Mercurial I

Mercurial version 2.7.1

Göteborg February 2016

Table of contents

General

About this document

This document contains course overheads and exercises.

About Mercurial I course

The course gives you a working understanding of the Mercurial. After completing the course you will be able to:

- Use Mercurial for version control management
- Understand the best practices used in an implementation and support context for Jeppesen Crew Planning products.

Mercurial

After many years, centralized version control systems have given way to distributed version control systems. Though there is much conjecture about the nature of this evolution, it is clear that distributed projects require tools that understand and function in such a model.

For a more succinct eulogy for CVS et al. look no further than the introduction sections of the book Mercurial: The Definitive Guide

Additional Material

There are additional sources of material for learning Mercurial; among them:

- Mercurial: The Definitive Guide:
    - http://mercurial.selenic.com/guide
    - http://hgbook.red-bean.com/
- The Mercurial project site
    - http://mercurial.selenic.com/wiki/Mercurial/

Slides

Exercises

Introduction

Get started

1. Press the Academy Desktop icon, this logs you on to the Citrix server.
2. Select **Programs > Linux Terminal RHEL6 (EoD8)** from the windows Start menu.

Additional Material

There are additional sources of material for learning Mercurial; among them:

- Mercurial: The Definitive Guide:
  - http://mercurial.selenic.com/guide
  - http://hgbook.red-bean.com/

- The Mercurial project site
  - http://mercurial.selenic.com/wiki/Mercurial/

Basic Mercurial Commands

Exercise 1

Purpose

Work with the basic mercurial commands and settings.
At all times, always use the `hg --help` command.

Exercise 1.1 Getting started

Step 1 Create a repository

1. Go to the directory ~/mercurial_system/user
2. initialize a repository

Hint: hg help

3. view the status of the repository; all files should appear as new/untracked. Note: The command `hg view` will not work since the repository does not yet contain any changesets.

Hint: hg status

Step 2 Adding files to the repository

4. add the `lib` and `crc` directories

Hint: hg add …

5. commit the added files
6. modify at least two of the committed files
7. commit those files, passing the commit message as an argument on the command line

Hint: hg commit –m “” …

Hint: hg help config | less – read about the username issue

Exercise 1.2 The Ad-hoc Mercurial Webserver

Step 3 Start a web server to view the repository from a web browser.

Hint: hg help serve

Step 4 Using a web browser, navigate to the server.

Exercise 1.3 Working with the Repository

Step 5 Ignoring uninteresting things

Several files are created by various programs that are not interesting (python bytecode files, emacs backup files, etc.) we need to instruct hg to ignore them.

8. Start Studio; once it starts, close it down.

Hint: ../mercurial_in_linux_start.sh

9. look at the status of the lib directory; are there files we do not need to track?

10. Add ignore information for the following file fragments:

```
*.pyc
Any files ending with the tilde (~) character
emacs editing/crash files (start with .#)
```

Hint: Both glob and regexp settings may be needed.
Emacs editing/crash files are created by emacs automatically to track a file with unsaved changes. Once saved, these files go away.

11. add the ignore settings and commit them

12. list the files that hg is ignoring

Hint: hg help status

Step 6 Recovering from mistakes - Method #1

13. Modify some files which cause the script command:

```
../mercurial_in_linux_start.sh –E rave_compile_normal.sh
```

to no longer work.

Hint: edit some files in `crc/modules`/

14. undo all changes

Hint: hg revert …

Step 7 Recovering from mistakes - Method #2

Again, make some changes that cause the script command:

```
../mercurial_in_linux_start.sh –E rave_compile_normal.sh
```

to no longer work.

15. commit those broken changes

16. Look at the changeset history

17. Fix the changeset

Hint: hg help commit – look for the flag that modifies the last changeset

18. look at the changeset history again; the broken changes should be “gone”.

Step 8 Recovering from mistakes - Method #3

19. re-commit the broken changes

20. remove the newly committed changeset.

Hint: hg help – look for the command that creates a back out changeset

Hint: The “merge” option is needed in cases where you are performing a `backout` on a changeset that is not at the tip of the repository.

21. Now look at the history; what’s different from the prior command result?

22. add the directory `bin` and commit

23. create a backout changeset for the bin directory addition.

Exercise 1.4 Remove, Rename, Tag

Step 9 Removing a tracked file/directory

24. add **and** commit the directory matador_scripts

25. delete the directory matador_scripts using the shell command rm

Hint: rm -r matador_scripts

26. what is the status in hg? Is this the intention?

27. revert the directory

28. remove the directory again, this time using hg

29. commit the remove change

Step 10 Renaming a file

30. add and commit the directory etc

31. rename the file etc/roles.xml using the shell command mv

Hint: mv etc/roles.xml etc/ex1_3.xml

32. status? Is this ok?

33. revert the directory etc

34. what? Is this what I wanted?

35. remove the file etc/ex1_3.xml

36. Use hg to rename the file etc/roles.xml to ex1_3.xml
37. Looking at the status again, is this better?
    Commit this change.

Step 11 Tagging

38. create a tag for the current state, calling it ex1_3
39. look at the changeset history; notice that a changeset is created automatically to announce the new tag on the prior changeset.
40. list the tags in the repository

Exercise 1.5

Step 12 Wrap-up

41. add and commit the rest of the files in the working directory to your repository.
42. make sure everything is working; i.e. start studio, compile the rules.

Hint: `../mercurial_in_linux_start.sh`

Hint: `../mercurial_in_linux_start.sh -E rave_compile_normal.sh`

43. If the previous step was not possible to execute, if for example the directory `bin` is missing so that the script rave_compile_normal.sh could not be found, can you explain what happened to it?
44. create the tag `ex1_3_final`

Multi-User Repository

This exercise focuses on the multi-user aspects of Mercurial repository management.

For this exercise, the instructor will provide you with the name of the repository (<instructor_repo>) to work with.

Exercise 2

Purpose

To work together with other developers using the same repository.

Exercise 2.1 Clone

Step 1 Go to the directory ~/mercurial_system

Step 2 Clone the teacher repository <instructor_repo>

Hint: hg clone <repo name> <destination>

Step 3 go to the clone base directory

Step 4 View the changeset history

Hint: hg view

Exercise 2.2 Simple Changes

Step 5 add the file `./user/data/config/$USER.txt`, adding some arbitrary information.

Step 6 add and commit those changes. After each person commits, do you see their changes as well as yours? If so, why? If not, why not?

Step 7 push your changes to the main repository

Hint: if hg complains about push creating multiple heads, do: 1) pull; 2) merge; 3) commit; 4) push

Hint: Depending on the other people in the course, you might have to do this multiple times.

Step 8 Once everyone has pushed, pull the changes from the repository.

Hint: hg pull

When you view the changeset history, what do you see? Notice the underlined changeset comment? What does this represent?

Step 9 Update your working directory to the tip revision.

Exercise 2.3 Creating Conflicts

Note Also known as EMH (Extreme Merge Hell) or MHB (Multi-Headed Beast)

Mercurial uses merging as a foundational concept; you will never allow yourself to experience the example below, but practice is nonetheless useful.

Step 10 Modify the etc/users.xml; add a new user with name being your course userid.

Step 11 commit that change and push; if you were quick enough, your changes made it in without a problem. If not, force the push with the -f option.

Step 12 pull from the repository; what do you see? What do we do now?

Exercise 2.4 Recovering from EMH/MHB

So now that you've made a mess, let's clean it up

The *instructor* will perform these operations for the group; steps below are a guideline of what to do.

Note You'll notice that changeset version numbers are not valid across repositories. They are only used for local convenience. In communication about changesets, always refer to the hash key or the revisiohg loghn tag (if applicable.)

Step 13 show the changeset history, and note the revision numbers for each head in the MHB

Hint: hg view &

Now, repeat the following steps for each unmerged head.

Step 14 merge with the first one in the list (the order doesn't really matter, although it might normally be wise to start from the oldest to the newest changeset.)

Hint: hg merge <id>

Step 15 commit the merge

Hint: hg commit –m "Ex 2.4 MHB merging"

After all mergers, make sure the end result is reflecting what you really want to have. Changesets that create conflicts in one file may also have introduced changes to other files that become invalid after the merge.

Exercise 2.5 Working in Sub-Teams

Sometimes it is useful to form sub-teams for a project, and have them share changesets without pushing to the main repository so frequently.

The instructor will have you team up into groups of 2-3.

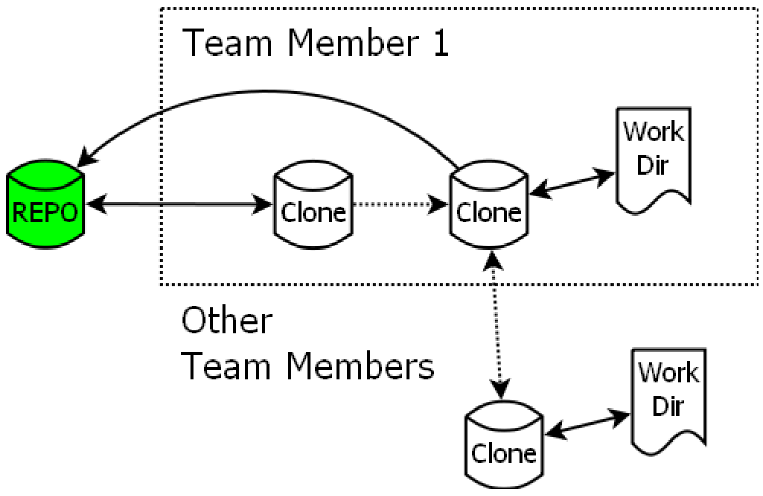

Figure 1: Exercise Working Procedure

Step 16 Each Team member 1 Follow steps 17 and 18; others skip to step 19

Step 17 Synchronize your clone with the main repository.

Hint: pull, merge/update, commit, push

Step 18 Create a new clone from this synchronized clone that you have been working on.

Step 19 Team members 2 and 3: Clone Member #1's new clone.

Step 20 Depending on your group and member number, modify the appropriate file located in `crc/modules` (adding anything you like, as long as the system still compiles):

Team	**Member #**	**File to edit**
1	1	apc_pac_standard
1	2	bc_as_apc
1	3	crew
2	1	crew_pos
2	2	crew_pos_ccr
2	3	crew_pos_ccr_matador
3	1	crew_pos_ots
3	2	crg_basic
3	3	crg_crew_info
4	1	crg_crew_pos

Team	Member #	File to edit
4	2	crg_crew_requirements
4	3	crg_roster
5	1	crg_roster_compact
5	2	crg_roster_cost
5	3	crg_roster_dutybased
6	1	crg_trip
6	2	duty
6	3	duty_ccp
7	1	duty_ccr
7	2	fairness
7	3	hotel
8	1	report_trip
8	2	report_trip_statistics
8	3	rule_exceptions

For each group, push changes back to the member #1's clone.

Step 21 Commit the above changes.

Step 22 Check if there are incoming changes before pushing; if there are, pull and update/merge before pushing

Hint: hg incoming <member #1 repo path>

Hint: hg pull

Hint: hg update/merge

Step 23 Once you have all the repository changesets, push your changes

Hint: hg push [destination]

Step 24 Member #1 now push to the main repository, bypassing their intermediate clone (see Figure 1: Exercise Working Procedure).

Hint: specify the name of the main repository instead of the default name, which is to push to where the clone was created from.

Hint: hg outgoing - check what outgoing changesets will be pushed

Patching and Extensions

This exercise focuses on managing patch sets, and some of the many extensions available in Mercurial.

This is not a complete list, but just a starting point for future investigation.

The same teams from Exercise 2 will be used again.

Exercise 3

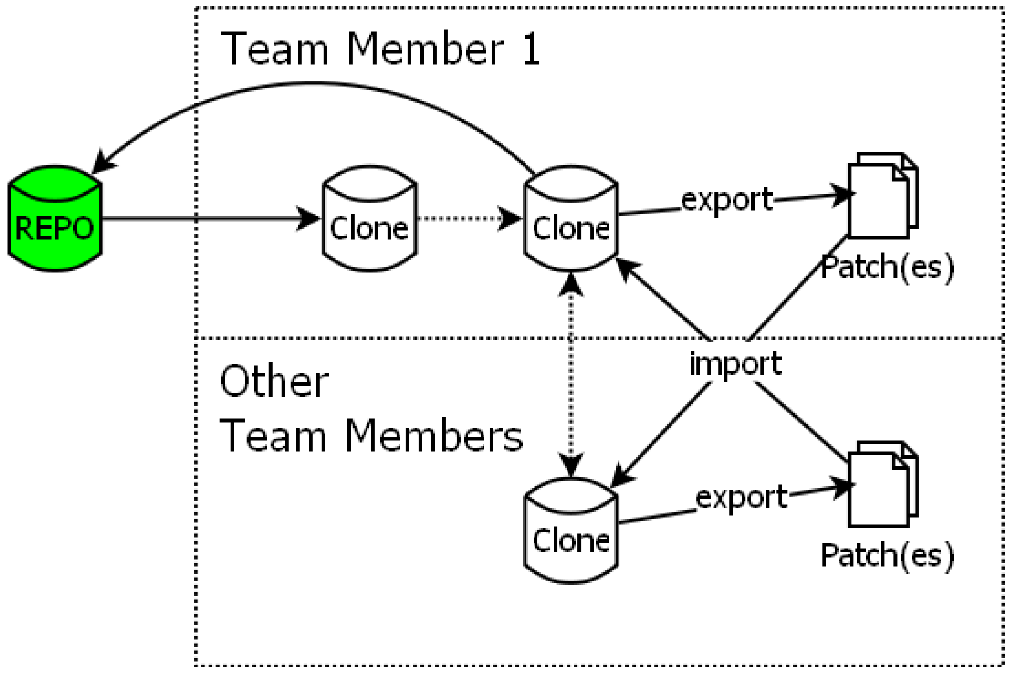

Figure 2: Exercise 3.1 and 3.2 context diagram

Exercise 3.1 Creating Patches

Step 1 Each team member should create 1 or more changesets in their team clone.

Step 2 commit the changes, but do not push

Step 3 Create a patch file

Hint: hg help export

Note Options like %N and %n are very useful when multiple changesets are selected for creating patch files.

Exercise 3.2 Importing Patches

Step 4 Each team member should provide their patches to the other members in their team.

Step 5 import the patches

Step 6 push to the team member #1 repository

Step 7 Team Member #1 pushes to the main repository.

Exercise 3.3 Named Branches

In some cases, named branches are needed (multi-product projects, for example). It is nice to keep everything together, but separate at the same time.

Step 8 Use your original personal (synched) clone from exercise 2

Step 9 Create a named branch called your userid

Hint: hg help branch

Step 10 create two or three changesets on your new branch

Step 11 push the new branch

Hint: This needs to be done with the --new-branch option, since hg needs to be explicitly told that you intended to do this.

Step 12 After everyone is done, perform a pull and view the changeset history. Notice the differences?

Step 13 Update your working directory to use one of the other branches

Hint: hg help update

Exercise 3.4 Graft

Step 14 graft a changeset from one branch to another.

Hint: hg help graft

Exercise 3.5 Closing branches

Step 15 update the working area to the branch you want to be working on

Hint: hg update -C …

Step 16 close the branch

Hint: hg commit …

Exercise 3.6 Extensions

Many extensions are included with Mercurial; many more exist, and could be optionally included.

Step 17 View the list of extensions

Hint: hg -v help extensions

extdiffs - Meld the Graphical way

When the extdiff option is activated in the configuration, you can use any of the configured diff tools instead of seeing diffs in the command shell.

Step 18 Configure extdiff in your ~/.hgrc file, adding meld as a diff tool

Step 19 Produce a diff between 2 branches using the meld extension command.

Hint: hg meld -r <branch 1> -r <branch 2>

Solutions

Exercise 1 Basic Mercurial Commands

Exercise 1.1 Getting Started

```
cd mercurial_system/user

hg init

hg st

hg add lib/ crc/

hg ci –m "Ex 1.1: Adding directories lib and crc to
repository"

hg ci –m "Ex 1.1: changes to two files"

hg view
```

For `ci` to work you need to add your user id to a config file:

```
$HOME/.hgrc

[ui]
username = My Name my.name@mycompany.com
```

For `view` to work you need to activate an extension in a config file (more about extensions later):

```
$HOME/.hgrc

[extensions]
hgk =
```

Exercise 1.2 The Ad-hoc Mercurial Webserver

```
hg serve
```

Copy http link into a webbrowser.

Exercise 1.3 Working with the Repository

e.g: `user/lib/python/menu_commands.pyc`

Create the file `user/.hgignore`

With the following content:

```
syntax: glob
*.pyc
*~
.#*
```

```
hg add .hgignore
```

```
hg ci –m “Ex 1.2: Adding ignore file”
```

```
hg st –i
```

```
hg revert <crc/source/your_file>
```

```
hg commit --amend
```

```
hg backout <changeset number>
```

A new changeset was created that undid the previous changeset, and was automatically committed.

```
hg add bin/
```

```
hg ci –m “Ex 1.3: Adding dir bin/”
```

```
hg backout <changeset number>
```

Exercise 1.4 Remove, Rename, Tag

```
hg add matador_scripts
```

```
hg ci –m “Ex1.3: adding dir matador_scipts”
```

```
rm –r matador_scripts
```

```
hg st matador_scripts
```

```
>! matador_scripts/matador_fast
```

hg complains that the file has been removed by a non-hg command

```
hg revert matador_scripts
hg remove matador_scripts
```

```
hg add etc/
hg ci –m “Ex 1.3: Adding dir etc”
```

Hint: mv etc/roles.xml etc/ex1_3.xml

```
mv etc/roles.xml ex1_3.xml
hg st
>! etc/roles.xml
>? etc/ex1_3.xml
```

The move was not recognized as a move, but rather a remove and create.

```
hg revert
```

```
hg rename etc/roles.xml etc/ex1_3.xml
```

```
hg ci –m “Ex 1.3: Renamed roles file”
```

```
hg tag ex1_3
```

```
hg tags
> tip  8:1b1923…
> ex1_3 7:a35a2…
```

```
hg add
hg ci –m “Ex1.3: Adding remaining dirs”
```

```
hg tag ex1_3_final
```

Exercise 2 Multi-User Repository

The teacher will create an instructor_repo, e.g.:

```
cd </users/teacher/mercurial_system/>
hg clone user instructor_repo
```

Exercise 2.1 Clone

```
cd
```

```
cd mercurial_system/
```

```
hg clone </users/teacher/instructor_repo/>
user_clone
```

```
cd user_clone
hg view
```

All the changesets/tags/etc from the instructor repo have been inherited.

Exercise 2.2 Simple Changes

```
cd data/config
```

```
echo <some arbitrary info> > student#.txt
```

```
hg add student#.txt
```

```
hg ci –m "Ex 2.2: Adding student# config file"
```

No, not yet, since each student is only working with their personal clone.

```
hg push
```

```
hg pull
hg merge
hg ci –m "…"
hg push . . .
```

```
hg pull
```

The underlined changeset is the one you are currently working on.

```
hg upd
```

Exercise 2.3 Creating Conflicts

Edit the file: `etc/users.xml`

```
hg ci –m "Ex 2.3: Adding new user"
```

```
hg push
```

```
hg push –f
```

```
hg pull
hg view
```

Exercise 2.4 EMH / MHB

Teacher Demo.

Exercise 2.5 Working in Sub-Teams

Team member 1:
```
<Synch your user_clone>

hg clone user_clone user_ex2
```

Team members 2 and 3:

```
hg clone <team_member_1>/user_ex2 user_ex2
```

Edit your only your file (this is to avoid conflicts and unnecessary complexity to the exercise).

```
hg ci -m "Ex 2.5: student# changes"
```

```
hg incoming <team_member_1>/user_ex2
```

```
hg pull
```

```
hg update
```

Make sure it still compiles

```
hg push
```

Team Member 1:
```
hg outgoing <path>/instructor_repo
hg push <path>/instructor_repo
```

Exercise 3 Patching and Extensions

Exercise 3.1 Creating Patches

Edit your file from Ex 2.4 again.

```
hg ci -m "Ex 3.1: adding changeset to my file"
```

```
hg export -o "student#_%N_%n"
```

Exercise 3.2 Importing Patches

```
cp <other_member_path>/student#_X_Y .
```

```
hg import student#_X_Y


hg push <team_member_1>user_ex2
```

Team Member 1: `hg push <path>/instructor_repo`

Exercise 3.3 Named Branches

```
hg branch student#
```

```
edit; hg ci –m “Ex3.3: adding changesets to new branch”
```

```
hg push
```

you may need to `pull` before you are allowed to push your new branch.

```
hg pull
```

```
hg view
```

Notice that there are now multiple heads. This is Ok, since they are deliberate branches.

```
hg upd <branch>
```

Exercise 3.4 Graft

```
hg graft <revision>
```

Exercise 3.5 Merge one branch with another

```
hg update –C <branch>
```

```
hg merge <branch>
```

Merge will fetch changes from the other branch into the current one and you will continue to work with the current branch.

Exercise 3.6 Extensions

```
hg –v help extensions
```

In `~/.hgrc`:

```
[extensions]
extdiff =
[extdiff]
meld =
```

```
hg meld –r <branch 1> -r <branch 2>
```

Appendix A – Command Reference

Mercurial Distributed SCM (Version 2.7.1)

List of commands (**bold** items are covered in this course):

Command	Description
add	add the specified files on the next commit
addremove	add all new files, delete all missing files
annotate	show changeset information by line for each file
archive	create an unversioned archive of a repository revision
backout	reverse effect of earlier changeset
bisect	subdivision search of changesets
bookmarks	track a line of development with movable markers
branch	set or show the current branch name
branches	list repository named branches
bundle	create a changegroup file
cat	output the current or given revision of files
clone	make a copy of an existing repository
commit	commit the specified files or all outstanding changes
copy	mark files as copied for the next commit
diff	diff repository (or selected files)
export	dump the header and diffs for one or more changesets
forget	forget the specified files on the next commit
graft	copy changes from other branches onto the current branch
grep	search for a pattern in specified files and revisions
heads	show current repository heads or show branch heads
help	show help for a given topic or a help overview

Command	Description
identify	identify the working copy or specified revision
import	import an ordered set of patches
incoming	show new changesets found in source
init	create a new repository in the given directory
locate	locate files matching specific patterns
log	show revision history of entire repository or files
manifest	output the current or given revision of the project manifest
merge	merge working directory with another revision
outgoing	show changesets not found in the destination
parents	show the parents of the working directory or revision
paths	show aliases for remote repositories
phase	set or show the current phase name
pull	pull changes from the specified source
push	push changes to the specified destination
recover	roll back an interrupted transaction
remove	remove the specified files on the next commit
rename	rename files; equivalent of copy + remove
resolve	redo merges or set/view the merge status of files
revert	restore files to their checkout state
rollback	roll back the last transaction (dangerous/deprecated)
root	print the root (top) of the current working directory
serve	start stand-alone webserver
showconfig	show combined config settings from all hgrc files
status	show changed files in the working directory
summary	summarize working directory state
tag	add one or more tags for the current or given revision
tags	list repository tags
tip	show the tip revision
unbundle	apply one or more changegroup files
update	update working directory (or switch revisions)
verify	verify the integrity of the repository
version	output version and copyright information

Appendix B – CVS Command Cross-Reference

This is a simple cross reference to help facilitate the transition from CVS to Mercurial.

<u>THIS DOES NOT IMPLY THAT YOU SHOULD REPLICATE CVS BEHAVIOR USING MERCURIAL COMMANDS!</u>

Note This is not intended to be a comprehensive list, but should cover the most commonly used commands.

CVS	HG	Comment
add	add	
-	addremove	add new files, remove missing files
annotate	annotate	
-	backout	Remove a committed changeset
checkout	clone	Begin working with a repository
commit	commit + push	
diff	diff	
(using tkcvs)	meld, or kdiff3, or tkdiff, or other.	visual diff
export	archive	Create an un-tracked copy of the repository working files at the specified revision.

init	init	
log	log	
remove	remove	
-	rollback	undo the last command action.
status	status	
tag	tag	
tag -b	branch	
update	pull + update pull + merge	Depending on repository state, either an update or a merge will be required.