In this module, you will learn

- the purpose of version control systems
- version control concepts and terms
- how to version control your files with Mercurial

What Is Version Control

Have you ever wished life had an UNDO button?

“Boy I wish I could undo that speeding ticket...”

“If only I hadn’t said that to her...”

Have you ever gotten lost trying to find someplace you knew how to get to just a few weeks earlier? Or forgotten how to work a math problem on a test even though you could do it on the homework the week before?

While no computer program will get you out of that ticket or help you figure out why you forgot that math problem, we can solve similar kinds of problems affecting the documents and files we create and use. A version control system keeps track of the history of the changes you make to your files along with summary comments describing each change.

version control system: a software system for tracking and managing file changes
Why Version Control?

Why do we need version control? Our documents and files live in a dangerous world. While viruses and spyware are the sensational, newsworthy dangers, their dubious glamour belies the reality that they are far less likely to pose a significant threat to your files than simple, mundane user error.

Have you ever lost work by

- clicking the wrong menu item or dialog button,
- deleting a file you thought you no longer needed, or
- overwriting old, obsolete changes only to later discover they weren’t so obsolete after all?

A version control system can help to mitigate all of these dangers. While it won’t eliminate the possibility of losing files or data, version control allows you to

- undo changes if you mess something up, even if you have already saved, closed the program, or even rebooted
- view a complete history of your changes with annotations describing the process
- collaborate with others safely without the danger of one person’s changes overwriting another’s

In this module we will focus on the first two of these features. We’ll be using the TortoiseHg\footnote{Hg is the chemical symbol for the element mercury.} version of Mercurial, a powerful yet easy to use modern version control system.
Concepts and Terms

Before we start using version control, let’s take a moment to explore the key concepts and terms you’ll need to know.

The two main concepts in a version control system are the working directory and repository. The working directory is the copy of the files you make changes to; it is just a normal folder with normal files that are viewed and modified the same way as if you were not using version control.

**working directory**: the editable copy of the files in a version control system

For example, suppose you were working on an assignment for your web class and had created a folder to hold the assignment’s files. That directory and its files would be your working directory, which might look something like this:

```
Documents\fall2009\CS2251\assignment1\assignment_goals.txt
me.jpg
my_first_web_page.html
```

The repository is where the real magic happens: it tracks the history of your changes to the working directory. Each revision in the repository history contains a complete snapshot of your project’s working directory at a given point in the past. The revisions are also annotated with a comment describing the changes that occurred between that revision and the one before it. When you look at the repository history, the comments help you understand how the project has changed and grown through time. The comments also help you to find the right earlier version if you need to undo something.

**repository**: the history of changes to the files in the working directory

**revision**: a snapshot of an earlier version of the files in the working directory

Here is the complete revision history showing the repository snapshots, also called changesets, including the list of modified files and the comment describing the changes in that revision:

```
changeset: 0
files: .hgignore assignment_goals.txt
description: added description of course goals

changeset: 1
files: assignment_goals.txt my_first_page.html
description: Added first web page with some basic starter text

changeset: 2
files: me.png my_first_page.html
description: Added a picture of me to the web page
```

Figures 1–3 show the first, second, and third revisions for the hypothetical web assignment:
Figure 1: First revision: adding initial files to the repository.

Figure 2: Second revision: Added first web page with some basic starter text and updated goals description

Figure 3: Third revision: Added image to the webpage
Who Are You?

Before we get started using version control, you will need to let Mercurial know who you are. Mercurial insists that you identify yourself when committing your work and sharing changes with others, so you will need to set your username in Mercurial’s global settings. If you are working on your own machine and installed TortoiseHg yourself, the installer used the information you entered during the installation process to configure your personal information in its settings. However, if you are on a shared computer or lab machine, you will need to set this information manually before you can continue.

To review your user settings and set your username:

1. On the desktop, 
   
   Right-click → TortoiseHg → Global Settings

2. Click the Commit tab

3. In the Username box, enter your full name followed by your email address in angle brackets, like this:
   
   Your Name <you@email.com>

   You can leave the other settings <unspecified>

4. Click Apply, then click Close.

You should only have to do this once for each new machine you use Mercurial on, e.g. your home PC and the lab machines. After you set it up once, your preferences are saved with your user account files, so Mercurial will know who you are when you login.
Exercise 1: Working With An Existing Repository

As your first experience using version control, let’s take a look at some of the things you can do in an existing repository:

1. Create and open a work location for this assignment, e.g.

   Documents\school\cs\version_control

2. **Clone** the example repository for this assignment. Cloning, as the name suggests, creates a duplicate copy of an existing repository for you to work on, complete with its entire history. By cloning a repository, you can work independently of other people; Mercurial includes other tools for communicating changes with others. To clone the example repository:

   a. Right-click in the **version_control** directory you created and choose

      TortoiseHg → Clone...

   b. The clone dialog appears; you need to change the **Source Path**, which is the location of the repository you want to copy.

      ![Clone Dialog]

      In the Source Path text box, paste in the repository URL:

      http://bitbucket.org/drocco/mercurial_example/
c. Click Clone; a dialog box appears showing the progress of the clone.

†Some versions of Adobe Acrobat may incorrectly replace the _ in the example URL with a space. If this happens, you will get an error, so if you copy-and-paste the URL, double check to make sure that there is an underscore between “mercurial” and “example”.
Figure 4: Working directory after a successful clone.
3. The revision history is a version control system’s most powerful tool. To access the revision history, you use TortoiseHg’s Workbench. Let’s examine the history of this repository:

   a. Right-click in the folder and choose

      Hg Workbench

      Mercurial displays the revision history for the project

   b. Click on Revision 1 to see the changes that were made between it and the initial revision. When you click on a particular revision, Mercurial shows you the differences between the highlighted revision and the one before it.

   ![Hg Workbench showing the revision history for the example project. Each changeset or revision is shown on a line in the revision history, with the most recent change at the top. Each change includes a summary description of the change, the user who made the change, and the date of the change. The bottom half of the window shows the details for the selected changeset. On the left, TortoiseHg lists the files that were changed in the selected revision and indicates if the file was modified (M), added (A), or removed (R). The right side shows the differences between files from this revision to the previous one.](image)

   Figure 5: Hg Workbench showing the revision history for the example project. Each changeset or revision is shown on a line in the revision history, with the most recent change at the top. Each change includes a summary description of the change, the user who made the change, and the date of the change. The bottom half of the window shows the details for the selected changeset. On the left, TortoiseHg lists the files that were changed in the selected revision and indicates if the file was modified (M), added (A), or removed (R). The right side shows the differences between files from this revision to the previous one.
4. Not only can you examine the revision history, you can also go back to any previous version Mercurial knows about.

   a. From revision log, click on Revision 1, then Right-click and choose “Update”

   b. Click the “Update” button in the resulting dialog to confirm your choice

   c. Examine the working directory; note the significant changes:

       • me.png was added after this revision and is no longer present in the working directory

       • the other files have also been edited

   d. Repeat these steps for Revision 0; don’t forget to examine the changes in the working directory

   e. Finally, update the working directory back to Revision 2
Making Changes

Now that we’ve seen Mercurial’s tools for examining the history of a working directory, it is time to get to the core of version control: making changes.

Exercise 2: Making Changes

1. In your working directory, open the text file assignment_goals.txt using Notepad++ or your favorite text editor.
2. Edit the file: add a new paragraph at the end:
   
   This is my first edit. I’m excited about learning to manage changes to my files with Mercurial!

3. When you are satisfied with your changes, save your work by clicking File → Save
4. View the revision log for the project: Right-click in the folder and choose Hg Workbench

Ummmm, nothing happened! Our changes aren’t there! What’s going on??!
How a Version Control System Tracks Your Changes

A version control system tracks the changes to your files, but it does not do so automatically. To track a change, you must save the revision to the repository; this is called committing (sometimes called checking in).

While it might not seem like it at first, this is actually a Good Thing™. With a manual commit process, the version control system allows you to decide when you’ve made a coherent, significant change and gives you the opportunity to describe the change in a commit comment. Note that significant changes don’t necessarily have to be large: if you are working on an English paper, for example, you might save your work and commit after every couple of paragraphs.

If you look closely, you’ll see that Mercurial tells you which files in your project have changed by placing an exclamation icon next to the file:

![Figure 6: The file icons show which files are up to date with a check mark (green) and which files have been modified and need to be committed with an exclamation mark (red).](image)
Exercise 3: Committing Your Change

To commit your change as the latest revision in the repository:

1. Right-click in your working directory and choose
   
   HG Commit...

   to bring up the commit dialog

2. Enter a commit comment. The comment should contain a brief summary of the change that is being committed.

   Good commit comments are a vital part of effective version control. Without them, it is much more difficult to understand the revision history and find earlier versions of your projects. Commit comments don’t need to be long, but do take the time to write a descriptive summary of all your changes.

3. When you are satisfied with your comment, click “Commit” to save your changes as a new revision in the repository.

4. Close the commit dialog, then bring up the change log (Right-click → Hg Workbench) to see your changes. Click on the revision you just committed to see its changes. Take note of your new commit comment and the diff pane that shows all the changes that were part of the commit.
Repository Management

Initializing A New Repository

For many of your projects and assignments, you will not have an existing repository to start from, so you’ll need to make a new one before you start the project. In Mercurial, each project or assignment has its own repository and working directory. Mercurial makes it very easy to create a new repository.

1. First, create the working directory for your project or assignment.
   a. Navigate to the folder that will hold the new working directory. For example, if this is a class assignment, you would navigate to the folder containing your assignments for that class. Right-click in the folder, then choose New → Folder to create the new working directory.
   b. Give the folder a name that describes what the project is; press **Enter** to save the name
   c. Double-click the folder you just created

2. To create the repository:
   a. Right-click → TortoiseHg → Create Repository Here
   b. In the initialization dialog, click “Create” to create the repository
   c. Close the initialization dialog

Mercurial has now created the repository—the “hg” folder—and your working directory is now prepared for version control.

Create your files as you normally would: use Word to create Word documents, Photoshop for images, or whatever program is required by your project or assignment.
Adding Files

Once you’ve created a document and **before you sink 6 hours of work into it**, you need to add it to the repository for revision tracking:

1. Select the files you want to add in your working directory
2. Right-click the selected files and choose TortoiseHg → Add Files

   
   ![TortoiseHg menu](image)

   TortoiseHg displays a confirmation dialog showing the current status of the selected files. Click Add to add the files.

   ![TortoiseHg confirmation dialog](image)

   TortoiseHg updates your working directory to show the files that have been marked for addition.
**Warning**

Your files are marked for version control but are not yet saved to the repository!

Add and commit are two different operations in Mercurial (and most other version control systems). It is important to remember that added files will not be tracked in the revision history until they are committed!

3. Commit your changes as you normally would
   
a. Right-click → HG Commit...

b. Add a commit comment. For initial repository check ins, a comment like “initial history of Flurbovia essay” works nicely

c. Click commit to add your new document to the repository
Exercise 4: Creating a New Project From Scratch

Let’s practice what we’ve learned by creating a new project entirely from scratch.

1. Create a new working directory to hold your project:
   
   Documents\school\cs\first_repository
   
The exact name and location does not matter, so use a name that conveys the purpose of the directory to you and place it in a logical location in your file organization scheme.

2. Create the Mercurial repository: inside your new, empty working directory,
   a. Right-click → TortoiseHg → Create Repository Here
   b. In the initialization dialog, click “Create” to create the repository
   c. Close the initialization dialog

3. Create a new text document
   
   Right-click → New → Text Document

   Name the file README.txt

4. Edit the file, adding your name and the date followed by a new line

5. Add the file to version control

6. Commit the new file

7. Edit the file, add a new paragraph summarizing the purpose of this assignment. Commit your changes when you are finished.

8. Add a second file to your working directory.
   a. Create a new file or copy it from another location. The file can be anything: another text file, a picture of you, a doodle you drew in Paint, etc.
   b. Add the file to version control but don’t commit it yet
   c. Edit the README.txt file, adding a paragraph describing the file you just added
   d. Save and commit your work

9. Make at least one other change of your choosing. Don’t forget to commit when you are done.
Summary

Congratulations! You’ve learned the basics of version control and a good working set of Mercurial’s commands, including:

- the motivation for version control systems
- main ideas in version control
- how to use Mercurial and TortoiseHg to manage changes to your projects and files

Key Points

- the **working directory** contains the editable copy of the files for a project
- the **repository** holds the history of changes to the files in the working directory
- a **revision** or **changeset** is a snapshot of an earlier version of the project stored in the repository
- the **changelog** shows the detailed history of the revisions to the project
- **commit** changes to save them as a new revision in the repository history; include a descriptive commit comment to record the purpose and scope of the changes

Additional Resources

Mercurial is a powerful version control system; while you’ve learned enough to perform many critical version control tasks, there are plenty of other features and tools for you to explore and learn. These resources are great places to look if you have questions or want to learn more:

- TortoiseHg Quick Start Guide: [http://tortoisehg.bitbucket.org/manual/2.0/quick.html](http://tortoisehg.bitbucket.org/manual/2.0/quick.html)
- Source Control HOWTO: [http://www.ericsink.com/scm/source_control.html](http://www.ericsink.com/scm/source_control.html)