

Introduction to Unmanned Ground Vehicles

Instructor: Micho Radovnikovich

Email: mtradovn@oakland.edu

ECE 495/595 – Winter 2017

Mercurial Concepts

Mercurial repositories, also known as Hg repositories have many features that allow large teams of people to easily manage project code and documents. In this course, only a few of these features will be necessary to use. This document outlines the fundamental operations required for the course.

1. Terminology

There are many terms with regard to repositories that can be confusing to those unfamiliar with them. The ones discussed here are specific to Mercurial, but other systems usually have similar functionality, even if they have different official names.

Clone

Cloning a repo means making a local copy of it on an individual computer, along with the entire history of past revisions. Doing this allows somebody to make local changes to the repository on their own machine, pending future synchronizations with the central repository.

Fork

Forking a repository means to make a completely new central repository by first copying the original. Mechanically, this fork is completely independent from the original central repository it was forked from. This means that any changes made to the forked repository will not have any effect on the original. At a later point, the manager of the fork can submit a **pull request** to recombine pieces of the fork into the original, but this is not necessary for the course.

Commit

After a user makes changes to their local cloned copy of a repository, the user **commits** the changes to make a snapshot of the present state of the code and documents. After

committing changes to the repo, that specific version of the files can be returned to at any time. However, this only affects the local cloned copy of the repository, and does not update the central repo.

Working Directory

The working directory is simply the current state of the repository files, including all uncommitted changes.

Revert

Before committing any changes, the user can **revert** the changes to any specific file back to the previous version as it was before changes were made. The revert operation only reverts back to the latest committed revision. To change to a newer or older revision, the **update** operation must be performed instead.

Update

At any time, the user can **update** the working directory to any previously committed version. This includes both updating to a past version and updating to a newer version. However, unlike reverting, you cannot update just a selection of files; you have to update the entire working directory to a specific revision.

Push

To update the central repository with all the local committed revisions, the user **pushes** the changes to the original repository. After this, anyone else who has cloned the original repo will be able to see the changes.

Pull

If someone else has pushed revisions to the central repository, **pulling** will import all those new revisions into the local cloned copy of the repo. This does nothing to change the working directory. To actually change the files in the working directory on the local machine, an update operation must be performed.

Merge

If there are local changes to the repository after a pull operation, the pulled revisions must be **merged** with the local changes in order to incorporate the pulled revisions. If the

local and pulled changes are mutually exclusive, meaning they operate on completely different files, then merging simply combines the files.

However, if there are conflicts in any files, the user must resolve them before a merge operation can be done. That is, if any given file has been changed locally (by you), and that same file has also been modified and committed to the central repo, Mercurial needs for you to select which version to keep when pulling files into your local repo from the original. Sometimes, the software is able to combine the changes into a single new version of the file.

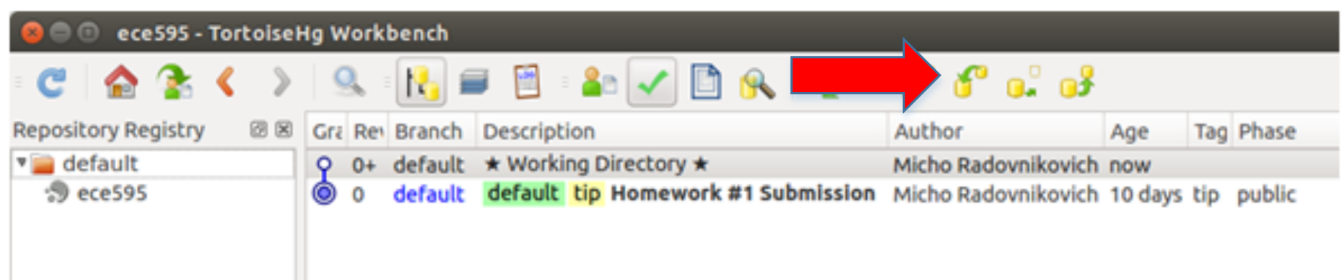
Branch

Branches are separate revision paths from the rest of the repository. Any commits to a repository branch will not affect or interact with any other branch. Branches are typically used for parallel development efforts, or to develop new features that are not desired to be in the main branch yet. A clearer understanding of branches can be gained from the corresponding example in the next section.

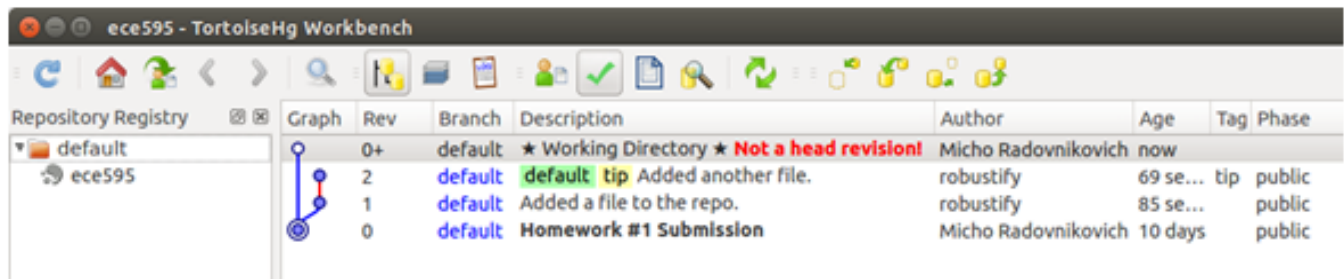
2. Examples in TortoiseHg

Pulling in Changes and Updating

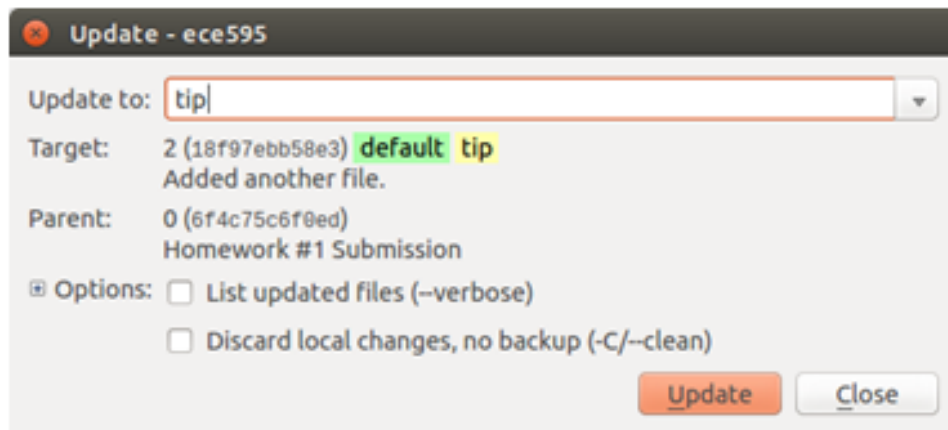
If someone else has pushed changes into the central repository that you want to incorporate into your local copy, you have to pull them first. This is done by first opening the desired repository and clicking on the **Pull incoming changes** button:



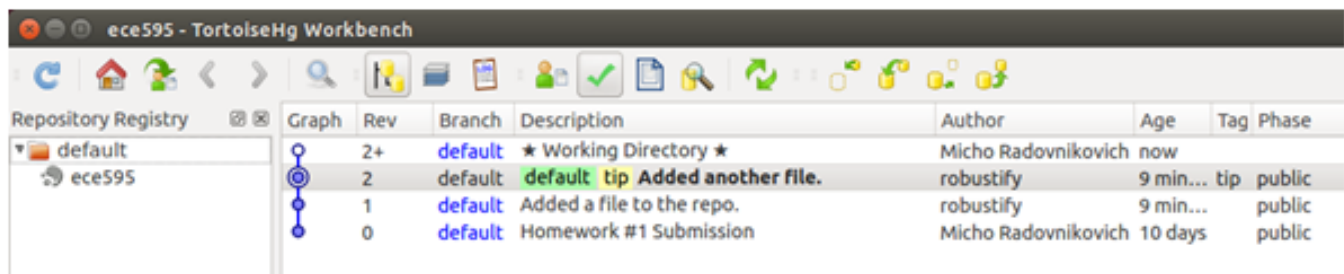
After entering your BitBucket password, the changes will be pulled and the revision graph will be updated. If the graph doesn't change, then there were no changes to be pulled in the first place! For example, you'll see some number of new revisions after you pull:



In this case, someone else added revisions 1 and 2. Notice how the working directory's revision number is 0+. This means that the version we are operating on currently is based on revision 0. After pulling changes, they are now on your local machine, but not yet actually visible in the file system. To actually see the changes in the files on your system, you have to run an update operation. To do so, right-click on the line of the version you want to update to. In this example, we select Revision 2. A screen like this should come up:



After clicking **Update**, you'll see the graph change again:

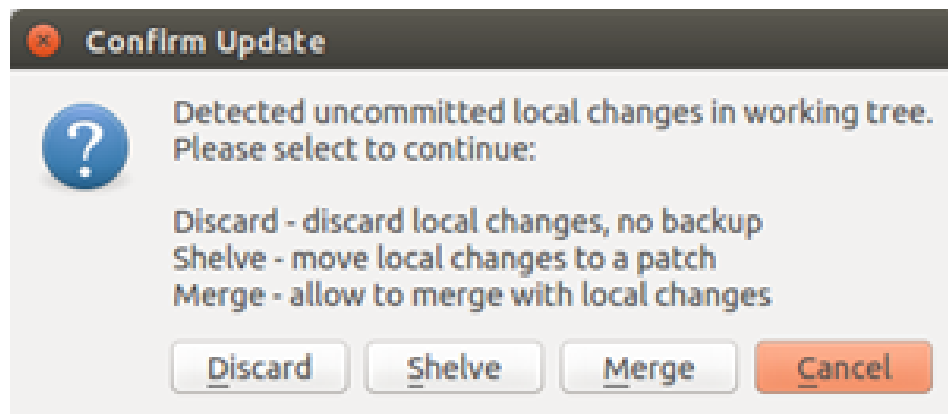


Now, you'll notice that the working directory's revision number is 2+ which means the files on your system are now based on the snapshot of Revision 2 from the central repository.

If you're wondering where the information of all the other revisions is hiding, you'll see a hidden folder called **.hg** in the root directory of your local clone. This magical folder has details of all the changes between the revisions, not just copies of each snapshot. The Mercurial operations simply read the information in this folder to appropriately update the files in the actual directory.

Merging Pulled Changes with Local File Changes

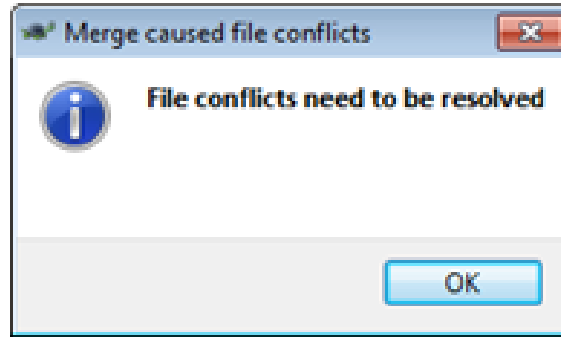
Typically when you pull changes, you'll be in the process of making your own changes to the current version. When this happens, you have to merge the new revisions you pull into your working copy. In this situation, update using the same procedure as in the last example. However, instead of just returning to the main screen after clicking **Update**, you'll get another window:



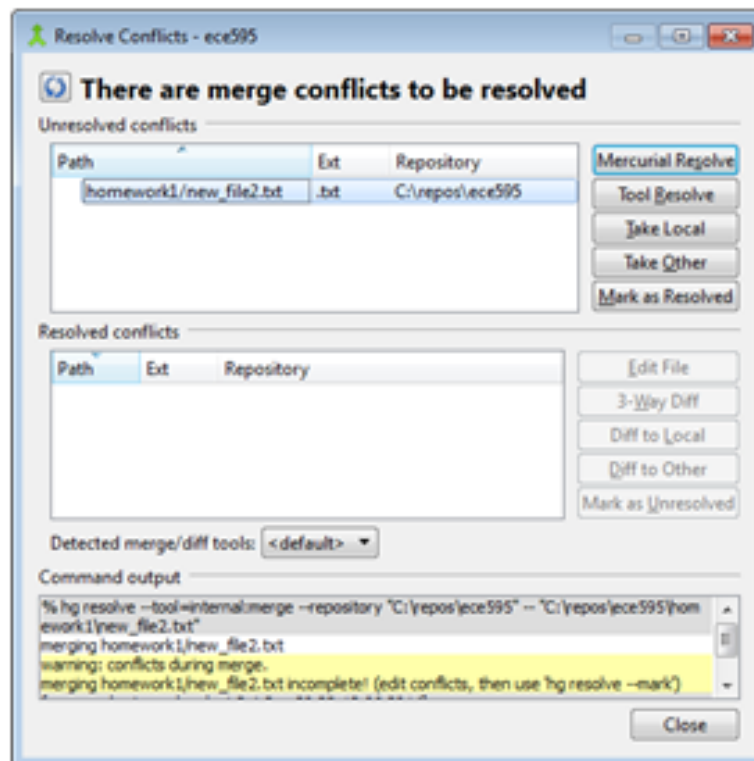
Clicking **Discard** will permanently delete all the changes you made locally (you cannot get them back) before updating to the selected revision. **Shelve** moves the local changes to a patch, which removes them from the working directory, but you can get them back later if you want. **Merge** attempts to update the files without doing anything to the local changes. As long as the files you changed haven't been changed in any of the pulled revisions, then it simply combines the new revisions with the working directory.

Merge Conflicts

If there is a conflict between local changes and pulled changes, you'll see this window after clicking **Merge**:



After clicking **OK**, you'll see a window to resolve the conflicts:

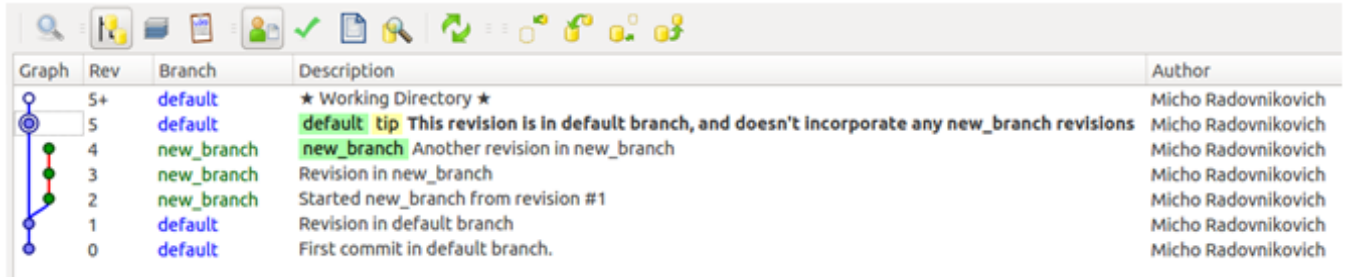


In this case, there is one conflict in the **new_file2.txt** file. **Mercurial Resolve** tries to combine the local and pulled copies using some kind of advanced algorithm. This only works some of the time. **Tool Resolve** opens a graphical interface that allows you to select which changes you want to keep from which revision, but only works with text files.

The above methods are useful in many situations, but most of the time, the **Take Local** and **Take Other** options are used. **Take Local** simply keeps the file the same as the local copy before the merge attempt, and **Take Other** discards the local changes and updates the file to the new version. **Mark as Resolved** doesn't do anything; both versions are kept together with different names indicating which revision they came from.

Branches

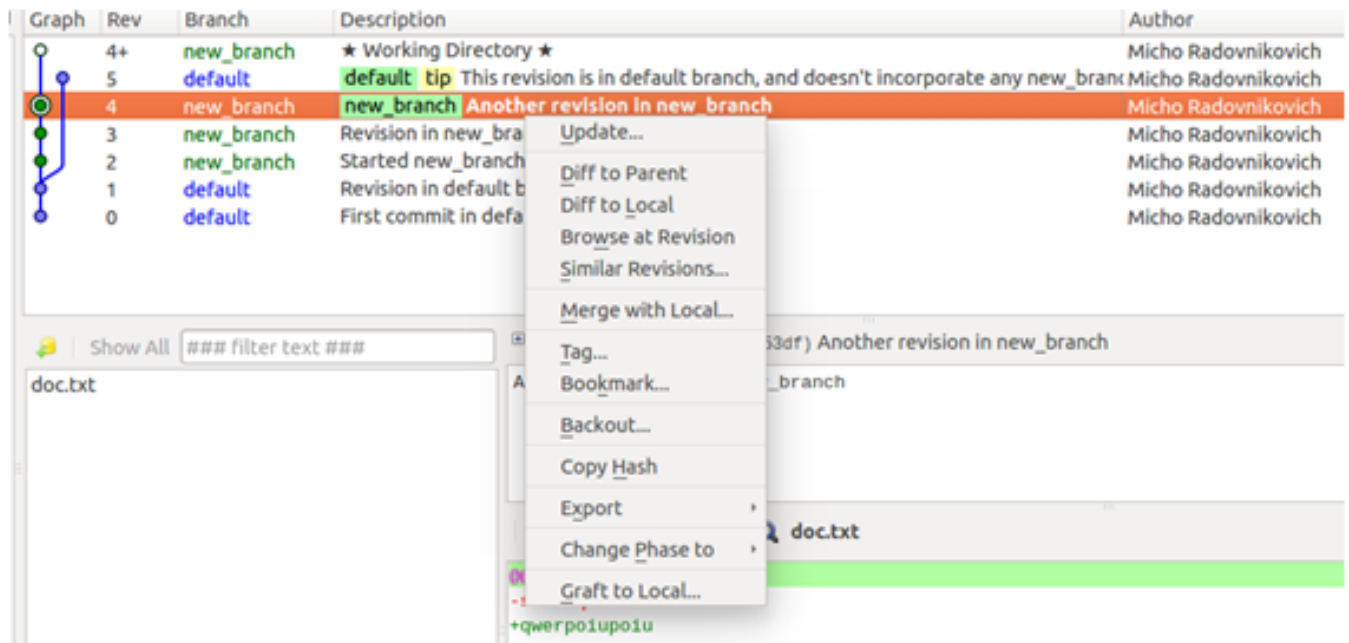
In TortoiseHg, branches are visualized like this:



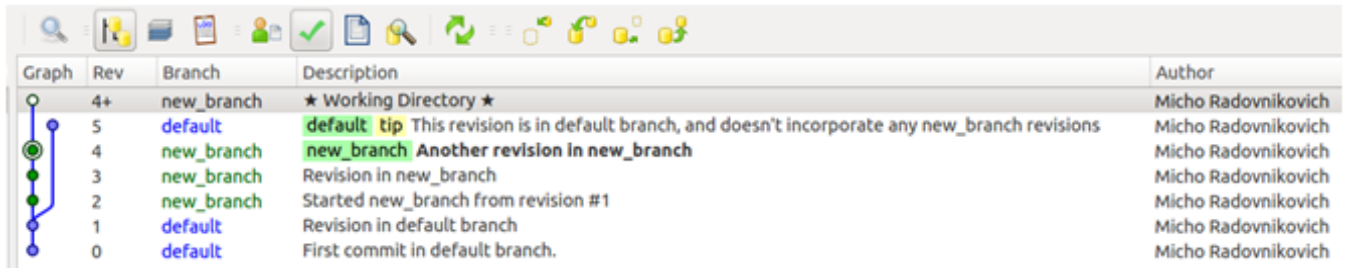
The screenshot shows the TortoiseHg interface. On the left is a revision graph with a blue line representing the 'default' branch and a green line representing the 'new_branch'. The graph shows a sequence of revisions: 0 (default), 1 (default), 2 (new_branch), 3 (new_branch), 4 (new_branch), and 5 (default). Revision 5 is the tip of the default branch. On the right is a table of revisions.

Graph	Rev	Branch	Description	Author
	5+	default	★ Working Directory ★	Micho Radovnikovich
	5	default	default tip This revision is in default branch, and doesn't incorporate any new_branch revisions	Micho Radovnikovich
	4	new_branch	new_branch Another revision in new_branch	Micho Radovnikovich
	3	new_branch	Revision in new_branch	Micho Radovnikovich
	2	new_branch	Started new_branch from revision #1	Micho Radovnikovich
	1	default	Revision in default branch	Micho Radovnikovich
	0	default	First commit in default branch.	Micho Radovnikovich

In this case, there are two branches: **default** and **new_branch**. Currently, the working directory is on the **default** branch, and is based on Revision 5. From the revision graph, it can be seen that this revision bypasses Revisions 2 through 4 completely, since they are in **new_branch**. At any time however, we can update to the **new_branch** revision by right-clicking Revision 4 and clicking **Update**:



Doing so in this case would make the TortoiseHg window look like this:



Graph	Rev	Branch	Description	Author
	4+	new_branch	★ Working Directory ★	Micho Radovnikovich
	5	default	default tip This revision is in default branch, and doesn't incorporate any new_branch revisions	Micho Radovnikovich
	4	new_branch	new_branch Another revision in new_branch	Micho Radovnikovich
	3	new_branch	Revision in new_branch	Micho Radovnikovich
	2	new_branch	Started new_branch from revision #1	Micho Radovnikovich
	1	default	Revision in default branch	Micho Radovnikovich
	0	default	First commit in default branch.	Micho Radovnikovich

Now notice that the working directory is in **new_branch**, and is based on Revision 4. From the graph, it can be seen that all the revisions except 5 are included in the current working directory.

BONUS:

You can now appreciate the following comic! Here, **Git** is another version control system similar but not identical to Mercurial.

