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 (Part 1)

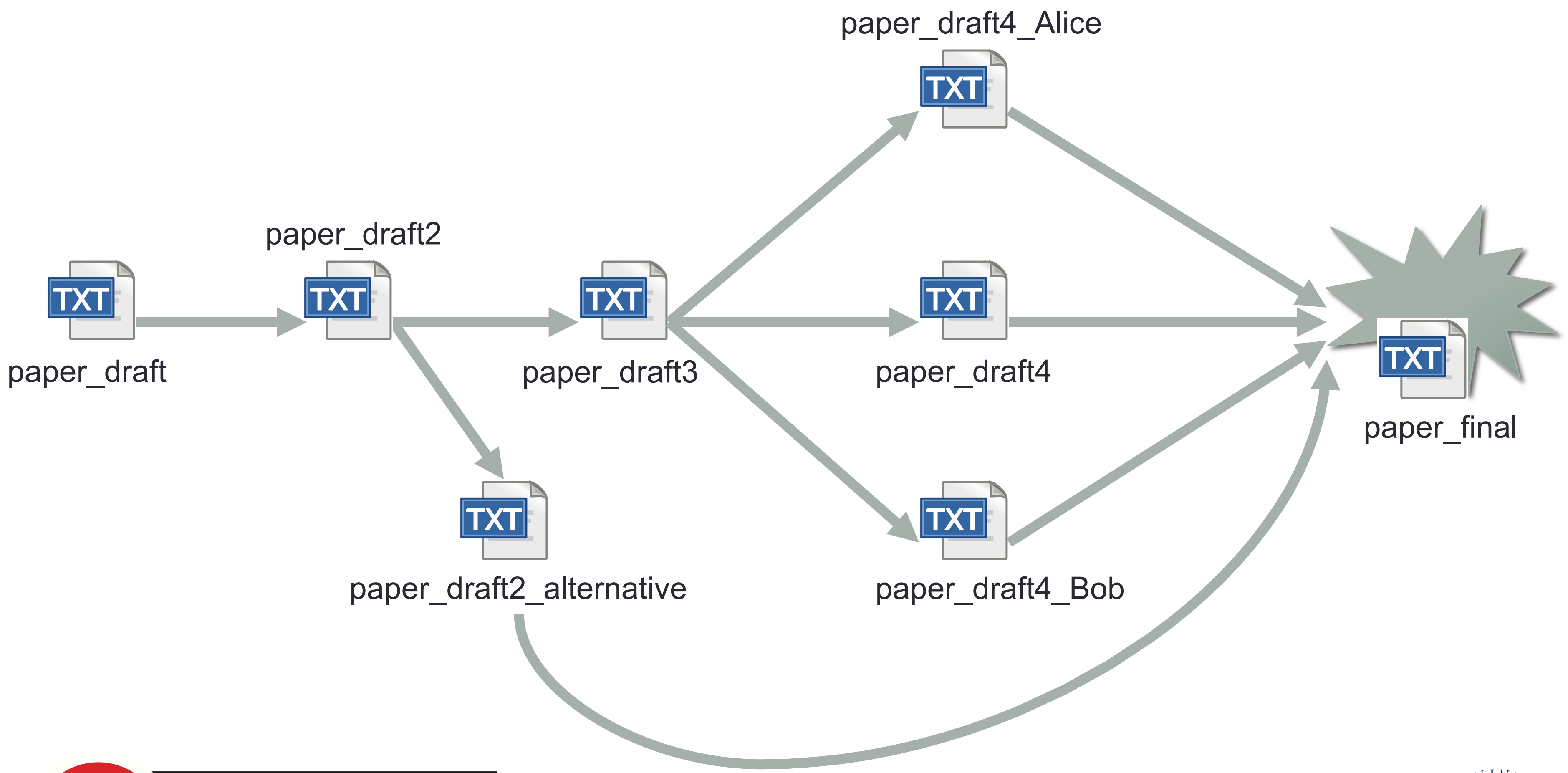
ARCHER Virtual Tutorial

Reusing this material

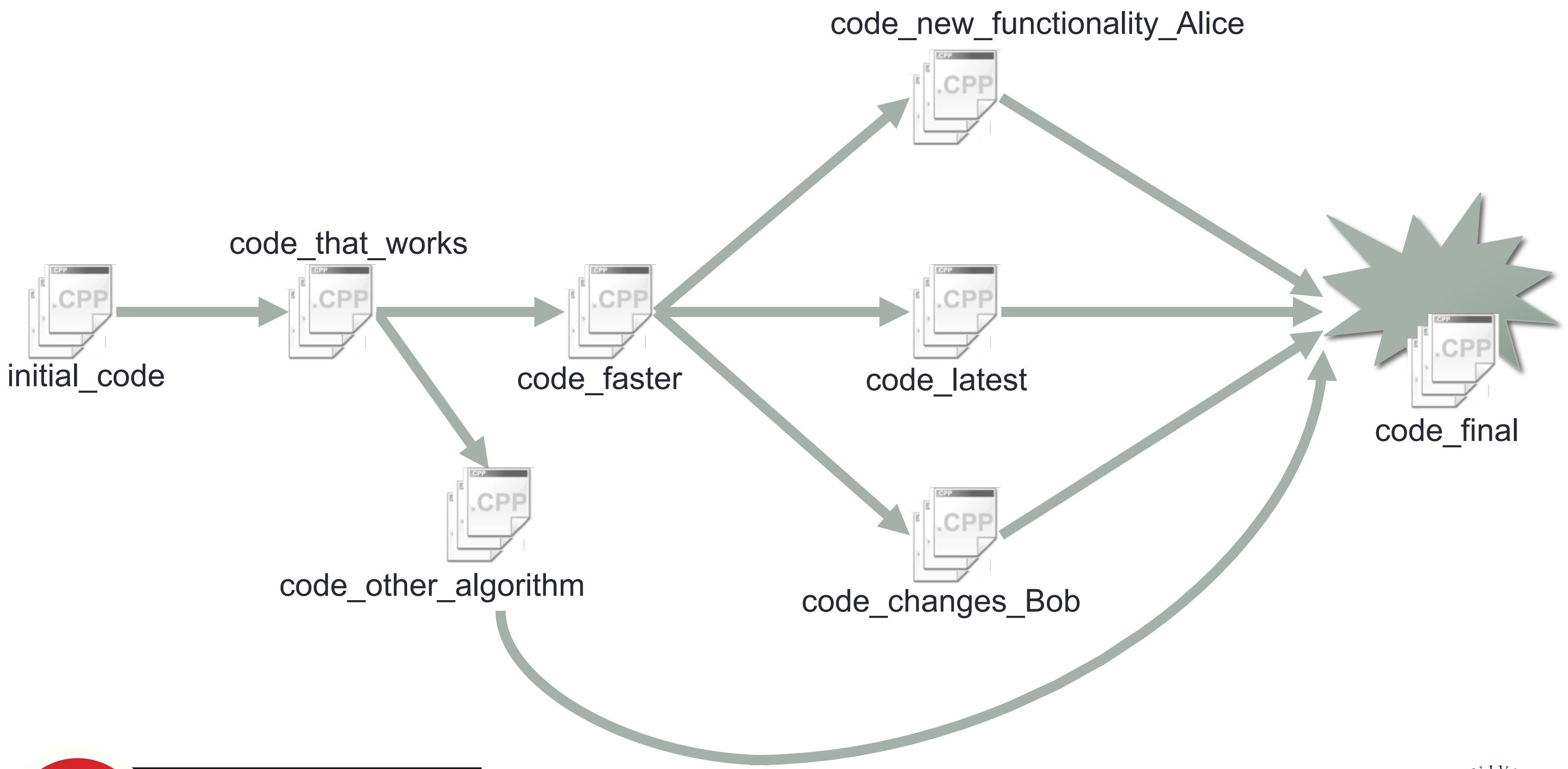
Outline

- Version Control – do it yourself?
- Version Control Systems
  - Benefits
  - Common version control systems
  - Core concepts and terminology
- Simple demonstration using SVN
- Word of warning

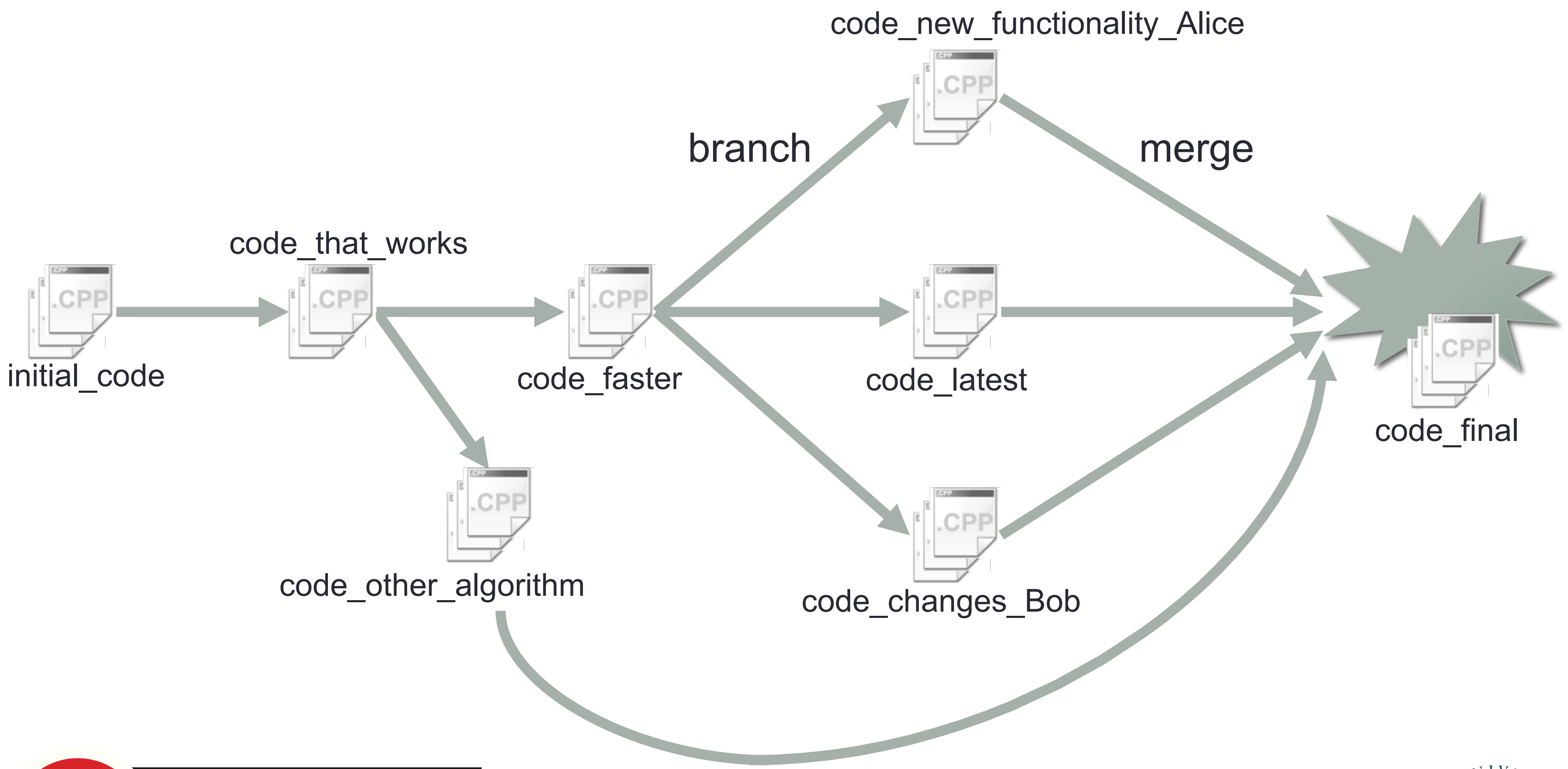
Version Control – do it yourself?



Version Control – do it yourself?



Version Control – do it yourself?



What's the problem?

- Forced to manually keep track of
  - The differences between multiple versions of a file
  - How multiple versions of a file are related (e.g. through branching & merging)
  - How versions of different files are related (e.g. code dependencies)
  - Which versions of which files should be used (together) as a basis for further work

Do we record this information in filenames and directory structure?* Inside the files themselves? Elsewhere?

*http://www.phdcomics.com/comics/archive.php?comicid=1323

What's the problem?

- Forced to merge versions manually:
  - To produce a version that meets specific requirements by combining content from multiple versions
  - To combine changes made due to editing copies of a file in different locations (e.g. personal laptop & work desktop)
  - To combine changes made by multiple authors who have each added useful content to their own copies of a file

- Again need to keep track of merge results for further work

What’s the problem?

Do-it-yourself version control:

- Time consuming
- Requires constant care and attention
- Prone to human error
- Unmanageable for many files / many versions
- Everybody has their own system
  - Difficult to collaborate

Version Control Systems

Focus on version control systems developed and optimised for plain text files:

- Simple text documents
- Any file containing human-readable markup - **source code**
- Numerical data formatted as plain text (e.g. .csv files)

Less useful for managing non-plain-text (i.e. binary) data:

- Documents encoded in a binary format (e.g. PDFs, MS Office)
- Executables
- Images & video
- Numerical data stored in a binary format (e.g. HDF5, netCDF)

Version Control Systems

Version control systems are software tools that:

- Provide a framework to record meaningful information about file versions in a consistent, systematic way
- Help automate the tracking of versions and the differences between them by recording the state of a set of files at a given time as a snapshot and providing easy access to these snapshots

Version Control Systems

Version control systems are software tools that:

- Provide a safety net whilst making changes (can recover previous versions of snapshotted files)
- Capture and preserve dependencies between particular versions of files, e.g. source code

Version Control Systems

Version control systems are software tools that:

- Allow for easy duplication and synchronisation of files in multiple locations
  - Avoids error-prone manual transferring of files
  - Can act as a backup of your data
  - Easily work on different machines

- Enable collaborative work on same set of files at the same time, automatically identifying contributions from different authors

Version Control Systems

Automatic change tracking facilitates **branching**:

- Modify one or more files with a particular goal in mind (e.g. new feature, bug fix) by creating a new branch
- Do this for multiple goals independently and in parallel (e.g. by different authors) by creating multiple branches

- At a later date we can combine differently-modified versions of the same files by **merging** them

- Allows us to pick and choose changes developed in isolation on different branches and integrate them as desired

Version Control Systems

We can use version information to enable

- Reproducible computational research
  - Report exactly which version of code produced published results
- Testing and development work
  - Track which version of code works, runs faster, etc.

Version Control Systems

Can access and use version control tools

- From the command line in a shell session
  - Common version control tools installed by default in Linux and OS X
- Using a standalone client application with a graphical user interface
- Through a web-based interface

Common 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 than CVS, e.g. at handling binary files
- Git
  - Newer, faster, powerful features, very popular for many new software projects thanks partly to GitHub website
- Mercurial
  - Like Git but simpler in some ways to use

Core concepts & terminology

Concepts and terms common to many version control systems:

Repository

Log

Working copy / working directory

Check out / clone

Merge

Update

Commit / check in

Branch

Core concepts & terminology

Concepts and terms common to many version control systems:

Repository

- Archive of all recorded snapshots of file versions
- Captures the changes between successive recorded versions of a file
- Keeps track if versions of a file are related through merging or branching
- Includes a log

Log

- Metadata describing *when*, *by who*, and optionally *why* each snapshot was recorded

Core concepts & terminology

Working copy

- Your local copy of (some of) the files in the repository
- Located on the machine you're currently using regardless of where the repository itself is stored
- Shows your current local versions of files
- Your versions differ from the latest versions in the repository:
    - if you have made changes, or
    - if somebody has updated the repository with newer versions

Core concepts & terminology

Working copy

- May contain files that are not yet recorded in the repository
- Unrecorded files and changes to existing files are not automatically propagated to the repository – this needs to be done explicitly
- Can have multiple working copies, e.g. on multiple machines

Core concepts & terminology

Check out / clone

- Obtain an initial working copy by duplicating (part of) a repository locally on your machine

- **Merge**
  - Combine two versions of a file or set of files into one
  - Can lead to conflicts
  - Version control systems will point out conflicts but *you* need to think and decide how to resolve these

Core concepts & terminology

Update

- Update your working copy with the latest snapshot in the repository
- Attempts to merge the latest versions of files in the repository into the corresponding files in your working copy
- Can lead to merge conflicts

Commit / check in

- Take a snapshot of the current state of one or more files in your working copy and record it in the repository.
- Transfers the following data from your working copy to the repository:
  - Changes you made to these files since the last time they were synchronised with the repository
  - A message commenting on these changes (the commit message)

Core concepts & terminology

- **Branch**
  - Create logical copies of one or more files in the repository
  - Typically done to pursue a particular direction of work such as, in software, a new feature / functionality
  - Newly spawned copy versions are tracked automatically as a distinct set and synchronised via commits and updates independently of the original parent files
  - File versions on one branch can be integrated with versions of the sames files on another branch through merging

Commit messages…

- Shown in the log
- Comments meant to inform use of the repository by
  - Yourself in future (hours, days, weeks, months, years from now)
  - Current and future collaborators
- Should be a meaningful summary explaining the reason for the commit, giving appropriate level of context / detail
- Typical format is
  - One line summary
  - Further details
- Avoid meaningless messages:
  - http://xkcd.com/1296/

Where do repositories live?

Repositories can live

- on a publicly hosted website (e.g. Bitbucket, GitHub)
- on a server at your institution
- on your own machine

More about this in Part II

Basic demonstration using SVN

- Going to:
  - Check out part of an existing repository
  - Inspect the log
  - Compare past changes
  - Make a change to a file and commit this new version to the repository
  - Create a new file and commit it
  - Delete the new file from the repository
  - Undo the change to the first file

Word of warning

- Version control systems are a powerful tool, not a magic bullet
- You need to think and decide how to manage your work
- When working collaboratively, need to communicate

Scripted practical & next tutorial

- A scripted practical using SVN will appear on the ARCHER website before the start of the second virtual tutorial on version control.
- This will help you put into practice the concepts from this presentation.
- In Part II we will
  - Explore differences between centralised and distributed models of version control and local and remote repositories
  - Demonstrate the basics of Git and how it compares e.g. to SVN
  - Consider which version control system you may want to use