CSc 179 – Configuration Management (CM)

Credits:

http://en.wikipedia.org/wiki/Software_configuration_management

http://en.wikipedia.org/wiki/Distributed_revision_control

Chacon & Straub: “Pro Git”

Software Quality Assurance From theory to implementation, Daniel Galin

Github Tutorial For Beginners: https://www.youtube.com/watch?v=0fKg7e37bQE

Agenda

- Why is Configuration Management (CM) needed?
- What is Configuration Management?
- SCM: Support, Control, and Service
- Software Configuration Item
- SCM Functional areas
- Repository
  - Location
  - What to put in?
- Branches: Merging and Conflicts
- Working scenarios
- Git Introduction
  - GITHUB tutorial

Why is CM needed?

- ``This worked yesterday and doesn't work now.” What happened?
- ``The user manual says to do this, but when I do it, something different happens."   Which is correct, the manual or the code? Why was one changed?
- ``The code changes that I made last week are no longer in the code.” What happened to the fix? Who changed the code and Why?
- ``The listing doesn't match what the program does!" Which is correct?
- ``Did the bug get fixed in this copy, too?"

Why is CM needed? (Cont)

- Control the changes
  - Versions of document need to be combined to form a product, or configuration
  - With many people working on many files, inconsistencies can occur
- Required for testing
  - We must know and control what source was used to produce a software system in order to know what is being tested
  - We need to be able to build and rebuild a software system reliably

What is Configuration Management?

- Software CM is a discipline for **managing the evolution** of software systems throughout all stages of the software life cycle.
- SCM is a component of SQA system.
  - Infrastructure component
  - Organizational framework
- **SQA (Software Quality Assurance)** teams are often required to take the responsibility of managing the CM system.

SCM: Support, Control and Service

- **Support**
  - Developers, organizations, customers
- **Control**
  - Specifications, documents, software, and other deliverables
- **Service**
  - “SCM is a service provider in that it supports people and controls data.”

Software configuration item

- **Software configuration item (SCI):**
  - An approved unit of software code, a document or piece of hardware that is designed for configuration management and treated as a distinct entity in the SCM process.
  - The main criterion: whether needed for future development or maintaince.
- Each SCI must have a unique name:
  - Augment the name with various attributes such as type of document, OS, language, etc.
  - It is not a DESIRE practice to have an SCI change name for each version. ➔ Use a consistent name and let the CM system to handle versions.
    - ~~main09-01-2019a.java, main09-01-2019b.java, main09-02-2019a.java~~
    - ~~main.java~~

Typical Software configuration item (SCI)

- **Documents**
  - development plan, requirement/design specifications, test plan, test report, user manuals, maintenance plan, change requests, CM plan, version description, standards, etc.
- **Software code**
  - source code, prototype
- **Data files**
  - parameters, configuration settings, etc.
  - test cases and test scripts
- **Software development tools**
  - Compilers, debuggers, linkers, etc.
  - IDE: Eclipse, Intellij, etc,
  - Design tools : UML tools
  - Build automation tools
  - Code review tools
  - Performance analysis tools

Version

- **SCI version**:
  - The **approved state** of an SCI at any given point of time during the development and maintainace process.
- **Software configuration version**:
  - An approved selected set of documented SCI versions that constitute a **software system or document** at a given point of time.
  - The activities to be performed are controlled by SCM procedures.
  - The software configuration versions are released according to the cited procedures.

SCM functional areas

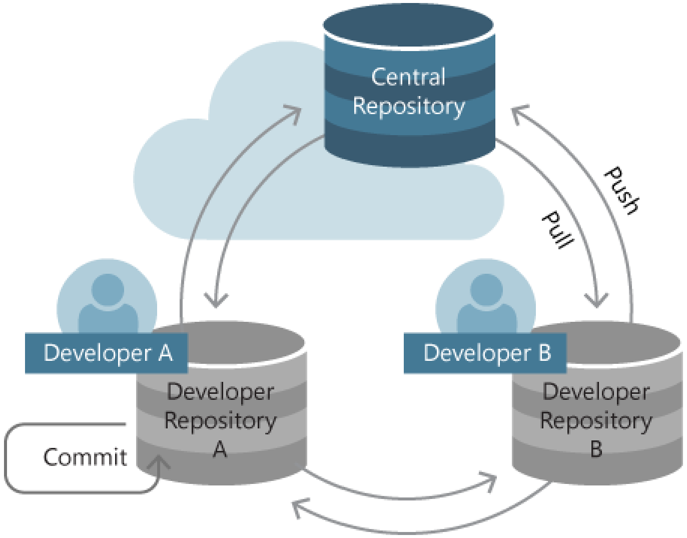
- **Identification**
  - identify components, structure
- **Control**
  - control releases and changes
- **Status accounting**
  - record, report status
- **Audit and review**
  - validate completeness

Software Version control

- Many version control systems are designed and used especially for software engineering projects
  - examples: CVS (Concurrent version system), Subversion (SVN), **Git**, Monotone, BitKeeper, Perforce
- Helps teams to work together on code projects
  - a shared copy of all code files that all users can access
  - keeps current versions of all files, and backups of past versions
  - can see what files others have modified and view the changes
  - manages conflicts when multiple users modify the same file
  - not particular to source code; can be used for papers, photos, etc.
    - but often works best with plain text/code files

Repositories (Repo)

- **repository**: Central location storing a copy of all files.
  - **check in**: adding a new file to the repository
  - **check out**: downloading a file from the repo to edit it
    - you don't edit files directly in the repo; you edit a local **working copy**
    - once finished, the user checks in a new version of the file
  - **commit**: checking in a new version of a file(s) that were checked out
  - **revert**: undoing any changes to a file(s) that were checked out
  - **update**: downloading the latest versions of all files that have been recently committed by other users



Source: https://www.visualstudio.com/learn/set-up-a-git-repository/

Repository Location

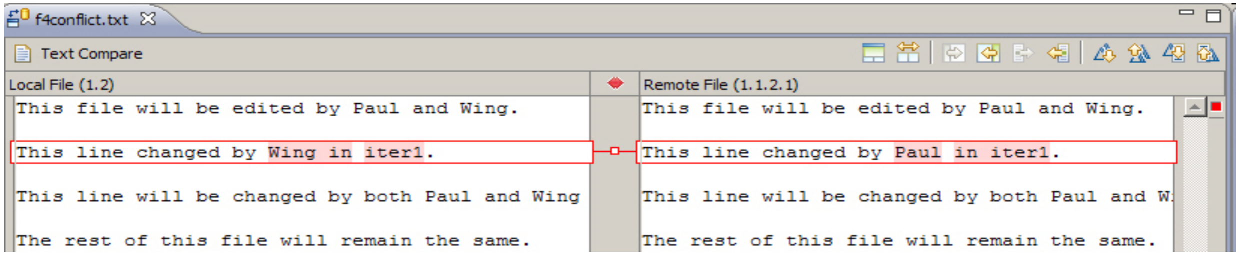
- Can create the repository anywhere
  - Can be on the same computer that you’re going to work on, which might be ok for a personal project where you just want rollback protection
- But, usually you want the repository to be robust:
  - On a computer that’s up and running 24/7
    - Everyone always has access to the project
  - On a computer that has a redundant file system (ie RAID)
    - No more worries about that hard disk crash wiping away your project!

What to put in a Repository?

- Everything needed to create your project:
  - Source code (Examples: .java, .c, .h, .cpp)
  - Build files (Makefile, build.xml)
  - Other resources needed to build your project: images, sound files, etc.
- Things generally NOT put in a repo (these can be easily **re-created** and just take up space):
  - Object files (.o)
  - Executables (.exe)
  - (Depending on Organizations: Security, etc)

Merging and conflicts

- **Merge**: Two sets of changes applied at same time to same files
  - happens when two users check out same file(s), both change it, and:
    - both commit, or
    - one changes it and commits; the other changes it and does an update
- **Conflict**: when the system is unable to reconcile merged changes
  - **Resolve**: user intervention to repair a conflict. Possible ways:
    - combining the changes manually in some way
    - selecting one change in favor of the other
    - reverting both changes (less likely)



Local File (1.2)

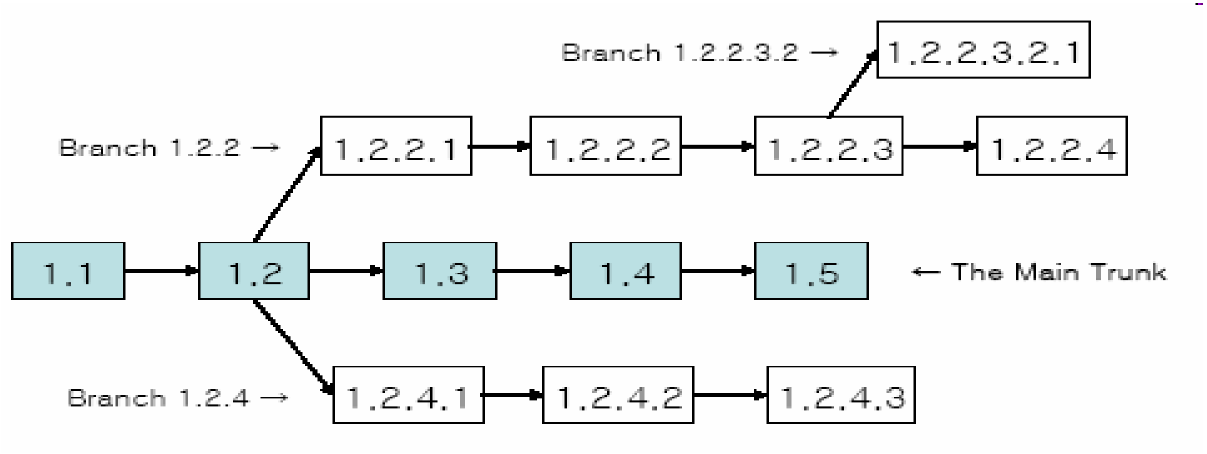
```
This file will be edited by Paul and Wing.

This line changed by Wing in iter1.

This line will be changed by both Paul and Wing

The rest of this file will remain the same.
```

Remote File (1.1.2.1)

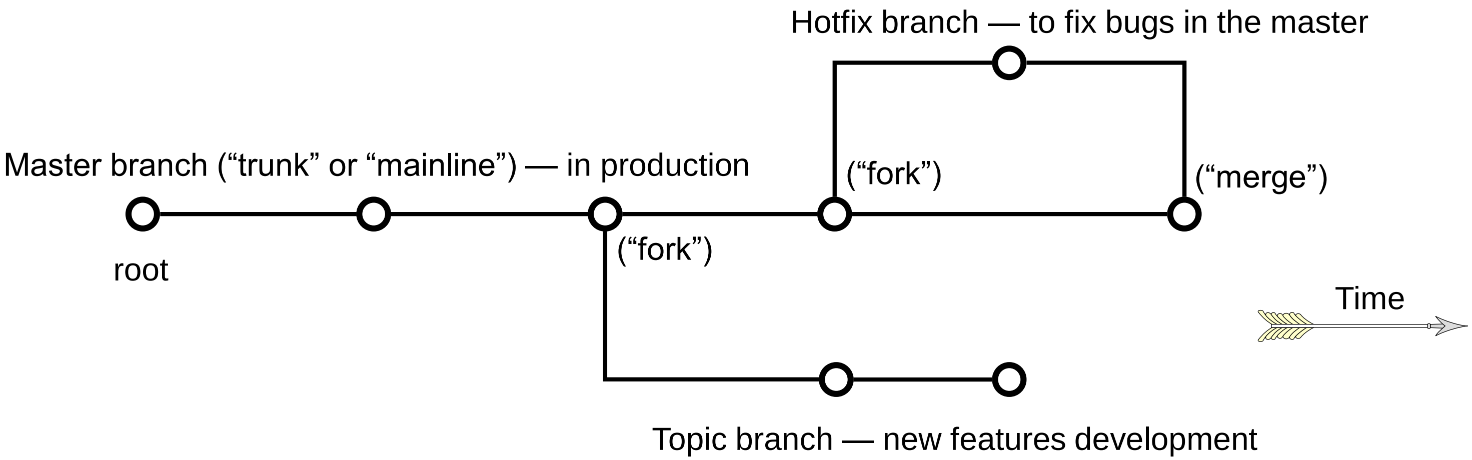
```
This file will be edited by Paul and Wing.

This line changed by Paul in iter1.

This line will be changed by both Paul and W

The rest of this file will remain the same.
```

Branches

- **branch** (fork): A second copy of **the files** in a repository
  - the two copies may be developed in different ways independently
  - given its own version number in the version control system
  - eventually be merged
  - **trunk** (mainline, baseline): the main code copy, not part of any fork

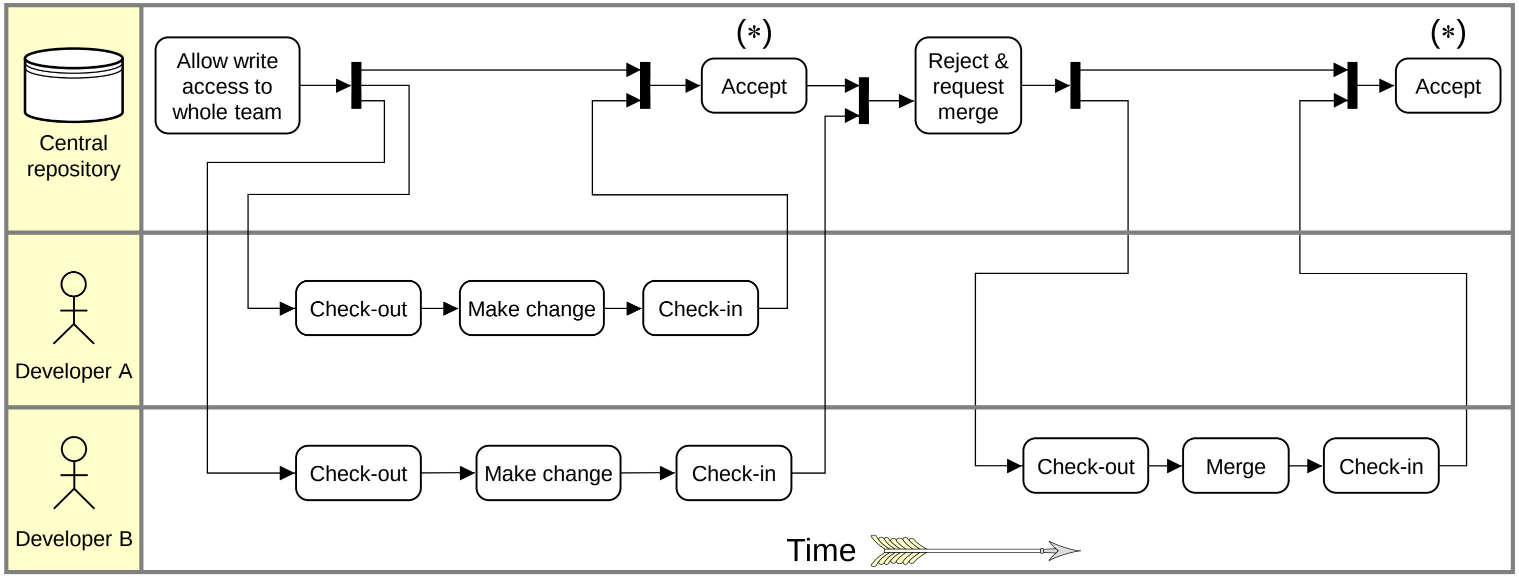


Version Graph and Branching


Hotfix branch — to fix bugs in the master
Master branch (“trunk” or “mainline”) — in production
(“fork”)
(“merge”)
root
(“fork”)
Time
Topic branch — new features development

- Each “commit” represents a different “version” of the software configuration at a different time
- Think of *branches* as separate folders, each with its own content and history
- The project snapshot at the tip of a branch represents the *latest version*

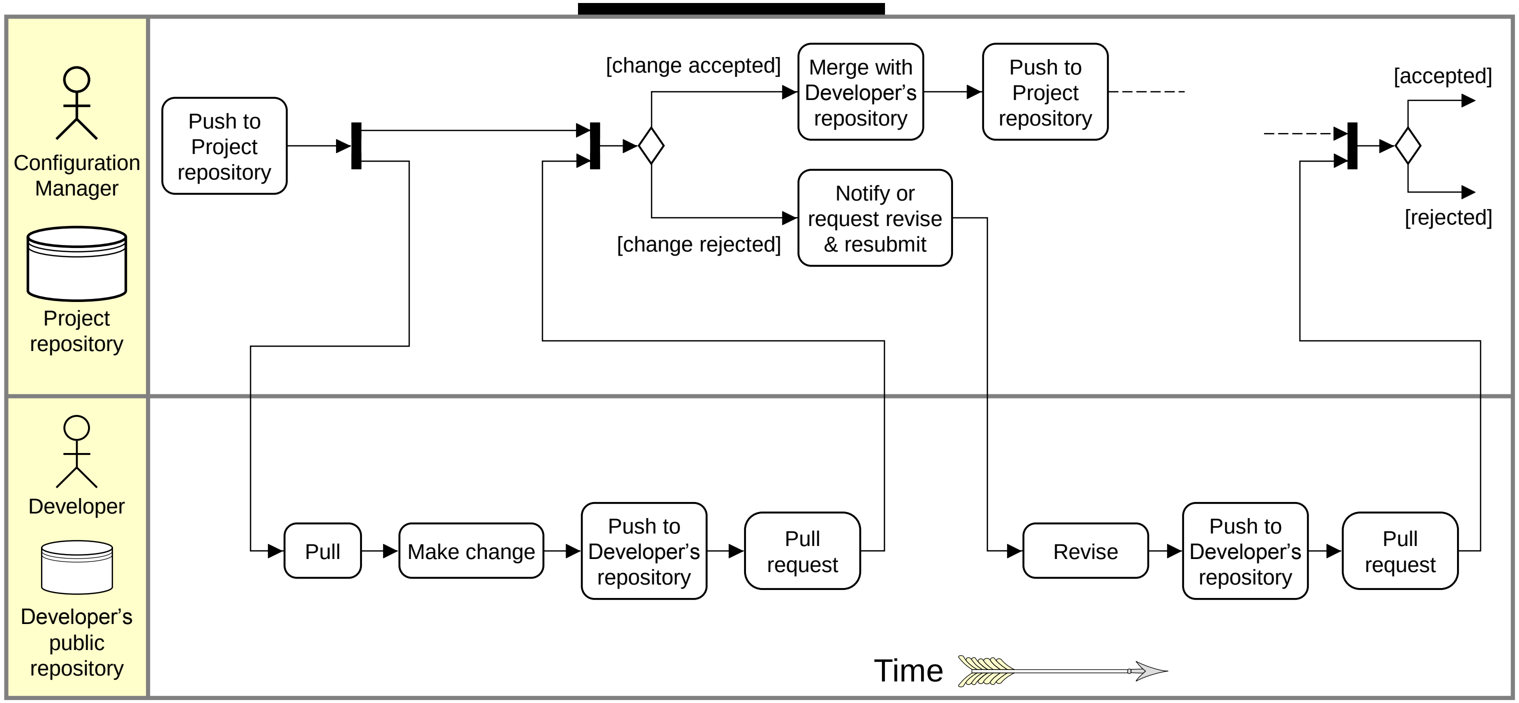
Working with Peers: Centralized Workflow



(∗) Assuming no other commits in the meantime; otherwise need to merge

- Example scenario: Two developers clone from the hub and both make changes
- The first developer to push his changes back up can do so with no problems
- The second developer must merge in the first one’s work before pushing changes up, so as not to overwrite the first developer’s changes

Working with a Managed Team


Configuration Manager
Project repository
Developer
Developer's public repository
Push to Project repository
[change accepted]
Merge with Developer's repository
Push to Project repository
[change rejected]
Notify or request revise & resubmit
[accepted]
[rejected]
Pull
Make change
Push to Developer's repository
Pull request
Revise
Push to Developer's repository
Pull request
Time

- Example scenario:
- 1. The configuration manager pushes the current version to the main project repository
- 2. A contributor clones that repository and makes changes
- 3. The contributor pushes the changed version to his own public repository
- 4. The contributor notifies the configuration manager requesting to pull changes
- 5. The configuration manager adds the contributor's repository as a remote and merges locally
- 6. The configuration manager pushes merged changes to the main project repository

Aside: So what is GitHub?

- GitHub.com is a site for online storage of Git repositories.
- Many open source projects use it, such as the Linux kernel.
- You can get free space for open source projects or you can pay for private projects.
- Do NOT use GitHub to store your homework!!

Question: Do I have to use GitHub to use Git?

Answer: No!

- you can use Git completely locally for your own purposes, or
- you could share a repo with users on the same file system as long everyone has the needed file permissions.

Git Resources

- At the command line: (where `<verb>` = config, add, commit, etc.)

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

- Free on-line book: https://git-scm.com/book/en/v2
- Git tutorial: http://schacon.github.com/git/gittutorial.html
- Reference page for Git: http://gitref.org/index.html
- Git website: http://git-scm.com/
- Git for Computer Scientists: http://eagain.net/articles/git-for-computer-scientists/

Popular Git commands

command	description
`git clone` ***`url [dir]`***	copy a git repository so you can add to it
`git add` ***`files`***	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help` ***`[command]`***	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init, reset, branch, checkout, merge, log, tag`	

Github Introduction (learning)

- Github Tutorial For Beginners
- https://www.youtube.com/watch?v=0fKg7e37bQE