Revision Control Systems

What is revision control?

* Revision control 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.

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 the branching of versions in a seamless operation.
  + To avoid copying all files when creating a new version of a project.
  + 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 compare the differences between different file versions (using diff).
  + To access previous project versions seamlessly (using version tagging)

Goals

* Maximizing productivity by automating integration tasks.
* Reducing confusion, minimizing mistakes.
* Maximizing software integrity, traceability, and programmer accountability.
* Assisting developers in providing 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

* A repository is created that contains some versions of the files composing a software system.
* After creation of the repository, a “snapshot” of the files (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 software.
* If more than one user is trying to commit changes to the same file, a merge operation has to be performed, which can be semi-automated, but is often non-trivial if extensive changes were applied.
* Frequent committing reduces the complexity of merge operations.

Concepts

* Repository
  + This is where a copy of your files and directories are stored. A special file structure is used for tracking the differences between subsequent versions of a file.
* Working Copy and Workspace
  + This is a copy of a group of files in your local file system (previously pulled from a Repository).
  + If the IDE integrates the use of a revision control plugin (e.g. Eclipse or Netbeans CVS or GIT plugin), the Working copy is automatically mapped onto the project workspace.
  + If you are using a separate revision control software client (e.g. WinCVS, Tortoise CVS, Git, Sourcetree, etc), you have to map your working copy files into the IDE’s workspace manually.
* Commit
  + This is the process of saving (or pushing) files to the Repository. You may commit specific files or a whole project to the Repository. Generally, only files that have changed since the last pull are subject to the commit operation.
* Checkout
  + This is the process of retrieving (or pulling) files from the Repository, i.e. downloading a local copy to your machine.
* Trunk干线的, branching and tagged versions
  + A branch is a collection of revisions that for some reason should not be committed 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 not satisfied with the result we could work on our own branch, without disturbing anyone else.
  + Branching is a powerful mechanism for controlled isolation.
  + The original set of versions, before the branch was created, is called the main line or main branch, or trunk.
  + 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.
  + By default, commit and checkout operations are applied on the latest tagged version on the trunk.

