1 Version Control

- Two flavours of Version Control
  - Centralized (B07 uses this)
  - Decentralized
- Centralised Version
  - Keep code in a centralized location (the "Repository")
  - Code in repository is the "Master Copy" (**<u>Never</u>** directly modify)
  - Instead make local copies of the repository on each computer you will be working on (working copy)
  - When major changes are made to local copy that you want to save, "commit" change to repo
  - Tools allow you to revert to a previous version of the source code (only when commits have occurred)
- Some Terminology
  - Repository/Repo
  - Working copy
  - Commit
  - Client program
  - Checkout
- Centralized systems include:
  - **S**ub**V**ersio**n** (**SVN**)
    * SVN is the successor to **C**oncurrent **V**ersions **S**ystem (**CVS**), and was built to help fix many issues in CVS
  - Git
  - Mercurial
  - ClearCase
  - Perforce
- SSH and SCP are **not** version control systems
  - **S**ecure **Sh**ell (SSH) is used to connect to a remote computer and work in a shell on that computer
  - **S**ecure **Cop**y (SCP) is used to:
    * Securely copy files from one computer to another
    * Transfer a copy of the files but does **not** version them
- Version Control – Managing Concurrency

  When two or more people want to edit the same file at the same time
  - Pessimistic concurrency
    * Only allow one writeable copy of each file
    * e.g. Microsoft Visual SourceSafe, Rational ClearCase
  - Optimistic concurrency
    * Allow writes, fix issues afterwards
    * Merging
      · SVN is either able to merge without help from the user, or
      · *Conflict*: SVN needs the user to resolve the conflict
    * e.g. Subversion, CVS, Perforce

- Optimistic Concurrency – Merging Options

  Select from: **(p)** postpone, **(df)** diff-full, **(e)** edit, **(mc)** mine-conflict, **(tc)** theirs-conflict and **(s)** show all options.

(e) edit	- changed merged file in an editor
(df) diff-full	- show all changes made to merged file4
(r) resolved	- accept merged version of file
(dc) display-conflict	- show all conflicts (ignoring merged version)
(mc) mine-conflict	- accept my version for all conflicts (same as above)
(tc) theirs-conflict	- accept their version for all conflicts (same as above)
(mf) mine-full	- accept my version of entire file (even non-conflicts)
(tf) theirs-full	- accept my version of entire file (same as above)
(p) postpone	- mark the conflict to be stored later
(l) launch	- launch external tool to resolve conflict
(s) show all	- show this list

- Integrating the code - Reasons for merge conflicts
  - Communication
  - Complex code bases
  - Experimental features being built
  - More than one project on the go that impacts this code
  - Two features being built in same class by different developers
- Branching
  - Branches are divergent copies of development lines
  - These versions are used to build out complex features, or do experiments, without having an impact on the main code line
  - Strategies include:
    * No branching
    * Release branching
    * Feature branching
- Storage scheme
  - Storing every copy of every file generated over the course of a project is not practical
  - Version control systems store incremental differences in files/folder structures
  - These differences store enough information to re-construct previous versions, without storing every single copy ever made of the file
- What's Stored Where
  - Server side: out of scope
  - Local copy contains a special directory, **.svn**

        * It stores (locally) the information subversion needs to keep track of your files, version numbers, where the repository is, etc.
        * Needless to say, you should not mess with the contents of this directory. Let subversion do its job

- General rules
    - Update and commit frequently
    - Never break the main branch
    - Always comment clearly what changes are in a revision
    - Test all code before accepting merge
    - Communicate with your team!