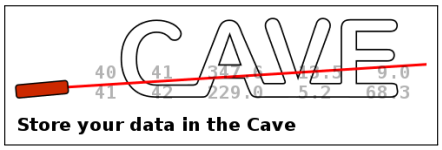


The Cave Archive and Versioning Experiment

How to use the CAVE to Manage your Club's Survey Data

Authors: Michael Lake, Philip Maynard
Suggestions and corrections should be emailed to: Philip.Maynard@uts.edu.au

Date: 2015-09-22 Revision: b60225131793

Contents

1 Introduction

The Cave Archive and Versioning Experiment (The SUSS Cave[1]) is a set of software programs designed to enable uploading, downloading, browsing, searching and archiving of data. It is based on a version control system, which allows more than one person to have updating access to project files and also provides a complete history of the files in the system. It is possible for a person who has updating access to gain access to every historical version of a file.

The publically accessable part of the archive is browsable by everyone through a website (see the next section), while people who have been given access to the private part of the archive can modify files, add files, remove files (but not historical versions of files) and add information ("metadata" [2]) about files (see the editing sections of the guide).

This how-to Guide explains how to use the SUSS Cave to manage your club's data. The initial focus of this system is on archival storage of cave mapping projects, and some of the information in the guide is specific to map data and map file formats. The system is also useable for more general information from your club, such as Minutes, articles, letters and similar.

This archive system relies on the Mercurial version control system. To provide a graphical user interface to Mercurial, the GUI package EasyMercurial can be used. These are both free software packages available for download. There are versions available for all personal computer operating systems but not (at this stage) for tablet operating systems. For details on installing these software packages, please refer to the Installation guide in the Appendix.

2 Current Repositories

This is a list of just some of the current SUSS repositories. The complete list can be found at http://suss.caves.org.au/cave/

Cave Area	**Public URL**
Cooleman Caves	http://suss.caves.org.au/cave/cooleman
Jenolan Caves	http://suss.caves.org.au/cave/jenolan
Wombeyan Caves	http://suss.caves.org.au/cave/wombeyan
Playground Caves	http://suss.caves.org.au/cave/playground

"Playground Caves" is a fictitious cave area for SUSS users to practice using the Cave software. It will be used extensively in this guide.

The actual mercurial repository is accessed via URLs like the examples below. A random string of numbers and upper and lower case letters is used to ensure that the actual URL for the mercurial repository cannot be guessed. Additionally these repo URLs require a username and password to login into them. Note that the random strings shown here are not the real ones. These are examples only.

Cave Area	**Private Repository URL**
Cooleman Caves	http://suss.caves.org.au/cavehg/To0dh9C7eks7Q/cooleman
Jenolan Caves	http://suss.caves.org.au/cavehg/n6g5Hu3cI1slA/jenolan
Wombeyan Caves	http://suss.caves.org.au/cavehg/uvT1SydHcWhu2/wombeyan
Playground Caves	http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground

[1] See recursive acronym http://en.wikipedia.org/wiki/Recursive_acronym
[2] Metadata is data that describes data.

3 Files you can Store in the Archive

How Files are Stored in the Archive

Because the repository stores all versions of a file in the repository we should ensure that the files that we add can be stored efficiently so that the repository does not grow too large. In this case efficiency is the ability of the version control system (Mercurial) to create "diffs" between one version of a file and the next version i.e. just the parts of the file that have been changed. Version control systems in general store the first copy of a file as the full file, and subsequent versions are stored as a "diff" against the preceeding version. Conceptually you can view it like this:

Version No.	What is stored
1	file1
2	diff1 (file2 - file1)
3	diff2 (file3 - file2)
4	diff3 (file4 - file3)

So to recall version 3 of a file the system takes the first version and applies diff1 and diff2 to obtain version 3. Diffs between versions of a file that is plain text are usually small so checks in a newer version of the file only small diffs are added to the repository. The repository size only grows slowly over time. However versions of files such as images may differ in most of their regions as each pixel may have been changed slightly. For instance, if you scale an image or alter its color levels every single pixel will be changed and a diff of this compared to its previous version will be nearly as large as the original file. Hence multiple versions of images (and generally most binary files) cannot be stored as efficiently as text files in version control systems. Checking in newer versions of binary files causes the repository size to grow faster.

Types of Files that can be Stored

Survey data files: Survey data formats that can be stored efficiently in the repository are Survex (.svx), Compass (.dat and .plt) and Therion survey files (.th) as these are plain text files. When you add another version of these types of files the repository only grows larger by as much as the difference between the versions, which is usually very little. Any other plain text data files are suitable.

Scanned images of field data and sketches: These would be JPG and PNG format images. These are compressed formats but can't be stored as efficiently as plain text files. Every time you add a version of an image the repository grows larger. Hence its best to only scan pages at 150 dpi and to be diligent in cropping blank areas before you commit the file. Do not use TIF files as they are too large.

Cave map working files: You are probably using a vector drawing package for this task. This might be Therion, Inkscape or Adobe Illustrator. Therion map files (.th2) are all plain text. If using Inkscape save as unzipped SVG. For Illustrator files see section "Tips for Specific Software".

Final cave maps: The format for final cave maps would be PDF. If the cave map is really final then the file can be stored in the repository as it will not be re-edited again.

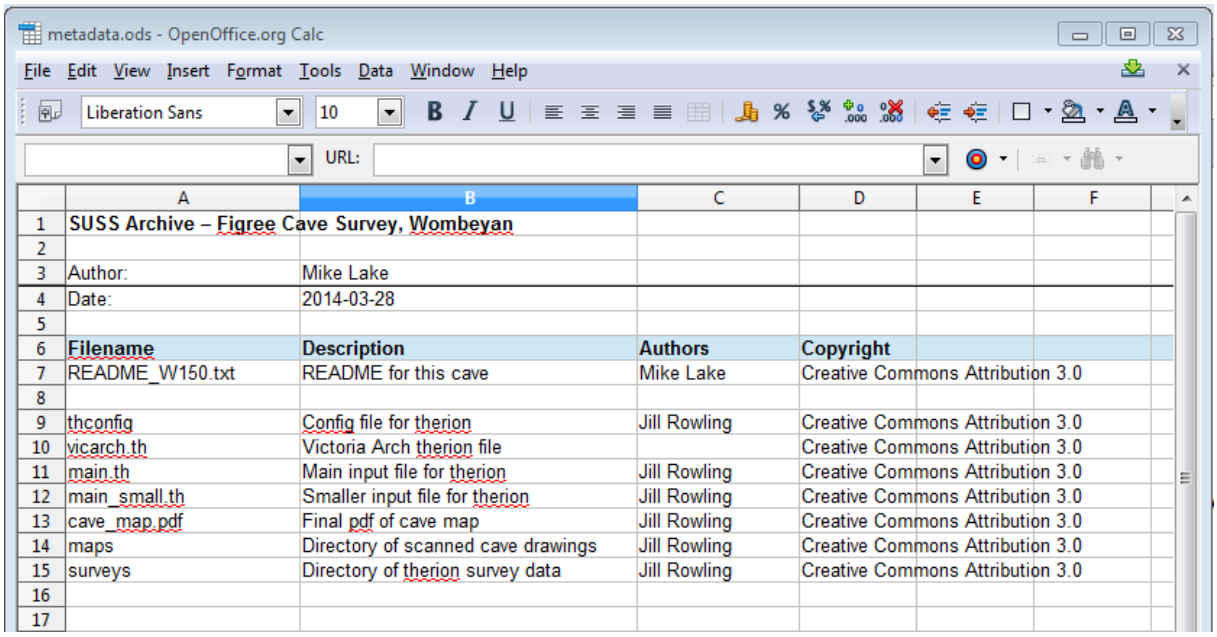
Word Precessor and Spreadsheet Files: Microsoft Office files cannot be stored efficiently. However, Open Office and Libre Office spreadsheet files can be stored efficiently if they are saved in the unzipped XML format (or if the Mercurial extension to unzip them is installed).

4 Metadata for Files and Directories

Files and folders submitted to the archive need to have some *metadata* uploaded with them that describes what each file actually is and perhaps some other fields to fully describe the contents.

In every branch you need to maintain a spreadsheet that lists each file and directory and describes them. When you Pull an existing branch of a repository, the spreadsheet file should be included in the list of files. The spreadsheet must be named `metadata.ods` and be in Open Document Format. Spreadsheets in the Open Document Format have extension `.ods`. The open document format can be created and edited using all popular spreadsheet programs.

The basic format of the required spreadsheet is a header row of allowed metadata fields such as Filename, Description, and possibly others and then rows listing the filenames and any directories and their metadata under each heading. When you Add a file to the branch, you need to add a row in the `metadata.ods` spreadsheet for that file. Once you have edited the `metadata.ods` spreadsheet file, you Commit it and Push as for other files in the branch.



	A	B	C	D	E	F
1	**SUSS Archive – Figree Cave Survey, Wombeyan**					
2						
3	Author:	Mike Lake				
4	Date:	2014-03-28				
5						
6	**Filename**	**Description**	**Authors**	**Copyright**		
7	README_W150.txt	README for this cave	Mike Lake	Creative Commons Attribution 3.0		
8						
9	thconfig	Config file for therion	Jill Rowling	Creative Commons Attribution 3.0		
10	vicarch.th	Victoria Arch therion file		Creative Commons Attribution 3.0		
11	main.th	Main input file for therion	Jill Rowling	Creative Commons Attribution 3.0		
12	main_small.th	Smaller input file for therion	Jill Rowling	Creative Commons Attribution 3.0		
13	cave_map.pdf	Final pdf of cave map	Jill Rowling	Creative Commons Attribution 3.0		
14	maps	Directory of scanned cave drawings	Jill Rowling	Creative Commons Attribution 3.0		
15	surveys	Directory of therion survey data	Jill Rowling	Creative Commons Attribution 3.0		
16						
17						

The metadata.ods file for Figtree Cave, W150

The `metadata.ods` spreadsheet is the basis for the browsable web page of the branch (See Section 2 above). Once per day a program runs on the ASF server and reads a file which specifies what metadata fields are allowable in the spreadsheet. This file is named `metadata_definitions.ini` and is controlled by the club administrator of the archive. The program goes through every repository and looks for a file named `metadata.ods` which should be present in every branch. The program checks that each spreadsheet file contains mandatory metadata fields, and any optional metadata fields if they are allowable. It checks that all filenames listed in the spreadsheet file for a branch can be found in that branch and that all files in the branch are listed in the spreadsheet file. If OK it then creates a web page with the data from the `metadata.ods` file. This is the page that appears when you view the browsable archive using a web browser.

Metadata Fields

Within the default branch of a repository is the `metadata_definitions.ini` file. This contains the names and definitions of all of the data fields that you can use to describe your data. Note that files

in the default branch can only be changed by the club administrator of your archive.

Metadata fields defined by the definitions file will be classed as [Mandatory] or [Optional]. Mandatory fields need to be filled out for every file that you add to the `metadata.ods` spreadsheet. Optional fields can be used for files you add to the `metadata.ods` spreadsheet if you wish.

Example Metadata Definitions File

Below is an example of a metadata definition file. The actual one that you will be using for a specific archive will be found in the "default" branch of that repository.

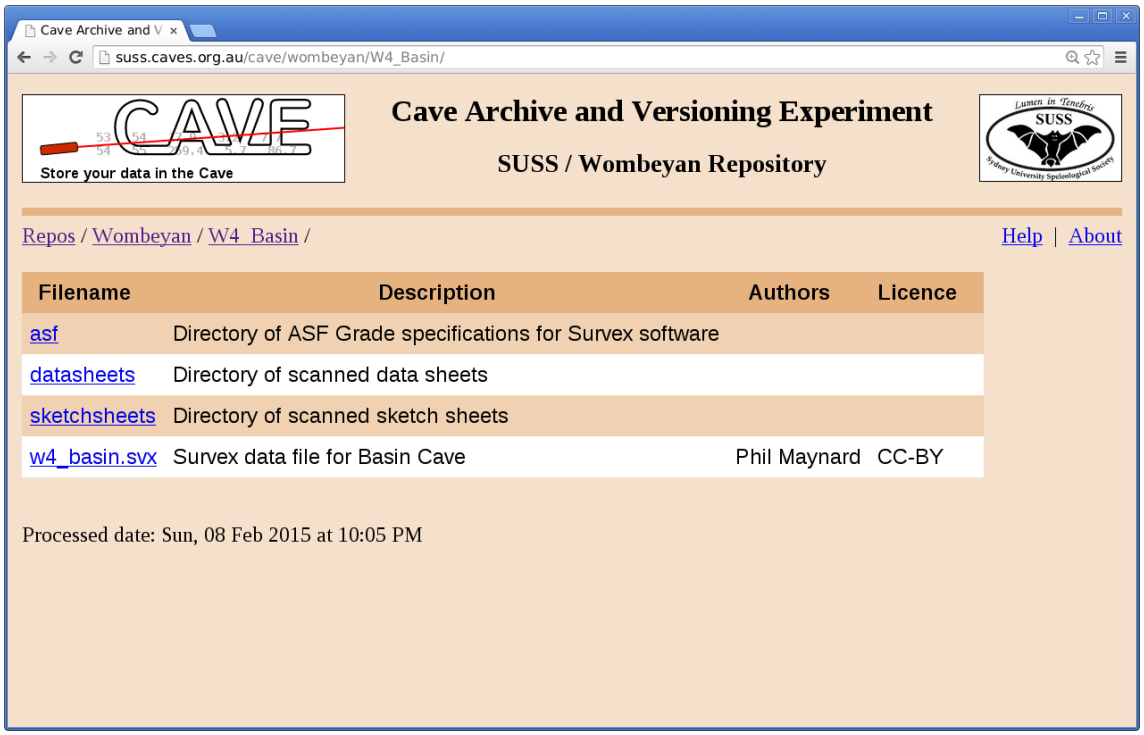
```
# This file specifies the allowable metadata fields.
# It is in INI file format (ref: http://en.wikipedia.org/wiki/INI_file)

# Fieldname must be single word, "foo" or "foo_bar"
# Definition must be a single paragraph.
#
# Format must be like this:
# fieldname: definition
# blank line
# fieldname: definition
# etc

[Mandatory]

filename: The file name as uploaded.

description: A text description of the document. One or two sentences should suffice.

[Optional]

authors: Anyone who contributed to the content of the file.

copyright_owner:  By default, this is the Lead Author unless rights
                  have been transferred.

license: The license under which this file is made available
         e.g. Creative Commons, public domain etc.
```

Adding New Metadata Fields

To add a new field contact the club administrator of your archive. They will edit the default branch of your repository, changing the `metadata_definitions.ini` file to add the field name and its definition. They will then commit and push the change to the ASF server. Once that is done you can use that new field in any of your `metadata.ods` spreadsheets within that repository.

5 The Browsable Archive – File Display

Each set of files that you place in the archive will be displayed on a public web page. This web page will display a list of file names, as shown below (an example of a single cave mapping project – Basin Cave at Wombeyan):



If you have decided to provide public access to these files, each file name in the list will be a link and clicking the link will view the file.

If you have decided not to provide public access to these files, the file names will not be a link and there is no way to access the files from the website. See the "Copyright and Access Control" section for details.

The web page provides a navigation system, underneath the title and logo. You can click on this to move to other caves, cave areas, or other archives set up by your club. The list of files can also contain folders with more data (eg, "surveys" in the web page shown above). Clicking on a folder opens the list of files in the folder. The list also contains other information; in the example above, the information available is the Description of the file, the author/s and the organisation the information originated from.

6 The Editable Archive – Version Control System

If you have been granted access to edit the archive, then you gain access and modify the archive through a *version control system*. This is a program that monitors and tracks files and records changes to them.

The defining property of a version control system is the ability for multiple people to work on a project simultaneously and collaboratively. Each person receives an up-to-date version of files when they download from the central server, and the version of the files uploaded by each person becomes the most recent and valid version of the file.

The second core feature of a version control system is that historical versions of files are never lost. Any previous version of a file which has been added to the archive is accessible to people who have been given access (but only the most recent is displayed on the web page to browsers).

Everyone with access can download and work on the files. *What happens if someone else edited the files at the same time as you did?* The version control system will alert you if any of your files have been edited since you downloaded them. You conduct a Merge on the files to resolve the differences (See the Merging section below).

Repositories and Branches

Repositories are the basic storage area of the archive. They have been defined in this system to organise the files by area – each caving area is a separate repository.

Branches are the main working divisions within a repository. They have been defined in this system to organise files by cave – each cave is a separate branch. Surface data, GIS information, etc will be in a separate branch.

When you download files from the archive (See the Workflow section below on how this works), you will receive the branch you requested, with the most recent version of all of the files. If you already have versions of these files in your Working Directory (see below), you'll receive updates of any files that have been updated to more recent versions than what you have.

Your Working Directory

The *working directory* is the location on your computer/device where you want to work on files for the repository. This is where the files will land when you download files from the ASF server. If you have files of your own that you want to place into the version control system, you copy them into your working directory (see below).

Workflow

The basic workflow that you use for working with the archive will be the following steps:

1. *Pull.* This downloads the current version of the branch(s) from the ASF server.
2. *Update.* This updates files in the working directory to the currently-selected branch and also to the most recent version in the history of the branch.
3. *Edit.* You can edit existing files, add new files to the branch, and remove files from the branch.
4. *Commit.* This places your changes into the version control system. New files come under version control at this point. Edited files will be updated in the branch to the latest saved state. Removed files will be taken out of version control at this point. Important: this only affects

versions of files in your working directory. At this stage, nothing has been uploaded to the ASF server and other users cannot see your Committed changes.

5. *Push.* This sends your Committed changes to the ASF server, and updates all files on the server to the version you Committed.

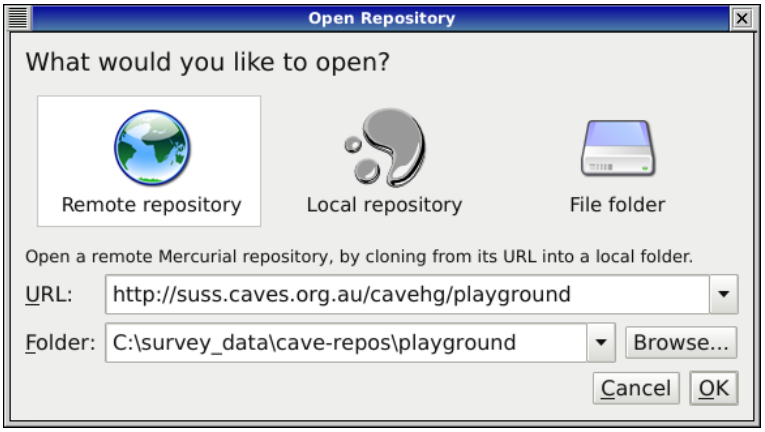
7 Working with an Existing Branch

Getting Access to the Remote Repository

The remote repositories are located on the ASF server. To gain access to the repositories, you will need a user name and password (ask your friendly club archive administrator). It is suggested that you set up a working directory on your computer for each repository you want to work with (see "Best Practise" section).

Pulling and Updating Files

Start the EasyMercurial application. Click the 'Open' icon. In the Open Repository dialogue box, click on 'Remote repository'. This asks for two addresses. The first is the remote server location ('URL'), and the second is your working directory ('Folder'). A drop down list in the URL address box allows you to look at previous addresses you have used. You can find your working directory by clicking 'Browse'. Enter the two addresses and click OK. You will be asked for a user name and password to gain access to the files on the ASF server.



Enter the remote repository source and the folder where your local repository will be created

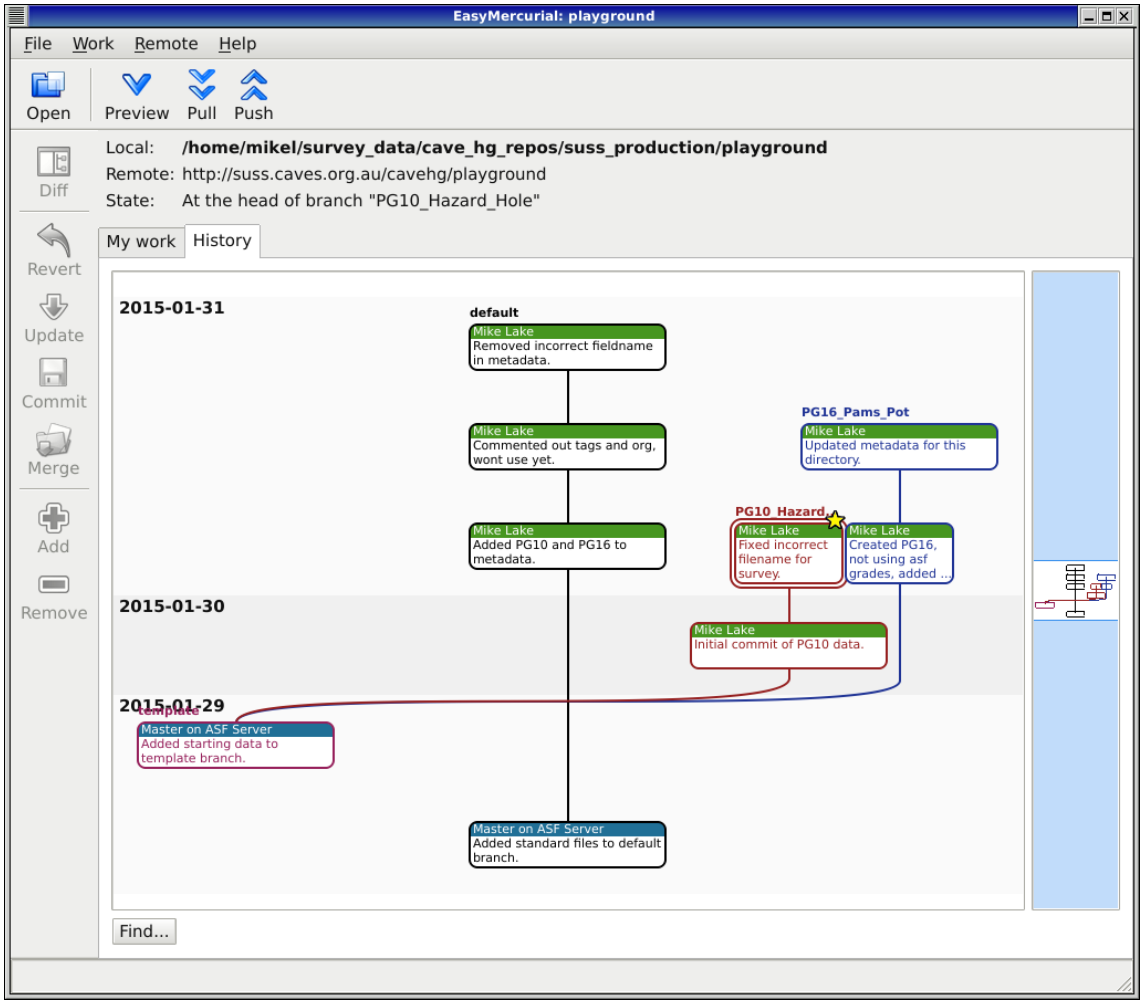
After the first time you have downloaded the remote repository to your working directory, you don't need to use 'Open / Remote repository' again. From the Open dialogue box, click on 'Local repository', and browse to your working directory, or select from a drop-down list. Click OK to open the repository in your working directory.

After the first time, to gain access to the remote server, click on the 'Pull' button at the top of the main EasyHg window. This will open a dialogue box with the address of the remote server. Click on Pull to confirm this. You will be asked for a user name and password to gain access to the files on the ASF server. If you have been given access, the remote archive will be downloaded into your working directory.

Once the files are downloaded, you should click on the 'Update' button on the left side of the main EasyHg window. This ensures that all of the files in your working directory are the most recent version.

Your Working Directory

Once you have Updated your working directory, your EasyMercurial window will look something like this:



The “History” tab

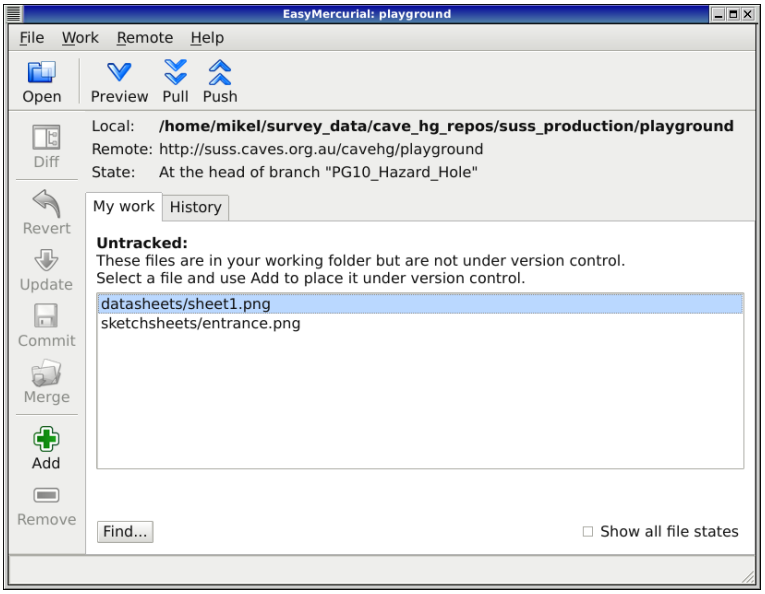
There are two tabs which provide you with information about the repository and the branches. These are the ‘My work’ tab and the ‘History’ tab. The History tab shown above contains a number of elements. In the main part of the window, the branches of the repository are shown graphically. Each branch is a vertical line. Each box on a branch is one Committed set of changes to the branch. The current set of files in your working directory is indicated by a star on the box. On the right hand side is a graphical tree showing the entire history of the repository. You can browse this to get at old versions of any branch. At the top of the window the address of your working directory is shown as ‘Local:’ and you can click on this to open the working directory. Underneath this is the address of the remote repository, and the ‘State:’ of the working directory; which is the current set of files you have in your working directory. This is also shown by the star in the main window.

You can change branches, and carry out various other tasks, by right-clicking on a particular box in a branch. In the example above, to change the current branch in the working directory from ‘Hazard Hole’ to ‘Pams Pot’, you scroll to the top of the Pams Pot branch and right click on the latest version of that branch (the highest box on the branch). From the drop-down menu, select ‘Update to this revision’. The relevant files will appear in your working directory, and the Pams Pot branch will now have a star at the head of the branch to indicate that it is the currently active branch.

The ‘My work’ tab shows the editing status of files in your working directory. When you have just

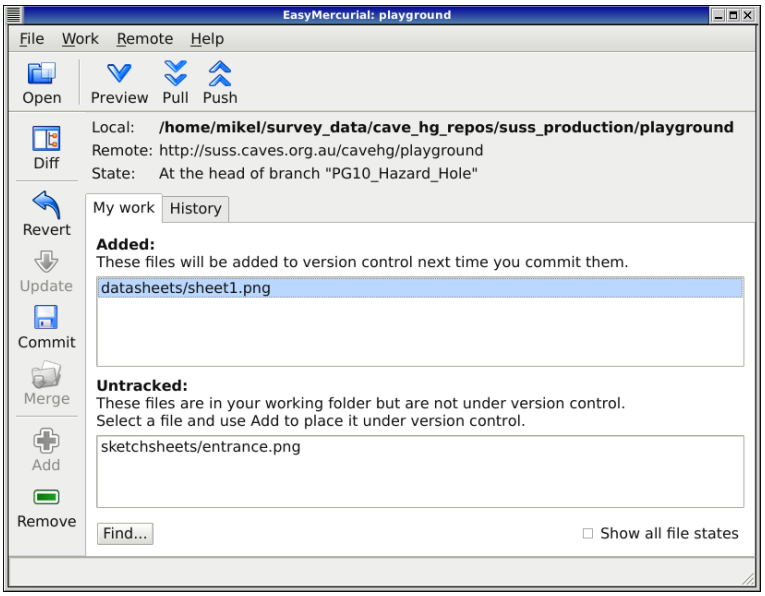
Updated your working directory, the My work tab will show essentially nothing, and have the comment "At the moment, you have no uncommitted changes". At the top of the window the address of your working directory is shown as 'Local:' and you can click on this to open the working directory. Underneath this is the address of the remote repository, and the 'State:' of the working directory; which is the current set of files you have in your working directory.

Once you have edited files in the working directory, or added files to the working directory, the files will be listed in the My work tab as shown in the example below. The first box ('Modified') shows files which are already part of the archive but which have new edits. These files are ready to Commit to the archive. The second box ('Untracked') shows files which are in the working directory but which are not currently part of the archive.



The "My work" tab

The first thing you should do here is bring any new files which you are adding to the archive under version control. To do this select the file in the Untracked box and then click on the 'Add' button on the left side of the window. The file will be moved from the Untracked box to the 'Added' box:

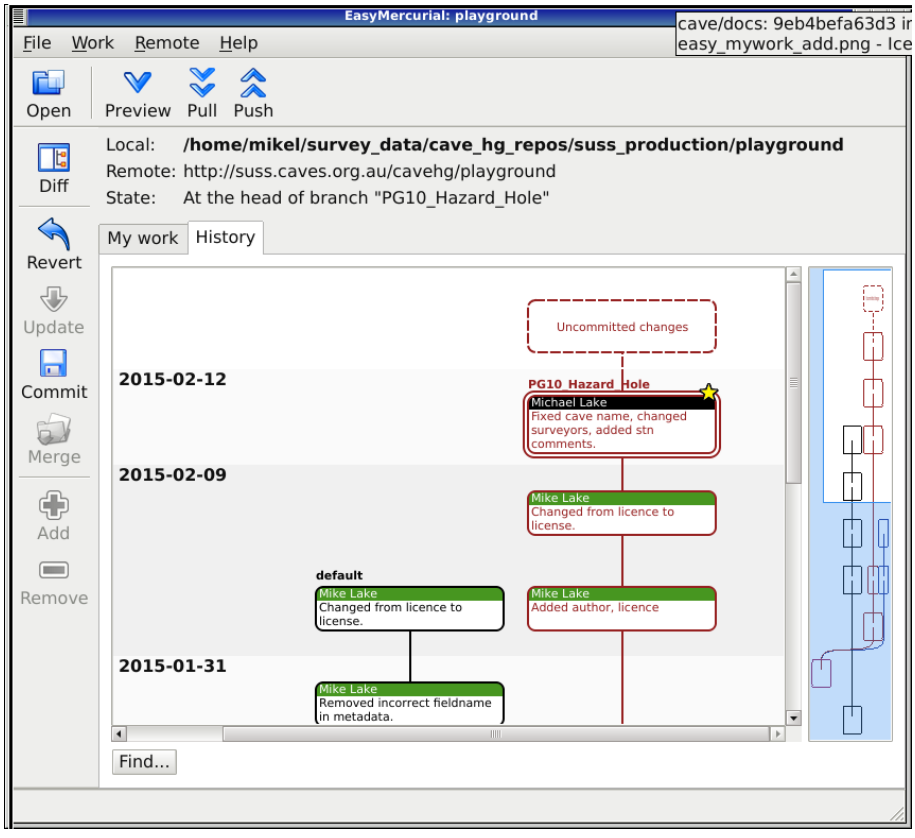


Files added to the archive

You can create new folders in your working directory and copy files into them. These files will appear in the Untracked box of your My Work tab with a name that includes the folder they are in (the "path" of the file, indicated by a slash between folders, e.g. `cavesurveys/datasheets/sheet1.jpg` is a file called "sheet1.jpg", residing in a folder called "datasheets", which resides in a folder called "cavesurveys" which resides in your working directory). These files are brought under version control using the Add button in the same way as other files. They will be added to the archive within their folder, and when uploaded to the ASF server they will reside in the branch on the same path as the path they reside on within your working directory.

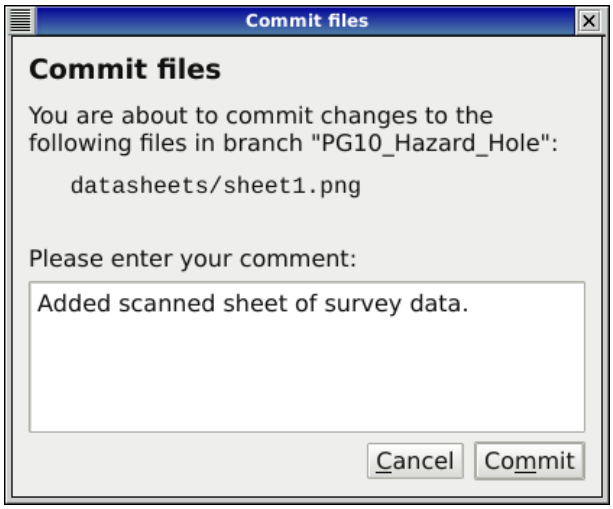
Committing Files

The History tab will look like this once you have Modified or Added files. The Uncommitted changes are not currently under version control. You do not need to Commit files every single time you make a change to a file. There is no limit to the number of changes you can make before you Commit the files. However, it is not a good idea to leave files uncommitted for days or weeks (See the Best Practise section, as well as the Merging section). As a rule of thumb, the trick is to make a set of related edits to a file (e.g. one section/chapter), or Add a related set of files to the repository, and then Commit the changes. You must Commit before attempting to Update to a different version or branch in the History tab.



Uncommitted files in the History tab

Once ready, the user Commits the files to the repository by clicking on the Commit button on the left side of the window. When you do this, a dialogue box will appear prompting you for a comment. Please add details about the set of changes you are Committing to the archive (see the 'Best Practise' section).



Commit dialogue box

Click on the Commit button to Commit the changes to the archive on your local machine. The History tab will update to show the latest revision of your files on the current Branch. You can Commit as many times as you like before sending the changes to the ASF server, but that's not such a good idea (See the 'Best Practise' section). If you have internet access, it's best to Push your Committed changes regularly as you Commit.

Pushing Files

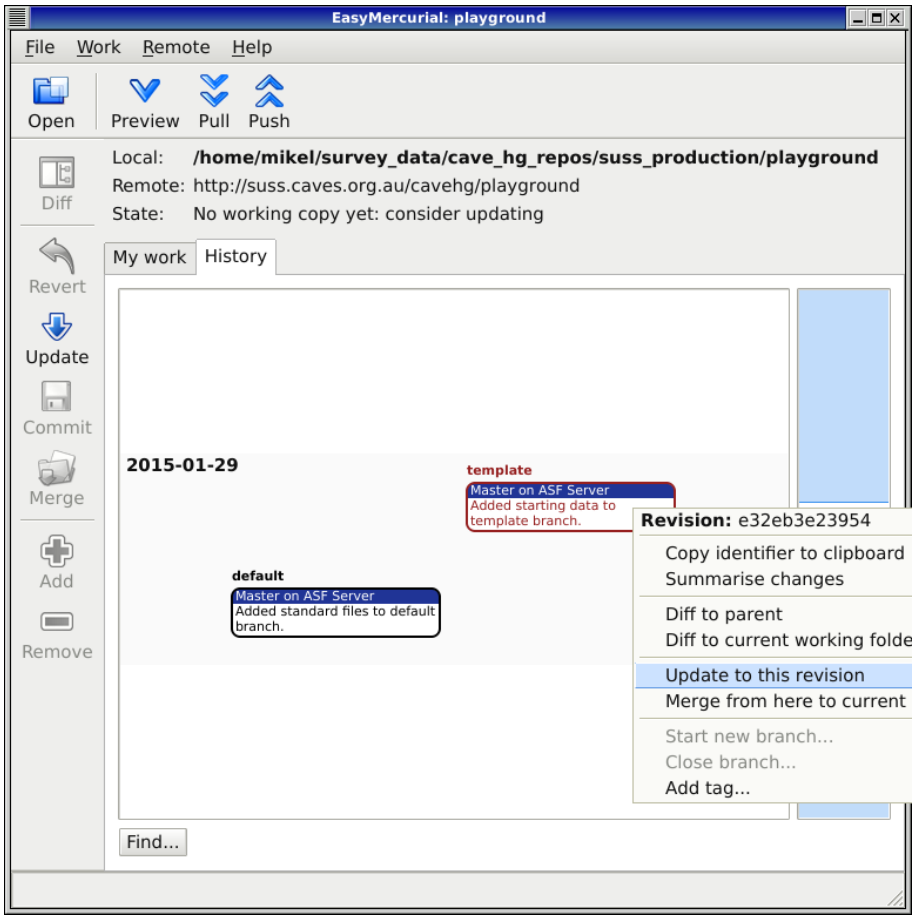
Once you have Committed changes in the repository, you need to upload them to the ASF server. Click on the 'Push' button on the top of the window. Confirm that you want to Push the Committed files and you will be asked to enter your user name and password. If the upload is successful, a confirmation of the upload will be shown.

8 Creating a New Branch

Before you attempt to create a new branch:

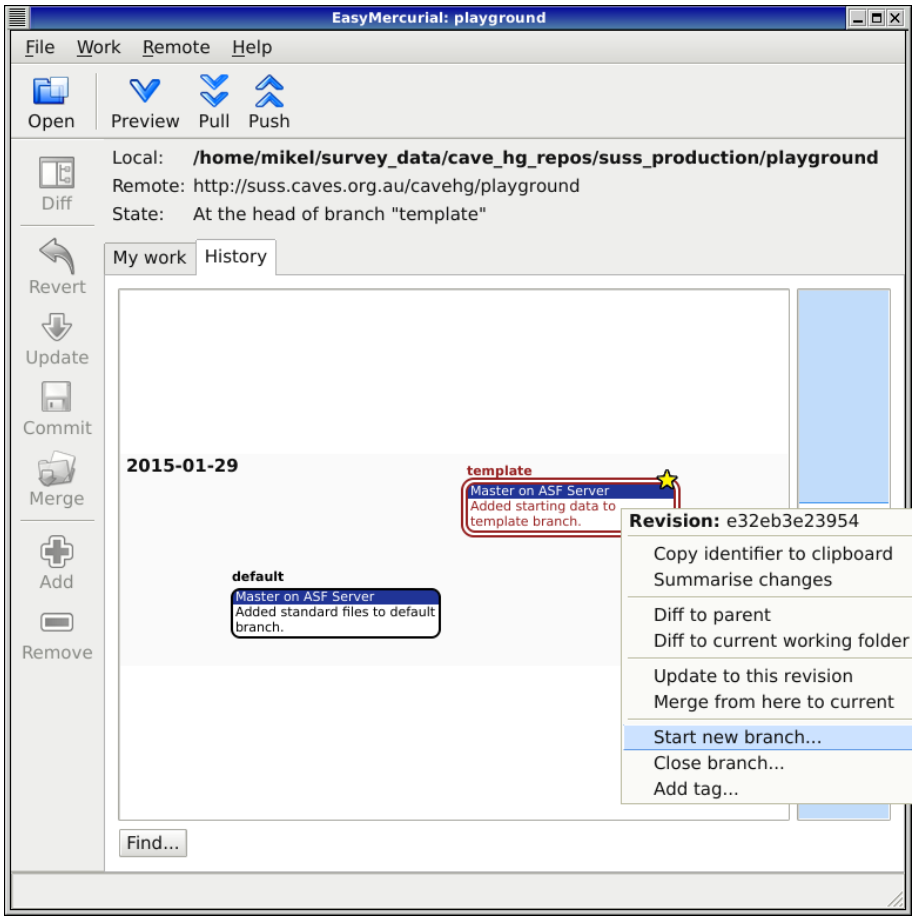
1. Make sure that you have no uncommitted changes for the repository.
2. Do a Pull from the repository on the ASF server to make sure that you have the latest updates from other contributors.

In the History tab, Update to the `Template` branch (right click on "template" and "Update to this revision"). You will get an "Update successful" box, click OK. There will now be a "star" on the template branch.



Go to the template branch and "Update to this revision".

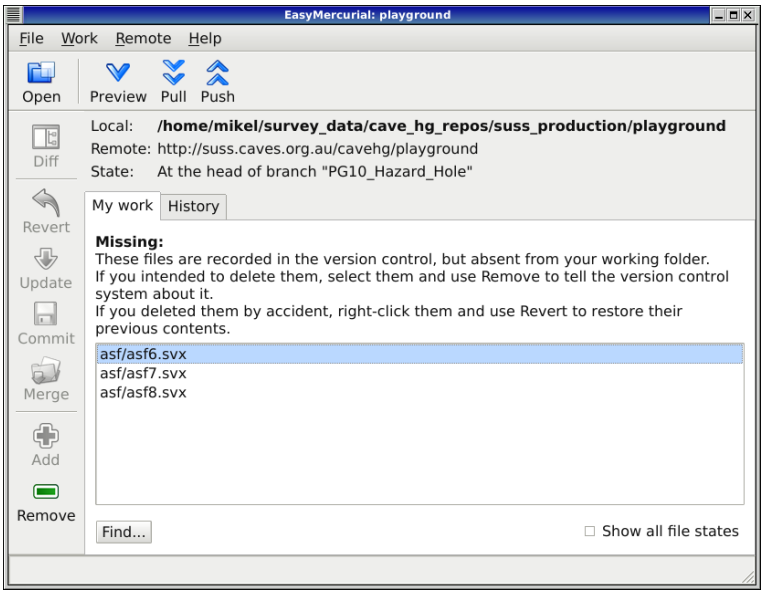
Now right click on the `Template` branch and from the drop-down menu, select "Start new branch".



Select “Start new branch”.

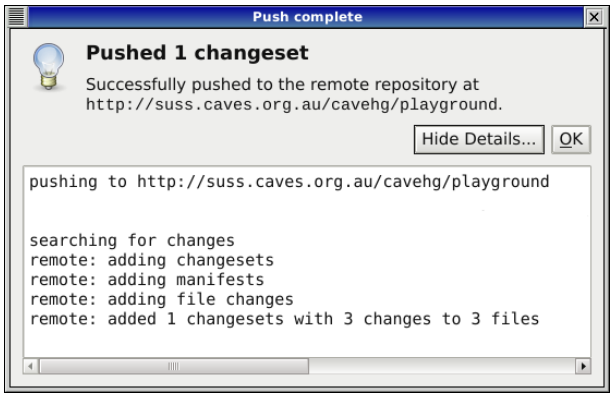
When you select ‘Start new branch’, you will be asked to name the new branch – use the tag number and name of the cave if the branch is for a cave, e.g. `J13_Mammoth`, `CL1_Main`, etc. See the Best Practise section.

After you create the branch, the History tab will initially show no change to the structure of the repository, but the State: line at the top of the window will say ‘On branch “*your-new-branch-name*”. New branch: has not been committed’. Change to the My work tab and carry out any necessary edits to your working directory. If you delete unneeded files from the working directory, they are moved to the ‘Missing’ box in the My work tab. Select them in the My work tab and click on the ‘Remove’ button on the left hand side to remove them from version control. Once you have removed unneeded files from the new branch, move any new files or folders into the working directory and use the Add button to bring them under version control.

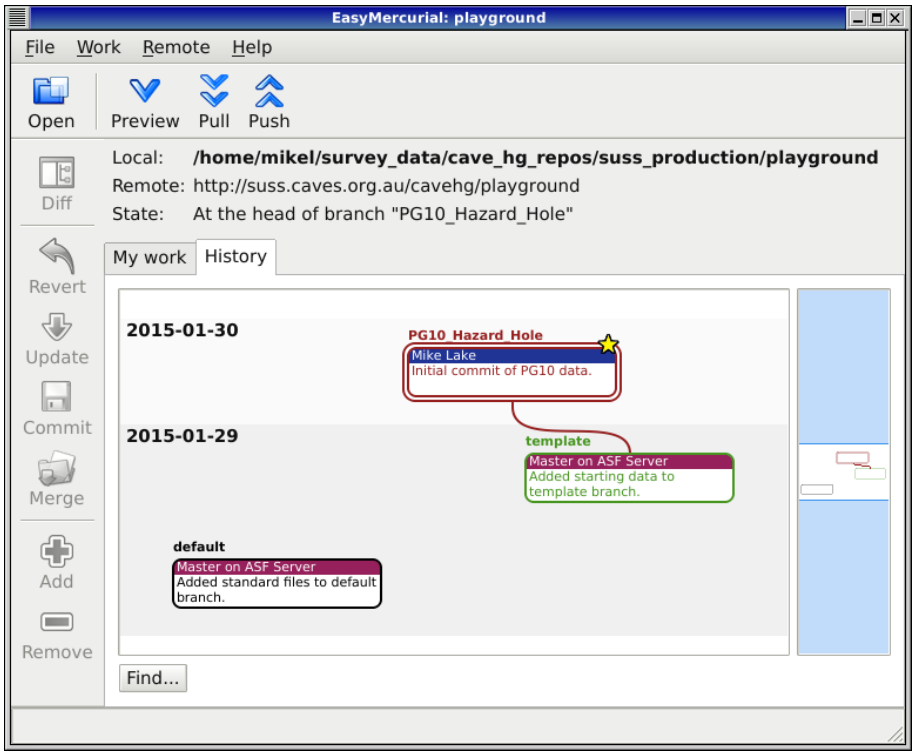


Deleted files to be removed from the version control system.

Edit any files as necessary (e.g. the `metadata.ods` spreadsheet file). When you are ready, click on the Commit button on the left side of the window. Enter a comment, and the new branch will appear in the History tab as a branch off the `Template` branch. The State: line at the top of the window will say "At the head of branch '*your-new-branch-name*' ". Push your Committed changes to the ASF server.



Push one change.



In the history tab you will see the new branch has been created.

Let your club administrator know when you create a new branch. They'll need to edit the `metadata.ods` file on the Default branch of the archive to make your newly-created branch show on the website.

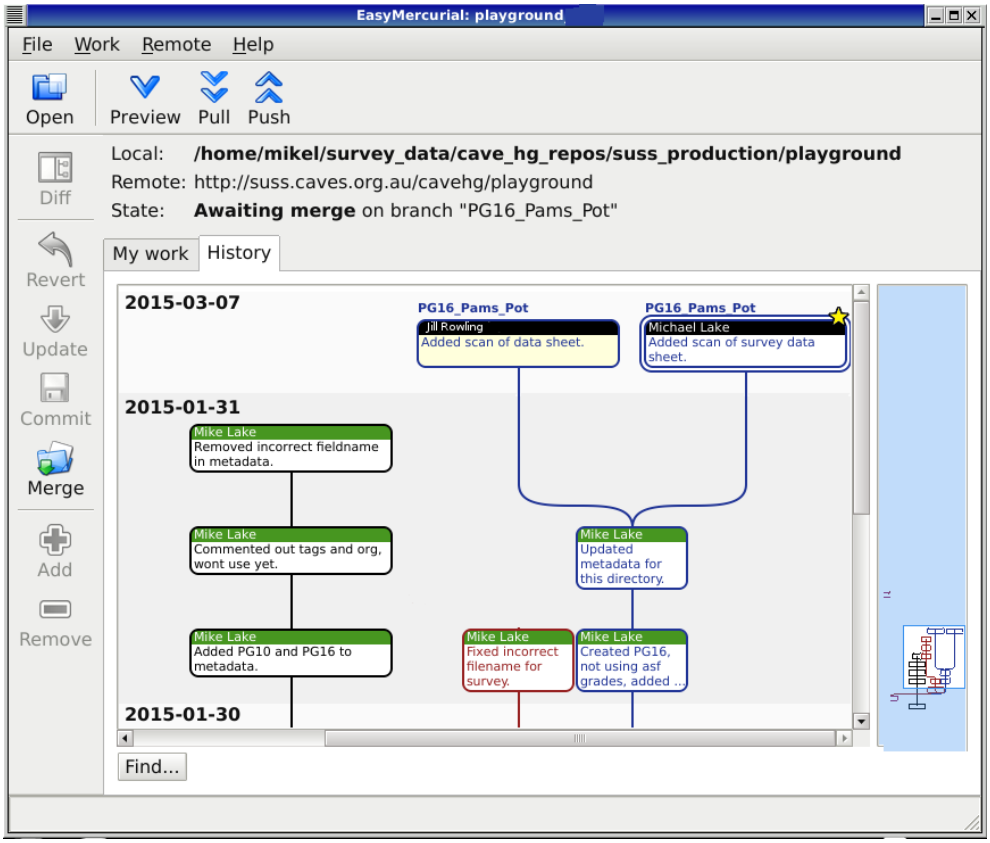
9 Merging

When you Push a set of Committed changes to the ASF server, your version of the documents becomes the definitive version. What happens when someone else is editing the documents at the same time?

If you try to Push your Committed changes up to the ASF server, and someone else has modified the branch since the last time you Pulled the branch from the ASF server, you will not be able to carry out your Push command.

Push command fails because someone else beat you to it

The other person's version of the repository is the definitive version on the ASF server at this point. That means the first thing you need to do is Pull their version of the branch from the ASF server. When you do this, the History tab will show the other person's Committed changes and your Committed changes as parallel branches split from the same parent version of the branch. Both versions of the branch will have the same name, and show the user name of the person who Committed them. The History tab will also show the times at which the various versions were Committed.



After Pulling changes – the other person's changes are shown in the History Tab

There are several possibilities at this point:

1. The other person has added files to the branch which you've never seen before. The Update button on the left of the window will be greyed out. You need to click on the Merge button on the left of the window. The new files will show up in your working directory, and in the Modified box in your My work tab. The files are ready to be Committed.

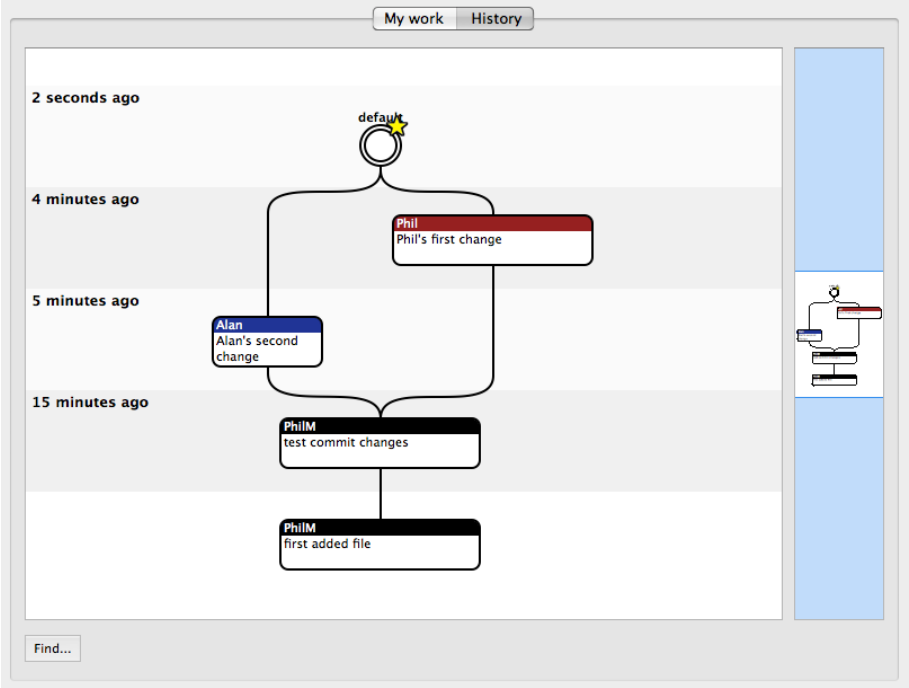
2. The other person has edited files which were on your version of the branch, but they haven't edited any of the files which you were editing. The Update button on the left of the window will be greyed out. You need to click on the Merge button on the left of the window. The changed files will be updated in your working directory and show up in the Modified box in your My work tab. The files are ready to be Committed.

3. The other person has removed files which were on your version of the branch (but not the files you were editing). The Update button on the left of the window will be greyed out. You need to click on the Merge button on the left of the window. The Merge command will have removed the files in your working directory! Do these files really need to be removed from the archive? You can put them back in by copying files into the working directory and Committing them. Alternatively, you can accept the removal of the files from the branch by Committing and Pushing the merged file list. You'd need to have a discussion with the other user.

4. The other person has edited files which you have also edited. This is the most complex situation, and you can avoid it by regular Pull, Update, Commit and Push commands (See the Best Practise section).

   You need to click on the Merge button on the left side of the window. The program will attempt to automatically merge the two versions of the file. In many instances, the program will be able to identify the difference, e.g. one version of a text file has information added at a particular point while the other version is unchanged. The program will automatically add the added information to the merged file. Inevitably, there will be points in the document where both users have made independent changes to the original document. The program cannot automatically resolve this situation. Instead, it will open an editing window where both versions of the file and a proposed output version are displayed. You will need to manually edit the output version of the file until all Merge conflicts are resolved.

After you have completed the Merge process, you need to Commit the merged branch immediately. Do not make any further changes to the branch until you have Committed the Merge. Once you've carried out the Commit command, the History tab will show the two parallel branches brought back together and you can now edit the branch normally. You can also Push this version of the branch to the ASF server.



Branch Merged, Committed and now able to be Pushed

TODO Show window decorations. Use cave example with better commit msg.

10 Getting Email Notification of Commits

You can get an email notification whenever another user makes a commit to the repository. The email will contain who committed, the date and the commit message of the "change sets" and links for each of them to the ASF resository web interface. This feature should be used as it allows you to keep abreast of changes to the repository. It also effectively reminds you to do a pull if someone has committed to the repository. An example is shown below.

Contact your club administrator to enable email notifications for you.

Below is an example of an email that is sent.

Subject: playground: 2 new changesets
From: mercurial@caves.org.au
Sent: Monday, March 09, 2015 9:45 PM
To: Michael Lake; Philip Maynard

```
details:   http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground/rev/c79cd4f994c3
branches:
user:      Mike Lake <Mike.Lake@uts.edu.au>
date:      Mon Mar 09 20:55:08 2015 +1100
description: Added new data to Hazzard Hole.

details:   http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground/rev/5de3d3c6d336
branches:
user:      Mike Lake <Mike.Lake@uts.edu.au>
date:      Mon Mar 09 20:59:12 2015 +1100
description: Removed asf grade files as we will not be using them.
```

11 Copyright, Licensing and Access Control

Copyright

When you create the document, you own the copyright. There's no process for you to go through to assert copyright – it's automatic. The term of copyright in Australia is the life of the Author plus seventy years. This is a huge deal for an archive. Fifty years after your death, we may be chasing up your great grandchildren for permission to use a file if you don't give permission up front.

When you write down cave data in the cave such as station, bearing, elevation and distance or passage widths it is not copyright to you. Such data is classed as a "mere aggregation of data" and as such is not subject to copyright. Sketching appears to be copyright.

Licensing

If you own the copyright for a document, you control the use of that document. Unless you give permission to others, they may not copy, lend, sell, display, modify (create "derivative works") or extract significant portions of your document. This is where licensing comes in. You can grant a license to another person to allow them particular rights to use your copyrighted document. For the club archive, this is critically important!

While a license could be written to say anything, there are pre-written licenses available for anyone to use which have stood the test of time. The most significant of these are published by the Creative Commons organisation, and it is strongly suggested that these licenses should be the basis of any document committed to an archive. An example of a Creative Commons license is the *Attribution* license. This gives permission to anyone to: copy, share, display and create derivative works from a document, subject only to the requirement that they give credit to the original author including credit for the original part of a derivative work. This license (abbreviated as "CC-BY") is what would make the most sense for a non-commercial archive document.

When you add a file to the archive, it will need licensing information placed in the `metadata.ods` file for the branch, unless it is purely a data file. Otherwise, it will be need to be subject to a default license policy decided by the club. It must be emphasised that an archive cannot be based on documents where all rights are reserved.

Changing your License

If you change your license for a document you must change the entry in the `metadata.ods` file and record this in the Commit message clearly and fully. Entering "Changed license" in the Commit message is not sufficient. You must enter something like "Changed license from CC-BY to CC-NC".

There is a very important reason for this: someone may use your work under a particular license, and if you later change that license we need to have the change and the date of the change recorded in the repository in a manner that we can easily find. The history of commit messages is the easiest place to look. We can extract just the commit messages for a particular file. We don't want to have to look at previous versions of the metadata.

Access Policy

Within the default branch of a repository is the `access_policy.ini` file. This defines the access policy for all of the branches in the repository, including all of the folders inside the branches. The policy can be overridden on a branch-by-branch basis by a user. Any change in policy for a branch applies to all of the folders inside the branch.

If the `access_policy.ini` file contains the line `default = PRIVATE` then in the absence of any override all data (except metadata) is considered private and the program that creates the browsable webpage will not copy the files – just the metadata.

If the `access_policy.ini` file contains the line `default = PUBLIC` then in the absence of any override all data (except metadata) is considered public and the program that creates the browsable webpage will copy all the files including the metadata.

Note that files in the default branch can only be changed by the club administrator. This means that only the administrator can change the default access policy. However any user with editing access to a branch can change the access policy of that branch by overriding the default policy as described below. The rationale here is that the admin person sets a sensible default policy for the entire repository and users override that on a per cave basis as appropriate.

Default Policy "Private" and Override on a Per-branch Basis

In the `access_policy.ini` file `default = PRIVATE` would be set. In any branch or folder within a branch, create a text file called `PUBLIC`. This file can be an empty file or it can can contain a comment describing why this branch has this access policy. This will set this branch or folder and all folders below this to public until another override (e.g. `PRIVATE`) is encountered. There is no need to add this file to the list in the `metadata.ods` file.

Default Policy "Public" and Override on a Per-branch Basis

In the `access_policy.ini` file `default = PUBLIC` would be set. In any branch or folder within a branch, create a text file called `PRIVATE`. This file can be an empty file or it can can contain a comment describing why this branch has this access policy. This will set this branch or folder and all folders below this to private until another override (e.g. `PUBLIC`) is encountered. There is no need to add this file to the list in the `metadata.ods` file.

12 Server Generated Content

There is some content that can be automatically generated by the server that the repo resides on. The mechanism for this is to create a text file in the directory that you wish to be processed. This is similar to how you set a directory to be private by creating a text file called PRIVATE in that directory.

At present only automatic creation of Survex files is supported.

Survex File Processing

If a directory contains a Survex file that could be processed by running "cavern" then you can create a text file called SURVEX which contains just the name of the Survex file and cavern will be invoked on the server to process that file. The Survex output (error file and 3d file) will then be available on the browsable website. If you have more than one Survex file then list them in the SURVEX file one per line.

As an example; if you want to run "cavern bullio.svx" then just place "bullio.svx" into a file called SURVEX.

13 Best Practise

1. *Working directories.* You need to create a working directory on your computer for your local work. If you are going to work with more than one repository (more than one caving area) then you need to create a different working directory for each repository you are working on. Naming the folders logically ("archive_jenolan", "archive_buchan", etc) will be helpful for you.

2. *When to Pull.* It's important to try and stay up to date with the work of others. This minimises Merging, and allows others to see your work. Remember, you can't Push your changes to the ASF server until you've Merged all of the changes made by other people. You should Pull and Update each time you're about to start work on the repository, and before you do something major like creating a new branch.

3. *When to Commit.* Group your Commits logically. When you've completed a scan of some sketches it's time to add them and commit (commit message "Added sketches"). When you have corrected some survey data and/or added new data it's time to commit (message "Fixed incorrect bearing, added left hand passage data."). If you have entered a day's survey data in a mapping project, it's time to commit. When you bring a batch of files into your working directory and add them all to the archive, it's time to commit. It's not necessary to commit every time you save a file or add one file, and doing that would create a very long log full of versions that are virtually identical. But its best not to combine logically different changes like adding new scans and fixing survey data into one big commit.

4. *Explain your Commits completely but succinctly.* The comment dialogue box that appears when you try to Commit is very important. All of the previous versions of the branch are available to you through the "Update to this revision" command, but if you don't know what a previous Committed revision actually consists of, then that's not very useful.

5. *When to Push.* If you have access to the internet, you should Push at the end of each editing session. You don't have to Push every time you Commit. On the other hand, the longer you delay your Push command, the longer it will be before other users can see your edits. You will increase the chances of having to do a Merge if you delay the Push command. If you don't have internet access, then you should Push as soon as you can gain access.

6. *Creating a new branch.* You can create a new branch from any historical state of any existing branch. When you create a branch, all of the files in the existing branch will be copied into the new branch. That's why the best way to create a new branch is from the *template* branch. The template branch will be empty of existing data files, but will have the files you need to start working on a new cave, e.g. a `metadata.ods` spreadsheet file which can be modified for the new cave. The Template branch will also have folders in the working directory set up for things like scanned data and sketches. On the other hand, if you branch off an existing branch that has data, you'll have to delete all of the existing data files in the new branch and Remove them in the 'My work' tab. Creating a new branch from a branch that has hundreds of files in it would be very, very tedious.

7. *Size.* Try to keep the repository as small as possible. Only store the canonical stuff. Don't store stuff that is automatically generated by the programs you use, e.g. store Survex data files, but not the .3d files generated by Survex from the data files.

8. *Size.* Scanned documents may the most significant part of your branch in terms of size. For data sheets and sketches taken in caves, we suggest scanning at 150 dpi. This allows sufficient detail for tracing in a vector drawing program, while reducing the file size by $\approx 75\%$ compared to 300 dpi. Use compression when you save your scanned files.

9. *Collaborate via email.* Use the email notification system so others know when you have made commits. This will remind them to pull before they make any changes.

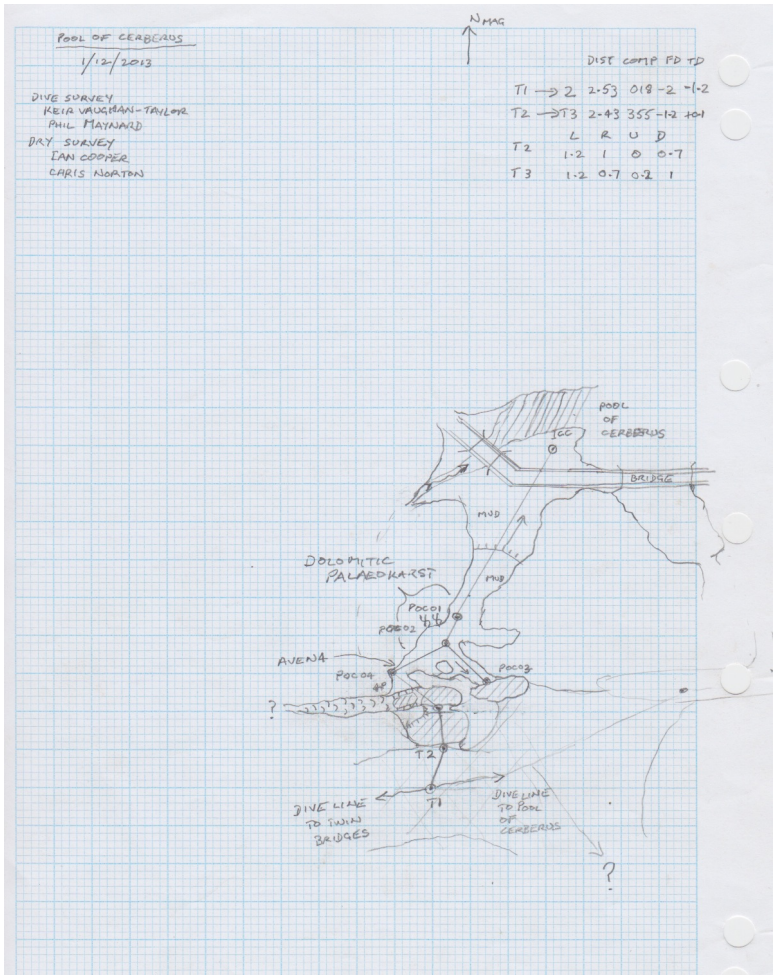
10. *Collaborate by dividing up the work.* Arrange for different persons to work on different parts of the data so that if merges need to occur they are simple merges. If lots of persons work on the same data eventually you will need to merge. If someone forgets to pull before they commit they will need to merge. Multiple merges can be tricky.

11. *Due care and attention.* When your club administrator gives you access to the archive, they are giving you a lot of power to do damage to the archive. No-one expects malicious behaviour from a club user, but reckless or negligent use of the archive could damage the club's records significantly. Pay attention to what files are already in existence in the branch, and what you are doing to them.

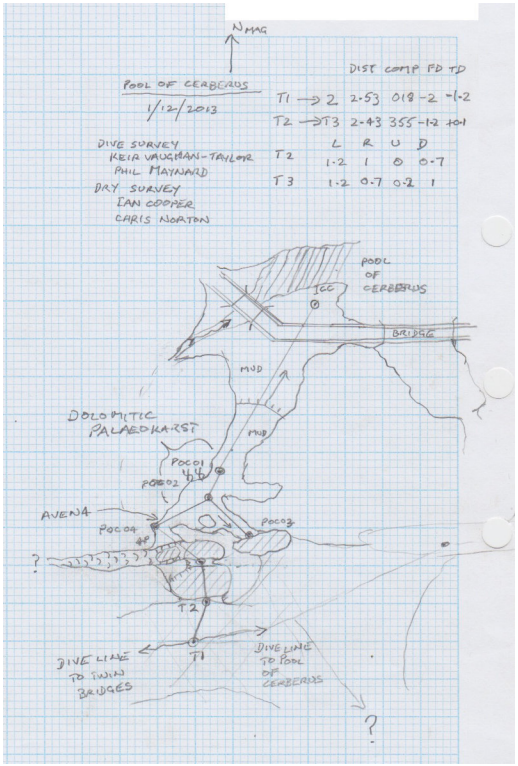
14 Including Scanned Images of Original Field Notes

For all surveys include scanned images of the original survey data, notes, and sketches. However we also wish keep the total size of the repository as small as possible. Therefore before adding an image check to see if it can be reduced in size.

Below is an example of a cave diving sketch scanned at 150 dpi and 8 bit colour. There is a large gap between the title at the top and the actual cave sketch. Before adding the image we can edit it, cutting and pasting the title to a different location within the image, then cropping the image to a smaller size.



Original scanned sketch

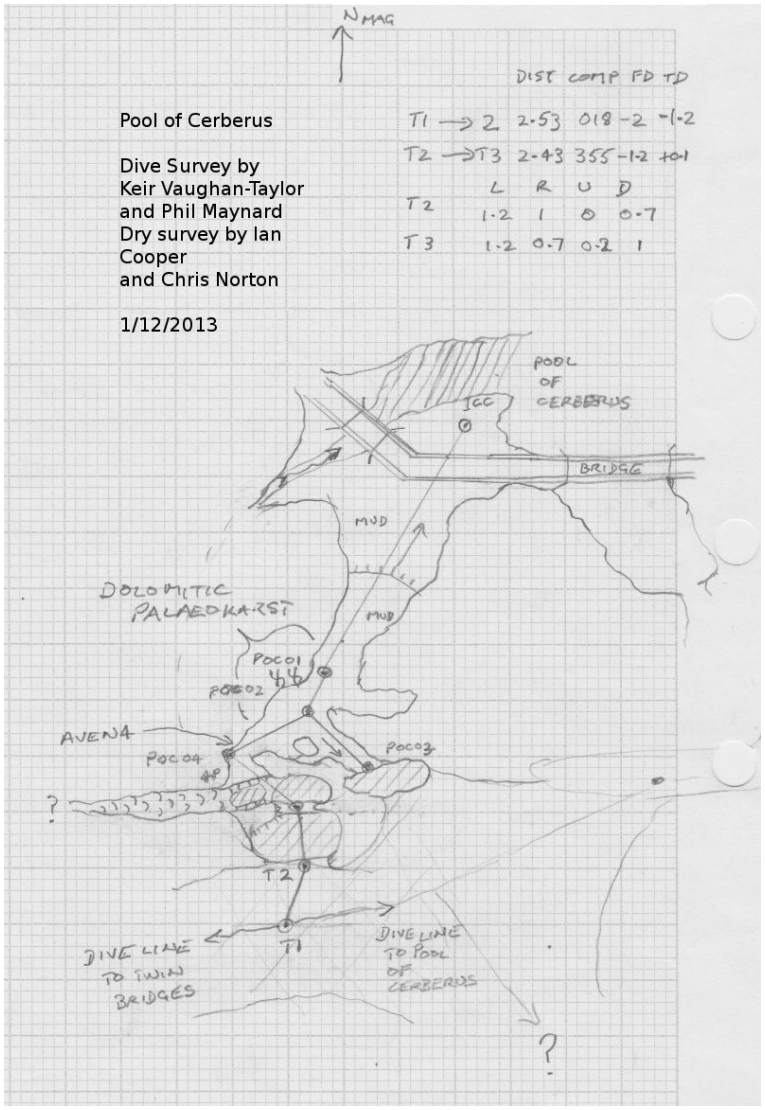


Edited sketch.

The original graph paper in this example was 20 cm wide (≈ 8 inches) and the scanned image is 1200 x 1500 pixels, 8 bit colour. It is 550 KBytes in size. The edited image is 800 x 1190 and is just 250 KB in size, that's half the size.

8 bit colour or greyscale: Should we use 8 bit colour or greyscale? The reason for scanning in 8 bit colour is that often an in-cave sketch has mud or smudges on the sheets and sometimes that mud will overlie your pencil marks. Using 8 bit colour you can tell a pencil mark from a mud splot. Once it is greyscaled you can't tell the difference. The same problem can arise on data sheets, a mud spot might look like a decimal point. Most of the time you would scan in 8 bit colour but always check the image and if you can safely greyscale then do so. In the example above we can use greyscale safely. Reducing the color from 8 bit color to greyscale then reduces the size to just 170 KB (see image on next page).

Adding non-original information: Sometimes a sketch may be missing important infomation such as a description, author or date. We can add this information using an image editor. When we add additional information we want the alteration of the image to be such that a user can tell the difference between the original "in-cave" sketch and subsequent additional information. This should be clear in the example below as the original data is in handwriting and the additional information is in a "computer font".



In the image on the left this is what you would do if the sketcher had not entered the required information. You can clearly see the information that has been subsequently added.

Also as this sketch was quite clear and there was no mud on the page to obscure faint drawing lines we have made it greyscale instead of 8 bit colour. The size is now just 175 KB.

Summary:

- **scan** at 150 dpi and 8 bit color. Use png or jpg with 85% quality.
- Check its **8 bit color**, reduce if needed. Consider using **greyscale**.
- Check pixel **height** and pixel **width** are appropriate for the original page dimensions and **scale** and **crop** if needed.
- Check if **required information** is on sketch, add if needed.

A5 page scans: Small note books used for cave surveys are usually A5 sized, this is 150 mm x 210 mm. A scan at 150 dpi of this size page would be:
150 mm/25 mm per inch x 150 dpi = 600 pixels across and
210 mm/25 mm per inch x 150 dpi = 1,250 pixels high.

A4 page scans: For larger sketches this is 210 x 297 mm and a scan at 150 dpi would give an image of 1240 x 1756 pixels.

Always check the size of a scanned image, if its significantly larger than the above figures then it may have been scanned at too high a resolution or the color depth may be higher than what is required.

15 Tips for Specific Software

Adobe Illustrator

Adobe Illustrator files are actually PDF version 1.4 format files. They can be exported to SVG format in two ways; either uncheck the option "Preserve Adobe Illustrator Editing" or set "Optimize for Adobe SVG viewer". This later option is the best because this way the SVG will not contain the binary blob of Adobe specific information. Ref: http://wiki.inkscape.org/wiki/index.php/Frequently_asked_questions

The Gimp

This bitmap editing software can be used to recsale and adjust the depth of your scanned images. The Gimp is open source and available on Windows, OSX and Linux. It's an excellent alternative to using expensive Adobe Photoshop. Ref: http://www.gimp.org

16 Glossary

Version Control System Concepts:

repository: This is where the versioned files are kept. A repository is usually just a folder containing many files.
push: This is when you copy all changes from your local repo to a remote repo.
pull: This is when you get a copy all changes from a remote repo to your local repo.
checkout: This is when you extract copies of files from your local repo to a "working directory".
checkin: This is when you enter copies of changed files from your "working directory" back into your local repo.
working directory: The directory where you have checked out files and where they will be edited.

Other useful definitions:

metadata: Data that describes data.
ODF: Open Document Format is an open standard for word processing and spreadsheet files. It is an ISO standard (International Standards Organisation) and all word processing and spreadsheet programs should support import and export of this format.

17 References

Copyright and Licensing

http://creativecommons.org.au Creative Commons Australia

Using the Mercurial Version Control System

http://mercurial.selenic.com The Home page for Mercurial has many links to tutorials.

http://www.ericsink.com/vcbe "Version Control by Example" is a good tutorial. The section covering Mercurial is here: http://www.ericsink.com/vcbe/html/mercurial_example.html

http://hginit.com "Hg Init: a Mercurial tutorial" is an excellent six-part tutorial by Joel Spolsky covering the key concepts.

Here are some references on using KDiff3

http://naleid.com/blog/2012/01/12/how-to-use-kdiff3-as-a-3-way-merge-tool-with-mercurial-git-and-tower-app

http://kdiff3.sourceforge.net/doc/merging.html

The open document format is the native format used by Libre Office and Open Office. These are free and open source office suites and they can be downloaded from the links below. ODF is also a published ISO standard (ISO is the International Standards Organisation).

LibreOffice can be downloaded from: http://www.libreoffice.org

OpenOffice can be downloaded from: http://www.openoffice.org

Information on ODF format: http://www.opendocumentformat.org and
http://en.wikipedia.org/wiki/OpenDocument

Appendices

A Installing EasyMercurial

EasyMercurial is a GUI for Mercurial. Download EasyMercurial from http://easyhg.org

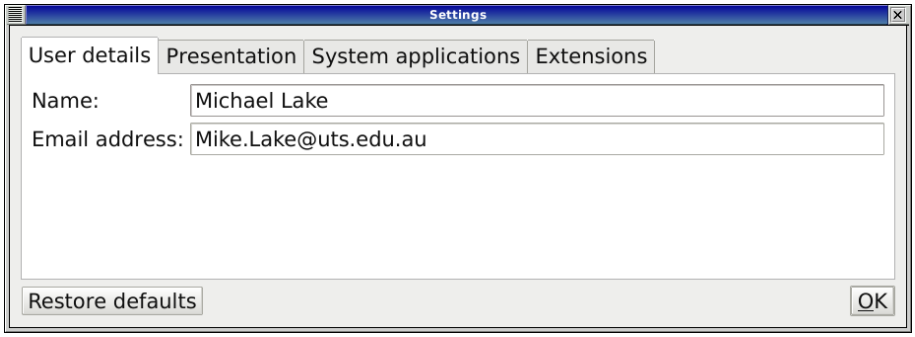
If you are using Vista, Windows 7, or Windows 8, just download, install and run. The Mercurial executable and the KDiff3 diff/merge application are included.

If you are using OSX then follow the instructions at the Easy Mercurial site.

Important: After installing EasyMercurial (and before doing any commits!), you need to set your real name (not login name) and email. The name you enter into the 'Name:' line will be your identifier in the History of the repositories, showing that a Commit came from you. You should not change this name once you set it.

Windows Users: Select File / Settings and select the User Details tab

OSX Users: Select Preferences and in the Preferences dialogue box, select the User Details tab.



Set your real name and email address. You must do this before you can push any of your changes up to the repository.

B Other Mercurial Graphical User Interfaces

You don't have to use EasyHg or the Hg command line to manage your local repository or access the remote mercurial repositories on the ASF server. There are several Mercurial clients that can be used.

SourceTree is a free Git and Mercurial client for Windows or Mac; http://www.sourcetreeapp.com/

TortoiseHg for Windows has been around a long time and is quite good;
http://tortoisehg.bitbucket.org

For a comprehensive list of "Graphical user interfaces" for Mercurial see
http://mercurial.selenic.com/wiki/OtherTools. This also covers native Mac OSX options.

C Using Hg from the Command Line

Using Hg from the command line is quicker, and more powerful features are available. If you are using Linux you will probably want to use the command line interface, even though EasyHg is available.

In Linux just install the mercurial client from your package manager.

Get help on the hg command with **hg help** or more detailed help on a command with **hg help command** e.g. **hg help status**

Cloning Repositories

First you need to clone one of the SUSS repositories in order to have a local repository to work with. In this section as an example we will clone into a local directory called `playground` from SUSS's Playground repo at `http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground`. Make sure you are in the top level of where you store your Mercurial controlled cave survey data. This might be for instance your local directory `/home/bill/survey_data/cavehg/`, or if using OSX it might be like `/Users/bill/survey_data/cavehg/`. You don't have to use `cavehg`, you might use `caverepos` or something else. However you should name it so that you will know that under this directory are mercurial repositories and not just "normal directories".

The command below will create the directory `playground` if needed.

```
hg clone http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground playground
```

The general syntax to use is `hg clone [--branch branch_name ] remote_url local_directory`. If you don't wish to clone the entire repo then using the `--branch` or `-b` option will clone just one named branch and any of its ancestors. If the local directory does not exist it will be created in the current directory. In the example below we clone just the PG10 Hazzard Hole branch.

```
hg clone -b PG10_Hazard_Hole http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground
   pg10_hazard_hole_branch
```

Note: To find out what named branches exist go to the public web site at http://suss.caves.org.au/cave/playground or the repo web site at http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground.

You can also clone from a local directory to another local directory. For instance, to make a clone of your local playground repo to a new repo in directory playground_2014:

```
hg clone playground playground_2014
```

Updating and Committing

The general procedure that a user follows is of multiple edit(s) followed by commits and a push to the ASF server.

`$ hg pull` Pull down latest version before you do anything.
`$ hg up` Update your working directory to this latest version.

⟶ perform edit(s)
↑ `$ hg st` Check what you have changed.
↑ `$ hg diff` Maybe check details of how its been changed.
↑ `$ hg ci -m 'commit message'`
⟵ repeat procedure

`$ hg push` Push all your latest commits to the ASF server.

Summary of Common Commands

Here is a short description of the commands most often used.

hg clone *source destination*: Clone, i.e. make a copy of, the repository at *source* (usually a `http` location) to your local directory *destination*.

hg status / hg st: Lists the status of the files in your working directory.
If a file under version control has not changed it will not be listed. If it has been modified, or files added or removed then the filename will be listed and prefixed by one of **M**odified, **A**dded, **R**enamed or Removed.

hg branches: Lists the branches in your local repository.

hg branch: Shows the name of the branch that you are currently on.

hg pull: You need to be in a repos working directory (e.g. `/home/bill/surveys/`). This command will pull down from the remote repo any changes into your local repo. Before you see any changes you will need to "update".

hg update / hg up: This will checkout from your local repo any new or modified files into your working directory.

At this stage you may have worked on a few files, making changes. Run **hg st** to show the status of the working directory which will summarise what files you have modified.

hg diff [FILE]: This takes an optional filename. If there is no file listed and only one version controlled file has been modified, then it will invoke whatever default "diff" program you have (see below) and compare the modified file with the latest one in your repository. If there is more than one modified file you need to invoke **hg diff FILE**.

hg commit -m 'some message': This commits modified, added or removed files from the working directory to the repository. You must supply a commit message. *Commit messages are important – they describe the history of what you and other updaters have modified.*

hg push: This pushes changes in your local repository to the remote repository on the AF server.

hg revert FILE: If a file in your working directory has been modified and you wish to throw away your recent changes and use the version that is in the repo use the revert command. Also see "hg help rollback" and "hg help forget".

hg add FILE: This will add the FILE in the working directory to version control.

hg remove FILE: If there is a version controlled FILE in your working directory this command will schedule it to be removed from version controll. It will delete the file from your working directory at the next commit.

hg rename / hg mv: Files in mercurial are not actually renamed; they are copied to a new file and the old one removed. The actual copy and delete will occur at the next commit.

Creating a New Branch from the Command Line

A summary of the procedure is below followed by a detailed example.

Summary

```
$ hg st          ← Check there are no outstanding changes.
$ hg pull        ← Pull down the latest changes from the ASF server.

$ hg up default
```
Now edit the metadata.ods file in the default directory, adding the new branch name.
```
$ hg ci -m 'Added new branch new-branch-name to metadata' ← Checkin your changes.

$ hg up template
$ hg branch new-branch-name ← Create the new branch.
```
Now add required files and/or remove any not needed files.
Edit the metadata.ods file in the templates directory with metadata for each new file and directory.
```
$ hg add your-files ← Tell hg what files you have added.
$ hg ci -m 'Added new branch new-branch-name and data' ← Checkin your changes.

$ hg push --new-branch
```

Detailed Example

In this section we will create a new branch for Hazard Hole (cave number PG10) in the "Playground" Caves repo.

First check there are no outstanding changes to commit.

```
$ hg st
$
```

Check you have the latest version.

```
$ hg pull
http authorization required
realm: Access to SUSS Cave Repos
user:
password:
pulling from http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground
searching for changes
no changes found
$
```

See what branches or caves we have.

```
$ hg branches
default              50:8e8908c8b53f
PG16_Pams_Pot        40:75ba4e66483f
template             39:88e9088cb5f3
```

From the above list of branches we can see that we don't have a branch for Hazard Hole so we can create one. (The numbers to the right of the branches are the `local revision number:unique revision hash`.)

First we update to the "default" branch and edit the metadata.ods to include an entry for this new branch.

```
$ hg up default
```

Update the metadata.ods file.

```
$ hg ci -m 'Added new branch for Hazard Hole to metadata.'
```

Next we update to the "template" branch. This contains typical files that most branches contain. It's also where we prefer new branches to originate from.

```
$ hg up template
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
$
```

```
$ ls
asf  metadata.ods
$
```

Now we create the branch.

```
$ hg branch PG10_Hazard_Hole
marked working directory as branch PG10_Hazard_Hole
(branches are permanent and global, did you want a bookmark?)
$
```

No we don't want a bookmark, we want a new branch, so ignore that warning message. The branch will have been created. We can check what branch we are on, it should be the new branch we just created.

```
$ hg branch
PG10_Hazard_Hole
$
```

Now at this stage I copied over from my old non-repo directory all the Pams Pot survey data to this directory. The general usage is `hg add "filename"` or `hg add "directory"`. Here I added all the files in one go. Notice the dot at the end of the add command, this is a shorthand for "all" files.

```
$ hg add .
adding README.txt
adding PG10.svx
$
```

Also you might wish to remove any files you don't want, such as the asf directory, if you are not using them e.g. `hg rm "filename"`.

Edit the metadata.ods for this new directory including the filenames and directories that you added. Note this must be done for all filenames and directories.

Now we commit the change to our local repo.

```
$ hg ci -m 'Added Pams Pot data.'
README.txt
PG10.svx
committed changeset 52:4d7becbf5426
```

Good, now let's push that changeset to the SUSS repo.

```
$ hg push
pushing to http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground
http authorization required
realm: Access to SUSS Cave Repos
user: mlake
password:
searching for changes
searching for changes
abort: push creates new remote branches: PG10_Hazard_Hole!
(use 'hg push --new-branch' to create new remote branches)
$
```

Ah, it won't let us automatically create a new branch, we have to be explicit that we really wish to do that.

```
$ hg push --new-branch
pushing to http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground
http authorization required
realm: Access to SUSS Cave Repos
user: mlake
password:
searching for changes
searching for changes
1 changesets found
remote: adding changesets
remote: adding manifests
remote: adding file changes
remote: added 1 changesets with 3 changes to 1 files (+1 heads)
$
```

Done. We can check right now that the change has been made to the SUSS repository by going to http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground and looking at the Log. After a while it will appear at http://suss.caves.org.au/cave/playground once the nightly cron job runs.

Example of Two Users Collaborating

This section provides an example of using the Mercurial via the command line. This is probably the main way that Linux users would use Mercurial, however the same commands can be used on a terminal in OSX or a Command Shell on Windows. The example consists of two users collaborating on some survey data.

Alice and Bob do a survey. The survex file of their survey is reproduced below:

```
; Playground Cave Area, NSW
; Survey of Hazard Hole, PG10
; Club: Sydney University Speleological Society
; Instruments: DistoX
; Surveyors: Alice and Bob
; Date: January 2015
; Description: Hazard Hole starts with a steeply descending passage,
;   ending in a nice chamber.

*calibrate declination -12.3

*begin pg10
*fix 1 0 0 0
*data normal from to tape compass clino
; From  To     Dist(m)    Bear       Elev   Comments
 1      2      1.11       31.5       14.0   ; 1 = tag at entrance
 2      3      1.89       86.5      -16.0   ; 2 = stal on roof
 3      4      5.83       72.0      -71.0   ; 3 = top of big stal on floor
 4      5      5.71       68.5      -60.0   ; all stns now are marks on mud wall
 5      6     11.95       64.0      -40.0
 6      7     22.39       44.5      -43.5
 7      8     16.12       46.0      -54.5
 8      9     13.46       38.0      -22.5
 9     10      5.65       31.6      -6.5
10     11      2.49       72.0      -35.5
11     12     15.87       10.5      -2.0
12     13      5.30      125.0       0.0
13     14     16.63      128.0      -4.5

14     15      8.82      188.5      -6.0    ; 14 = mud cairn in passage fork
15     16      5.72      107.0      -7.0
16     17      1.39       84.0      -16.5
17     18     16.92      171.5      +1.5
18     19      4.81      144.5      -6.5
19     20      1.89       86.0      +8.5
20     21     11.45      159.0      +5.5
21     22a    21.82      110.0      -0.5    ; 21 = cairn in middle of chamber
21     22b    12.0       165.0       0.0    ; letter suffixes are splay shots
21     22c    11.16      213.5      -3.0
21     22c    10.80      256.0      -6.0
21     22d    09.60      296.0      -4.0
21     22e     9.50       40.5       0.0
21     22f    10.05       86.5      +1.0
21     roof   15.2         -        up
*end pg10

; Survey of easterly trending tight passage.
*begin east
1      2      1.2     90.0      -10
2      3      1.0     95.0       -5
```

```
3      4      1.1     92.0        0
*end east
```

Alice pushes to the ASF repository: Alice commits this survey into the repository as a survex file. It's version 1. Alice's commit message is "Initial import of data."

```
alice$ hg ci -m 'Initial import of data.'
alice$ hg push
```

A few days later while looking at the original survey notes Alice sees that she has made a mistake in keying-in the data. There was some mud on the page next to the elevation entry for the leg 1 to 2. It was not a positive elevation leg, it was pointing down, and the mud was obsuring the minus sign.

Alice updates from the ASF repository: Alice pulls down, and updates to the latest version.

```
alice$ hg pull
alice$ hg up
```

There were no changes, and she now an up-to-date copy. Alice now edits the survex file.

```
1    2    1.11    31.5    14.0 ← This was wrong
1    2    1.11    31.5   -14.0 ← The elevation should be negative.
```

Alice pushes to the ASF repository: Alice's commit message is "Fixed error in elevation. Changed from -22 to +22"

```
alice$ hg ci -m 'Fixed error in elevation. Changed from 14 to -14'
alice$ hg push
```

Bob updates from the ASF repository: Bob pulls down this survey data (the usual `hg pull; hg up`) and makes a few changes. He changes the first survey station name from "1" to "PG10" so that the tag number will show when he turns on labels in the Aven cave viewer.

```
   1    2    1.11    31.5   -14.0   ; 1 = tag at entrance ← before the change
PG10    2    1.11    31.5   -14.0   ; 1 = tag at entrance ← after the change
```

```
bob$ hg ci -m 'Changed first station name from 1 to PG10.'
```

Also there are two Bobs in the club so he adds their surnames.

```
; Surveyors: Alice and Bob              ← before the change
; Surveyors: Alice Jones and Bob Smith  ← after the change
```

```
bob$ hg ci -m 'Added surveyors surnammes.'
```

Notice here that Bob did two separate commits to his local repository. He wants the survey boss to be able to see in the logs that there was a station name change and a surname change. They are quite different changes so should be in separate commits.

Bob pushes to the ASF repository: He now pushes these commits up to the ASF repository.

```
bob$ hg push
```

Alice pulls down the latest copy of the survey and sees Bob's change. The survex file she sees in the updated version of the branch contains both of their changes:

```
; Playground Cave Area, NSW
; Survey of Hazard Hole, PG10
; Club: Sydney University Speleological Society
; Instruments: DistoX
; Surveyors: Alice Jones and Bob Smith
; Date: January 2015
; Description: Hazard Hole starts with a steeply descending passage,
;   ending in a nice chamber.

*calibrate declination -12.3

*begin pg10
*fix 1 0 0 0
*data normal from to tape compass clino
; From  To     Dist(m)    Bear       Elev   Comments
PG10    2      1.11       31.5      -14.0   ; 1 = tag at entrance
 2      3      1.89       86.5      -16.0   ; 2 = stal on roof
 3      4      5.83       72.0      -71.0   ;
 4      5      5.71       68.5      -60.0
 5      6     11.95       64.0      -40.0
 6      7     22.39       44.5      -43.5
 7      8     16.12       46.0      -54.5
 8      9     13.46       38.0      -22.5
 9     10      5.65       31.6      -6.5
10     11      2.49       72.0      -35.5
11     12     15.87       10.5      -2.0
12     13      5.30      125.0       0.0
13     14     16.63      128.0      -4.5
14     15      8.82      188.5      -6.0
15     16      5.72      107.0      -7.0
16     17      1.39       84.0      -16.5
17     18     16.92      171.5      +1.5
18     19      4.81      144.5      -6.5
19     20      1.89       86.0      +8.5
20     21     11.45      159.0      +5.5
21     22a    21.82      110.0      -0.5
21     22b    12.0       165.0       0.0
21     22c    11.16      213.5      -3.0
21     22c    10.80      256.0      -6.0
21     22d    09.60      296.0      -4.0
21     22e     9.50       40.5       0.0
21     22f    10.05       86.5      +1.0
21     roof   15.2         -        up
*end pg10
```

At this stage there have been a few changes to the survey; both changes done by different people at sufficiently different times so that there were no conflicts. Now though it's a weekend, there are no caving trips, and both Alice and Bob decide to do some editing.

Alice and Bob both make a change to the same line(s). Note: both did the right thing and did a pull prior to making their edits so they both started off from the same parent version.

Here Alice commits and pushes first to the ASF repo. Bob commits then tries to push.

```
bob$ hg push
pushing to http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground
searching for changes
abort: push creates new remote head 31f119d65482!
```

```
(you should pull and merge or use push -f to force)
$
```

The push failed. Bob needs to do a pull first (to pull down Alice's changes) and then he will need to merge any conflicts.

```
bob$ hg pull
pulling from ASF server
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files (+1 heads)
(run 'hg heads' to see heads, 'hg merge' to merge)
$
```

Note: If you were able to run the hg heads command on the ASF server repo this is what you would see – one head being Alice's last commit.

```
ASF$ hg heads
changeset:   38:b030cbb42571
tag:         tip
user:        Alice Jones
date:        Wed Jan 21 17:54:59 2015
summary:     modfified
```

Just for interest, Bob checks the heads on his repo. Yep there are now two heads; one from his change to the parent and one from Alice's change to the parent that he just pulled down.

```
bob$ hg heads
changeset:   39:b030cbb42571
tag:         tip
parent:      37:6668487c6c13
user:        Alice Jones
date:        Wed Jan 21 17:54:59 2015
summary:     modfified

changeset:   38:31f119d65482
user:        Bob Smith
date:        Wed Jan 21 17:55:08 2015
summary:     changed


Bob$ hg glog | more
```

(Notice that instead of using "hg log" I'm using "hg glog"; this additional command is short for graphlog and it's a Mercurial extension enabled in your hgrc config file.)

Before the merge:

```
--- ASF Server ---                          -- Bob ----------------
@  changeset:  38:b030cbb42571               @  changeset:  38:31f119d65482
|  tag:        tip                          |  tag:        tip
|  user:       Alice Jones                  |  user:       Bob Smith
|  date:       Wed Jan 21 17:54 2015        |  date:       Wed Jan 21 17:55 2015
|  summary:    modfified                    |  summary:    changed
|                                           |
|  changeset:  37:6668487c6c13              |  changeset:  37:6668487c6c13
|  user:       Bob Smith                    |  user:       Bob Smith
|  date:       Wed Jan 21 09:43 2015        |  date:       Wed Jan 21 09:43 2015
|  summary:    Added surveyors surnames.    |  summary:    Added blackberry hole survey.
|                                           |
```

After a pull from the client:

```
--- ASF Server ---                          --- Bob ----
@  changeset:   38:b030cbb42571             o  changeset:   39:b030cbb42571
|  tag:         tip                         |  tag:         tip
|  user:        Alice Jones                 |  parent:      37:6668487c6c13
|  date:        Wed Jan 21 17:54 2015       |  user:        Alice Jones
|  summary:     modfified                   |  date:        Wed Jan 21 17:54 2015
|                                           |  summary:     modfified
|                                           |
|                                           | @  changeset:   38:31f119d65482
|                                           |/   user:        Bob Smith
|                                           |    date:        Wed Jan 21 17 2015
|                                           |    summary:     changed
|                                           |
o  changeset:   37:6668487c6c13             o  changeset:   37:6668487c6c13
|  user:        Bob Smith                   |  user:        Bob Smith
|  date:        Wed Jan 21 09:43 2015       |  date:        Wed Jan 21 09:43 2015
|  summary:     Added surveyors surnames.   |  summary:     Added blackberry hole survey.
|                                           |
```

```
$ hg merge
```

In your “merge” program do a manual merge to resolve the problem.

Changes now on Bob’s machine are:

```
Bob$ hg st
  M W384.svx

Bob$ hg diff
diff -r 31f119d65482 W384.svx
@@ -22,5 +22,6 @@
 10    9    1.35      0     -55   ; 10 = Non-descript point on wall, don't bother to find it.
 11    10   1.93      80    -45   ; 11 = Tip of rock.
-9     9A   5         60     +5   ; 9 -> 9A Low grade, dist and elev approximate only.
+;9     9A   5         60     +5   ; 9 -> 9A Low grade, dist and elev approximate only.
+9     9A   4.9       60     +5   ; 9A = end of extension
 7     12   4.50      50    -23   ; 12 = Lowest tip of rock pendant on NW wall
 12    13   5.07      30    -18   ; 13 = Tip of stalactite in freize of stals before dig.

Bob$ hg ci -m 'merged wih Alices changes'
```

```
Bob$ hg push
pushing to http://suss.caves.org.au/cavehg/oHet2DAdGe1Y/playground
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
Bob$
```

```
Bob$ hg heads
changeset:   40:47071f8c5dbc
tag:         tip
parent:      38:31f119d65482
parent:      39:b030cbb42571
user:        Bob Smith
date:        Wed Jan 21 20:36 2015
summary:     merged wih Alices changes
```

```
--- ASF Server ----
o    changeset:  40:47071f8c5dbc
|\   tag:        tip
| |  parent:     39:31f119d65482
| |  parent:     38:b030cbb42571
| |  user:       Bob Smith
| |  date:       Wed Jan 21 20:36 2015
| |  summary:    merged wih bills changes
| |
| o  changeset:  39:31f119d65482
| |  parent:     37:6668487c6c13
| |  user:       Bob Smith
| |  date:       Wed Jan 21 17:55 2015
| |  summary:    changed
| |
@ |  changeset:  38:b030cbb42571
|/   user:       Alice Jones
|    date:       Wed Jan 21 17:54 2015
|    summary:    modfified
|
o  changeset:  37:6668487c6c13
|  user:       Bob Smith
|  date:       Wed Jan 21 09:43 2015
|  summary:    Added surveyors surnames.
```

```
--- Bob ----
@    changeset:  40:47071f8c5dbc
|\   tag:        tip
| |  parent:     38:31f119d65482
| |  parent:     39:b030cbb42571
| |  user:       Bob Smith
| |  date:       Wed Jan 21 20:36 2015
| |  summary:    merged wih bills changes
| |
| o  changeset:  39:b030cbb42571
| |  parent:     37:6668487c6c13
| |  user:       Alice Jones
| |  date:       Wed Jan 21 17:54 2015
| |  summary:    modfified
| |
o |  changeset:  38:31f119d65482
|/   user:       Bob Smith
|    date:       Wed Jan 21 17:55 2015
|    summary:    changed
|
o  changeset:  37:6668487c6c13
|  user:       Bob Smith
|  date:       Wed Jan 21 09:43 2015
|  summary:    Added surveyors surnames.
```

The survey data entry and editing is finished. There is a copy of the data in the repositories of Alice, Bob and the ASF server. This also contains a history of the changes to the data that these people have made.

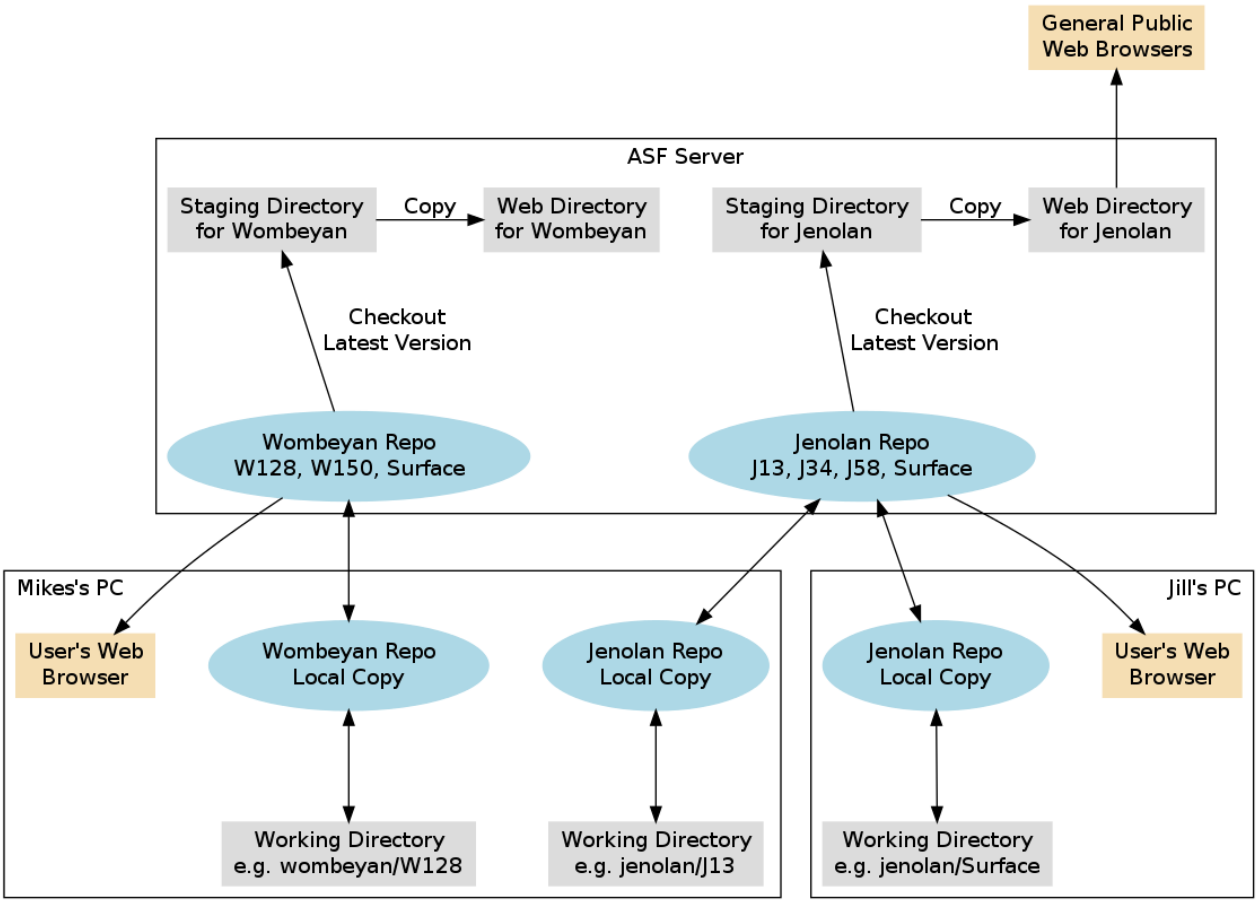
D How the System Works

The definitive version of any set of archived files resides on the ASF server. These are the files that will be listed on the club's archive website, and they may or may not be downloadable from the website. The access policy is defined in the default branch of the repository, editable by the club archive administrator only. The user can override the repository access policy on a branch-by-branch basis, or within a branch on a folder-by-folder basis.

The files on the website are updated from the club archive once per day, usually late at night. The `metadata.ods` file in each branch and in each folder within each branch defines the files and file properties stored by the branch. When a user wishes to work on a branch of the archive, they need to begin by cloning the repository to their local machine Pull command in the version control system.

The Mercurial version control system tracks uploads to the ASF server, downloads from the ASF server, and changes to the branches and repositories on the ASF server. It also maintains the *local* repository information, on a user's machine. This includes any additions or removals of files from the branch and any edited files within the branch. Mercurial maintains historical versions of the repository on the ASF server and provides them on demand.

The repositories are entirely club-based, and only people authorised by the club archive administrator will have access to the version control system. Within the club, more than one person may have a local repository for a particular caving area, and all users can Push changes to the ASF server.



The CAVE system overview. The blue ovals are Mercurial repositories. Repositories reside under a club's domain on the ASF server. They can be cloned onto an individual's local PC or laptop. Changes made there are pushed up to the ASF repositories.