Welcome!

Virtual tutorial starts at 15:00 GMT

Please leave feedback afterwards at:
www.archer.ac.uk/training/feedback/online-course-feedback.php
Introduction to Version Control

ARCHER Virtual Tutorial, Wed 12th November 2014
Arno Proeme <aproeme@epcc.ed.ac.uk>
Reusing this material

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.
Outline

• What’s the problem?
  • Common suboptimal workflows in research

• Version control
  • Benefits
  • Basic concepts and terminology

• Popular version control systems (SVN, CVS, Git)
  • Practical demonstration
  • Capabilities, strengths and weaknesses
  • Tips, quirks
  • Deciding which system / tool to use
What’s the problem?

• You have an existing functioning script (Matlab, or R, or Python, or …) or software configuration files stored on your laptop – this is version 0

• You make some changes to introduce new functionality, but to a copy of the script because you don’t want to break anything and want to continue to be able to use the functioning version. This yields version 1

• You transfer this script to a different machine (e.g. ARCHER) but it then needs further modifications to make it work in the new environment, giving version 2

• You make some other changes to the original copy (version 0) on your laptop that implement the same functionality as version 1 but in a different, faster way - this is version 3

• Again version 3 needs to be modified to work on the remote machine, leading to version 4
What’s the problem?

• Having multiple versions of files means you need to manually keep track of how these are related – this becomes unmanageable and error prone.

• Hard to keep track of which file does what, which one is up to date and should be used for further development, etc.
What’s the problem?

• Working in a team: files (source code, article for publication, or documentation) are being edited by multiple authors

• Changes need to be shared amongst team members
  • Emailing files is a bad idea!

• Changes to the same file including in the same place need to be merged to create a final, approved version
What’s the problem?

• It should be clear who made which changes, and why

• File locking / ‘write’ token
  • Stops others working - inefficient
Solution: Version Control

• Version control systems
  • Partly automate
    • Keeping track of development history (changes to files, including by multiple authors)
    • Maintaining multiple versions (variants) of code
  • Provide a safety net (can always recover previous versions that we’ve chosen to record)
  • Facilitate
    • Reproducible research and open science
    • Testing and development
  • Can act as a backup
Basic concepts & terminology

Some concepts and terms are common to different version control systems:

• Repository
  • the complete archive of all versions of all files that were recorded, including how they are related and what change(s) led to each version.

• Working copy
  • the set of all files currently contained in the directory where you are working (making changes). This typically differs from a version in the repository only by some of your latest modifications to files.

• Log
  • a record of which files in the repository were changed when, including (hopefully) comments by the author who made the changes.
Basic concepts & terminology

• Commit
  • Committing a file or set of files to a repository means that the current state of these files in your working copy is recorded in the repository as a version (a commit).

• Branch
  • “Streams” of parallel development consisting of successive and related commits, typically in order to explore a particular direction of development such as a new feature / functionality.
Common version control systems

- Three common open-source version control systems:
  - CVS (Concurrent Versioning System)
    - mature and established, not as popular any more
  - SVN (Apache Subversion)
    - successor to CVS, widespread
    - more flexible and efficient, e.g. at handling non-text files
  - Git
    - newer, faster, powerful features, popular for many new software projects
Repository model - centralised

- CVS and SVN based on the notion of a canonical ‘master’ repository containing the most complete, up-to-date versions of files, typically stored on a central server.

- Authors check out a working copy of the repository to their local machine, make changes, and (attempt to) upload these changes to the server and commit them to the repository as a new version.
Repository model - distributed

• Git and Mercurial based on local repositories

• No assumption of a canonical repository

• Synchronising with a remote repository is optional
  • no need for a server
Practical Demonstration (see recording)
Which VCS tool should I use?

• If joining an existing project:
  • whatever is already being used!

• For your own development work:
  • SVN or Mercurial
  • Git is very powerful, but has a steep learning curve – Mercurial is simpler
Useful Links

- http://svnbook.red-bean.com/
- http://github.com
Goodbye!

Thanks for attending

Please leave feedback at:
www.archer.ac.uk/training/feedback/online-course-feedback.php