurations.

General functioning

- First, an initial repository is created that contains the files composing a software system, i.e. an IDE's workspace.
- After creation of the repository, a "snapshot" of the files stored on the repository (e.g. the latest revision) can be retrieved, thus creating a local copy of the files that can be worked upon in isolation from the repository.
- When the changes to the local copy are completed to satisfaction, the changes can be committed to the repository, thus creating a new version of the files that have been changed in the repository.
- If more than one user is trying to commit changes to the same file, a merge operation needs to be performed, which can be semi-automated, but is often non-trivial if extensive changes were applied concurrently.
- Merge conflicts are created when the same line(s) of code has been changed in more than one concurrent change. These need to be figured out manually and can be tedious, time consuming and frustrating.
- Frequent committing reduces the complexity of merge operations.
- Generated files (e.g. IDE project files) should not be stored in a repository, as they take up space uselessly and may increase the chance of conflicts.

General Concepts

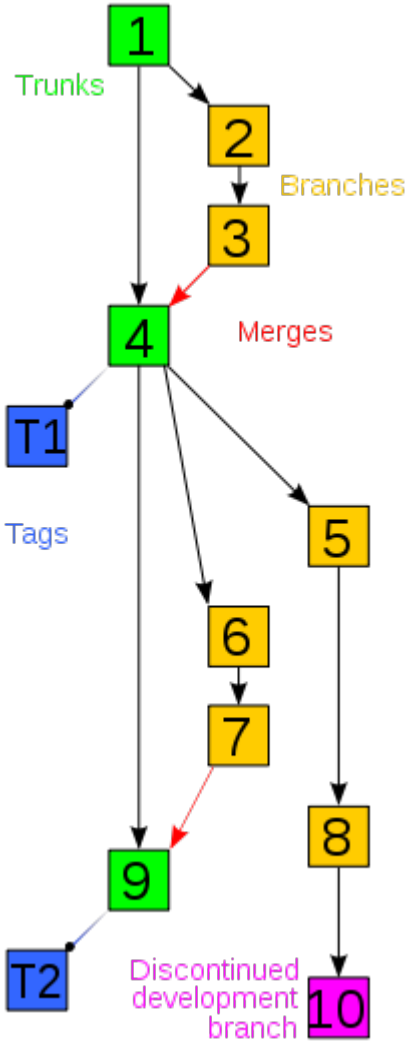
- Repository
  - This is where a copy of the project files and directories are stored. A special file structure is used for tracking the differences between successive versions of a file.
  - Most revision control systems have a remote repository and a local repository.
- Working Copy, Workspace
  - This is a copy of a group of the actual files in your local file system (previously pulled from a repository).
  - If the IDE integrates the use of a revision control plugin, the working copy is automatically mapped onto the project workspace.
  - If you are using a separate revision control software client (e.g. Sourcetree, Tortoise GIT, GitHub Desktop, etc), you may have to map your working copy files into the IDE's workspace manually.

General Concepts

- Commit
  - This is the process of saving files from the working copy directory to the local repository. You may commit specific files or a whole project to the repository. Generally, only the files that have changed since the last pull are subject to the commit operation.
- Push
  - Commits made to the local repository. The push operation aims at applying the local commits to a remote repository.
- Checkout
  - This is the process of retrieving the changes from the repository, i.e. downloading a local copy to your machine. Checkout does not merge conflicting changes.
- Pull
  - Retrieves the changes from the repository and applies a merge on the local copy.
- Cherry-pick
  - Retrieves a single commit from the repository, then applies it to the local copy, creating a new commit for this change applied to the local copy.

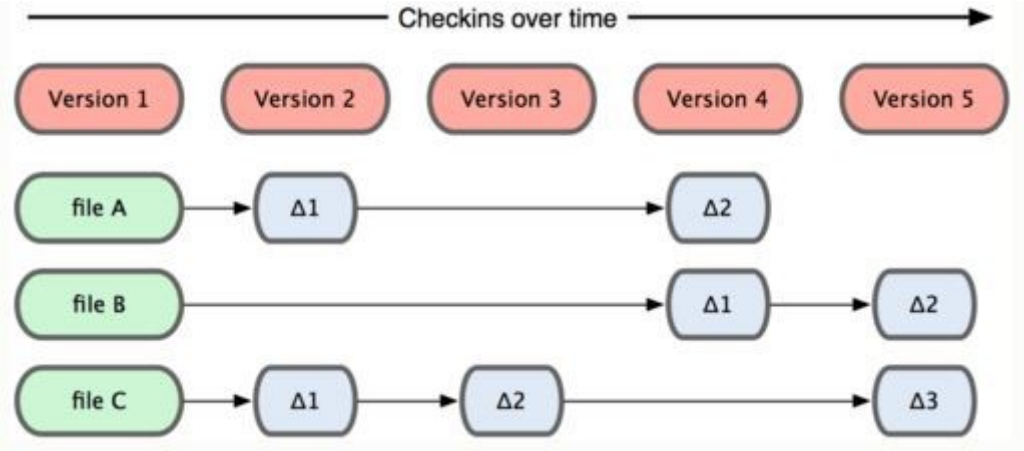
General Concepts

- Trunk, branching and tagged versions
  - A branch is a collection of revisions that for some reason should not be merged onto the main trunk of development.
  - For example, if we want to work on a part of the code doing changes that we are not going to share until we are satisfied with the result we could work on our own branch, without disturbing anyone else's code.
  - Branching is a powerful mechanism for controlled isolation.
  - The original set of versions, before any branch was created, is called the main line or main branch, or trunk (in green).
  - After a branch is created the trunk is still the default version.
  - We can always merge changes from a branch into the trunk or vice-versa (though it may be a complex operation).
  - At any time, one can tag the current state of revisions to create a tagged version that can be referred to by name or number later (in blue).
  - By default, operations are applied on the latest version on the trunk.
  - To switch branches, one needs to checkout on a branch.



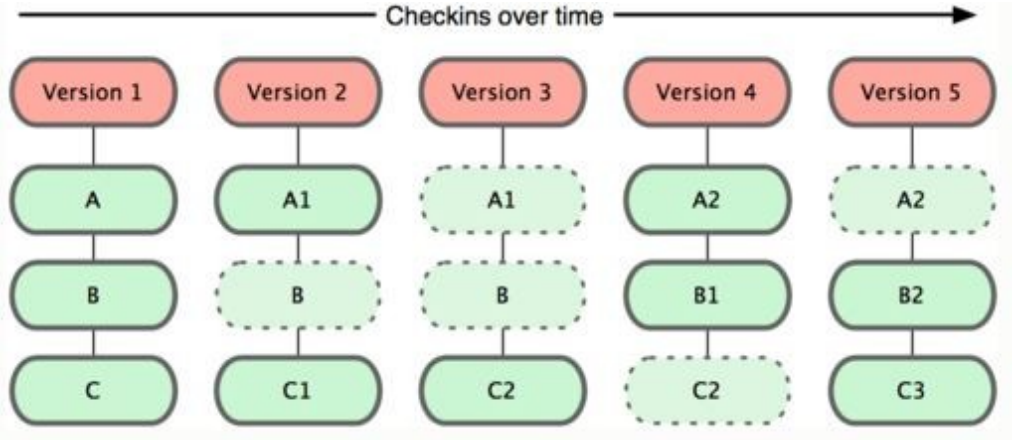
Approaches to store the changes

- Store individual changes to files (deltas)



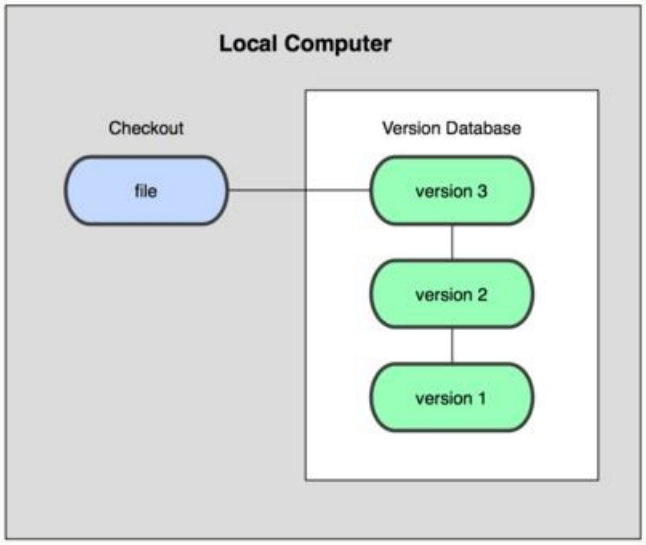
Approaches to store the changes

- Store entire files as new versions when they are changed



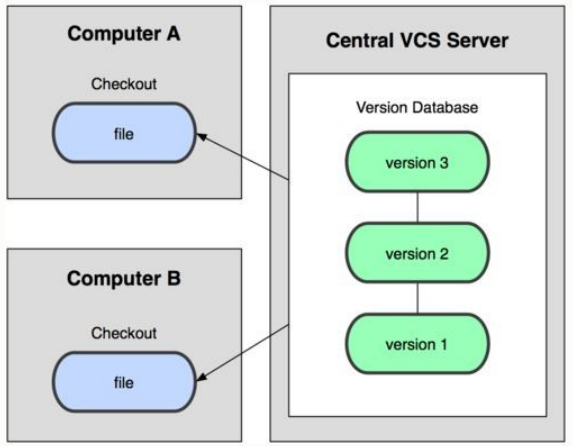
Approaches in repository distribution

- Local version control systems
  - Only a local repository exists that manages the revisions locally
  - No remote access
  - No concurrent changes
  - Normally uses the deltas storage approach
- Examples:
  - SCCS: Source Code Control System
  - RCS: Revision Control System



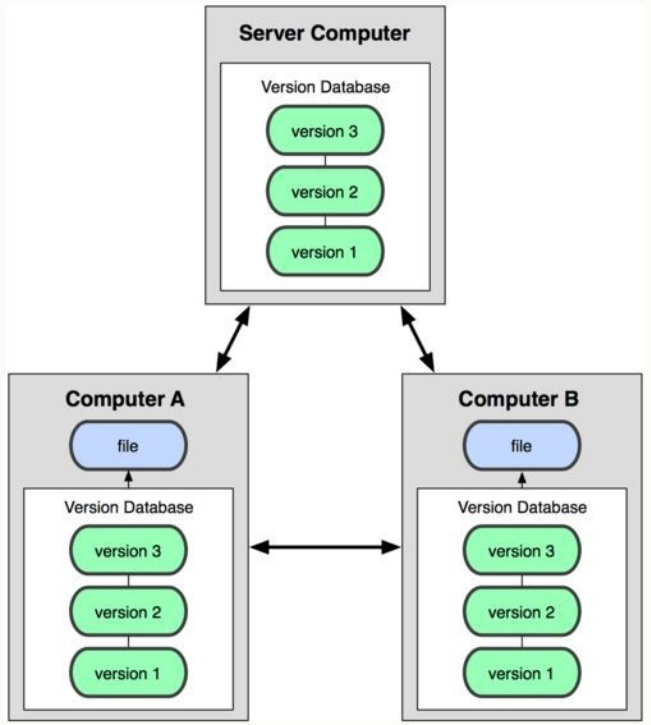
Approaches in repository distribution

- Centralized repository
  - A single server contains all the recorded versions
  - Clients can checkout any version remotely
  - Many advantages over local revision management
    - Distributed revisions
    - Teamwork
    - Project management
  - Disadvantages over distributed repositories
    - Single point of failure
    - File access concurrency
- Examples
  - CVS: Concurrent Versioning System
  - Apache Subversion (SVN)
  - Perforce – Helix, Hansoft
  - IBM Rational Clear Case



Approaches in repository distribution

- Distributed repository
  - A server contains all the recorded versions
  - Any client can then act as a server
  - Clients can checkout any version remotely from any server
  - Many advantages over centralized repository
    - No single point of failure
    - Teamwork with individual initiatives/storage
    - Project management
  - Popular in the free software movement
- Examples:
  - Git, GitHub, Bitbucket, Mercurial, GNU Bazaar or Darcs



In the project

- You are required to use a repository.
  - Every team member should have multiple commits during the production of each build.
  - There should be *dozens* of commits during each build.
  - Commits should be well-distributed over the duration of the production of the build.
  - The lab instructor will guide you in the usage of GitHub, though you may use something else.

- You are required to use a continuous integration tool
  - Setup so that every time a commit is pushed to the repository
    - the code is compiled
    - the tests are run
    - the javadoc is compiled
  - If any of these fail, the commit/push is rejected

References

- Scott Chacon. Pro Git. First Edition. Apress. 2009. ISBN-13: 978-1430218333
- http://git-scm.com/docs
- http://git-scm.com/book
- http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
- http://git-scm.com/