Excercise 00: Mercurial

Dr. Günter Kolousek

Just a few words at first: **Don't** print this file and, hence, save a tree!!! It's not necessary to print this document since it is formatted to be viewed on-screen.

1 Installation

1. If you are currently not really happy with your text editor of choice, I suggest trying "Sublime Text". It's a very performant, really nice and nifty editor for Linux, Windows, and Mac OSX. Take a look at **Sublime Text**! Admittedly, it is a commercial product and, therefore, it costs some money (currently 70USD) but it can be evualated freely as long as you are not annoyed when messages pop up from time to time.

   Other recommandable editors are:

   – **geany** A "standard" GTK-based editor which is easy to use and can be configured easily.
   – **jEdit** A Java-based editor with many plugins available.

- **SciTE** A editor for Windows and Linux which is small but quite sophisticated.
- **emacs** Are you a real hacker? Then, maybe you are qualified using this really cool and powerful tool!

Of course, a full-blown IDE like Netbeans or Eclipse is not the worst choice if you feel already comfortable with. A really nice IDE specifically for C++ and Qt is QtCreator.

2. Install Python using your favorite package manager.

Using a Debian or Ubuntu-based system, the following commands will do the trick:

```
sudo apt-get update
sudo apt-get install aptitude
sudo aptitude install python
sudo aptitude install python-dev
```

Using an ArchLinux-based system, the following command will do it:

```
sudo pacman -Syu
sudo pacman -S python
```

3. Install Mercurial. Using Ubuntu or Debian or something comparable, we have to install Mercurial manually because the version in the repository is usually quite old. If you an ArchLinux or Manjaro user, you are on the bright side of life because their repositories are usually rather up-to-date. Therefore, you can use the package manager:

   ```
   sudo pacman -S mercurial
   ```

   If you are not using such a linux distribution, you have to install the source and install it manually by typing the following commands:

   ```
   cd /tmp
   wget http://selenic.com/hg/archive/stable.tar.bz2
   tar xf stable.tar.bz2
   cd Mercurial-stable
   sudo python setup.py install
   ```

Now, test your installation:

```
cd
hg version
```

4. Configure Mercurial: In your HOME directory create a file `.hgrc` with the following content:

```
[ui]
username = FNAME LNAME <ixxxxx@student.htlwrn.ac.at>
# the next configuration specifies your preferred editor
# you can omit this configuration if you are happy with
# the default editor of your system.
editor = emacsclient -n
# the next line specifies the merge tool used
# we will install it a bit later
merge = meld
[extensions]
progress =
graphlog =
```

Substitute FNAME by your first name, LNAME by your last name, and ixxxxxx by your student number. Now, it's time to check your installation and configuration: `hg debuginstall` should print "no problems detected".

5. Install the visual diff and merge tool "meld" the usual way, e.g. `sudo aptitude install meld` or `sudo pacman -S meld` or using the graphical frontend of your package manager.

6. If you are new to Mercurial: It's time to learn about this DVCS (Distributed Version Control System):

   – **German Tutorial**

   – **Hg Init**

   Later on, we will do some excercises but for now that's enough!

7. If you want you can install TortoiseHg. It's a graphical frontend to Mercurial. You don't really need it, neither for this excercise nor for anything else. But if you insist, it needs some additional steps if you are on Ubuntu or Debian box: Go to **DownloadLinuxPackages** and follow the appropriate links. There, you will also find some documentation for installing.

   On ArchLinux or Manjaro it's a little bit easier:

```
sudo yaourt tortoisehg
```

2 Learning the Basics of Mercurial

1. Create a repository in the following way:

   ```
   hg init repo1
   ls
   cd repo1
   ls -a
   ```

   This will create a directory named "repo1" that is your new repository. It is called the working directory because it will contain all files you want to deal with.

Change into the repository and you will find a hidden directory ".hg" (called the store) inside that contains the history of your repository as well as some adminstrative files. Don't touch this directory!

2. It's time for your first changeset. A changeset is an atomic collection of changes to files in the repository. Such a changeset leads to a new revision of the repository:

   ```
   $ echo "Hello World" > hello.txt
   $ hg status
   ? hello.txt
   ```

   This means that "hello.txt" is not being tracked by Mercurial in this repository. We have first to add it to the repository.

   ```
   $ hg add hello.txt
   $ hg st
   A hello.txt
   ```

   Here you see that the commands can be abbreviated as long as they are uniquely identifyable. Try `hg s` and look at the output!

The last line tells you that the file "hello.txt" is now added to the repository. I.e., it is being tracked.

```
$ hg commit -m "Add first version of hello"
$ hg st
```

Ok, now you have commited your very first changeset to the repository. Look, now the status command don’t give no output at all.

Please stick to the following rules:

- ⋆ A commit should be one logical unit! I.e., something that you would use as a patch or to revert it. Anything else that consists of several parts should be commited in several steps!
- ⋆ A commit message should look like in the following:

```
Write clearly for what the commit stands for

Describe the problem this commit solves or the use
case this of this implemented feature. Eplain why
this solution was chosen.
```

- ⋆ Rules
  Be aware of the punctation as well as the spelling and the wording of the phrases.
  - ▷ In active speech
    - ◦ "Add this and that" instead of "This and that was added"
    - ◦ "Fix: #4711 (IE8 not supported)"
  - ▷ First line short (about 50 chars, max. 72 chars); Starts with an uppercase letter and ends without a dot!
  - ▷ Afterwards several paragraphs (max. 72 chars/row)

3. It’s time to look at the history.

```
$ hg log
changeset:   0:ea053dccf796
tag:         tip
user:        Guenter Kolousek <guenter.kolousek@gmail.com>
date:        Mon Jul 01 10:29:21 2013 +0200
summary:     Add first version of hello
```

Fine, now let’s change the file in the working directory and see the difference to the last revision:

```
$ echo "Goodbye" > hello.txt
$ hg diff
diff -r ea053dccf796 hello.txt
--- a/hello.txt Mon Jul 01 10:29:21 2013 +0200
+++ b/hello.txt Mon Jul 01 11:25:00 2013 +0200
@@ -1,1 +1,1 @@
-Hello World
+Goodbye
```

This shows the difference of the working directory to the revision with the revision number ea053dccf796. We see that the first line is changed from "Hello World" to "Goodbye".

The first line starts with `---` and denotes the original file and the second line starts with `+++` and denotes the new file. Afterwards, one or more hunks follow. Each of them starts with a line surrounded by `@@` characters. This range information specifies the range of changes of the original file (denoted by `-`) followed by the range of changes of the new file (denoted by `+`).
Each range is given by the first affected line followed by comma and the count of affected lines. Added lines start with a `+` sign, removed lines start with a `-` sign and unchanged lines start with a blank sign.

4. Damn! We wanted to *append* the word "Goodbye" to the file and not to replace the original content. So, we have to reset the file "hello.txt" of the working directory to the latest revision:

```
$ hg revert hello.txt
$ cat hello.txt
Hello World
$ echo "Goodbye" >> hello.txt
$ hg diff
diff -r ea053dccf796 hello.txt
--- a/hello.txt Mon Jul 01 10:29:21 2013 +0200
+++ b/hello.txt Mon Jul 01 11:36:24 2013 +0200
@@ -1,1 +1,2 @@
 Hello World
+Goodbye
$ hg commit -m "Add 'Goodbye'"
```

Look, there is now a file "hello.txt.orig" which has been created by Mercurial when processing the revert command. Look inside this file, think a bit about, and delete it.

5. Let's check the history again.

```
$ hg log
changeset:   1:b0ac05f6f1ec
tag:         tip
user:        Guenter Kolousek <guenter.kolousek@gmail.com>
date:        Mon Jul 01 12:00:29 2013 +0200
summary:     Add 'Goodbye'

changeset:   0:ea053dccf796
user:        Guenter Kolousek <guenter.kolousek@gmail.com>
date:        Mon Jul 01 10:29:21 2013 +0200
summary:     Add first version of hello
```

Now, let's look at the log information more closely.

⋆ The changeset is specified by a revision number (here 0 or 1) and a changeset ID. The revision number is used to identify a changedset in a single repository whereas a changeset ID is used to identify a changeset **uniquely** (i.e. globally unique)!

  But what are the actual changes?

```
hg annotate hello.txt
0: Hello World
1: Goodbye
```

This means: In revision 0 "Hello World" was modified and in revision 1 the line with "Goodbye" was modified.

Ok, now insert a line with the text "Nice to meet you." between these two lines. Then commit it without the option `-m`, i.e. issue the command `hg commit`. This will open your configured editor. Inside, create a long commit as described previously.

Now, look at the history and specify the option `-v` (for verbose) and the option `-r 2` because we are currently only interested in the commit message of the newly created revision:

```
$ hg log -v -r 2
changeset:   2:c68bc2efee30
tag:         tip
user:        Guenter Kolousek <guenter.kolousek@gmail.com>
date:        Mon Jul 01 16:57:49 2013 +0200
files:       hello.txt
description:
Give a warm welcome!

This improves the feeling of the user...
```

By default, only the summary line is shown. If you want to see the additional description then you have to add the option `-v` as in `hg log -v`.

Nice, but what about a prettier formatted output of the history which comes handy when we will have branches. Try the command `hg glog` yourself. It is available because we activated the "graphlog" extension in ".hgrc".

What are the changes actually? You already know, there is the "annotate" command that shows which line was modified at which revision:

```
$ hg ann hello.txt
0: Hello World
2: Nice to meet you.
1: Goodbye
```

But the most beautiful view of your repository is using TortoiseHg which can be started e.g. `thg log`. Try it and please take your time!

⋆ Let's go back and forth in time! And yes, we will use the command line again:

```
$ hg parents
changeset:   2:c68bc2efee30
tag:         tip
user:        Guenter Kolousek <guenter.kolousek@gmail.com>
date:        Mon Jul 01 16:57:49 2013 +0200
summary:     Give a warm welcome!
```

This will show us the parent revisions of the current working directory. Now let's go back to the roots ;-)

```
$ hg update 0
1 files updated, 0 files merged, 0 files removed, 0
files unresolved
$ cat hello.txt
Hello World
$ hg parents
changeset:   0:ea053dccf796
user:        Guenter Kolousek <guenter.kolousek@gmail.com>
date:        Mon Jul 01 10:29:21 2013 +0200
summary:     Add first version of hello
```

If you just want to see an arbitrary revision this is easy and going forward to the most current revision (the tip) is not a problem either:

```
$ hg cat -r 1 hello.txt
Hello World
Goodbye
$ cat hello.txt
Hello World
$ hg up
1 files updated, 0 files merged, 0 files removed, 0
files unresolved
$ cat hello.txt
Hello World
Nice to meet you.
Goodbye
```

3 Collaborating with Others

1. Making a clone is a common task when using Mercurial as DVCS. It's handy to collobarate with other users (such cloning is also possible over the network) but it is also really useful to implement new features that should be integrated into the main repository later. Let's try it!

```
$ cd ..
$ hg clone repo1 repo2
$ cd repo2
```

Now, you have a clone of the first repository with all managed files of the working directory and the whole history. Verify this statement! It's time to add another file to this repository and also modify the first one:

```
$ echo Günter Kolousek > authors.txt
$ hg add authors.txt
$ hg commit -m "Add authors.txt"
$ echo '!' >> hello.txt
$ hg commit -m 'Add forgotten ! when saying
Goodbye'
```

Ok, take a look at the history.

2. Each clone knows from which repository it is cloned and the changes could be pulled and also pushed (if the access rights allows it). The current changes which can be pulled or pushed can be queried:

```
$ hg incoming
comparing with /home/gntr/repo1
searching for changes
no changes found
$ hg outgoing
comparing with /home/gntr/repo1
searching for changes
changeset:   3:798428a13007
user:        Guenter Kolousek <guenter.kolousek@gmail.com>
date:        Mon Jul 01 21:18:45 2013 +0200
summary:     Add authors.txt

changeset:   4:4554438fe821
tag:         tip
user:        Guenter Kolousek <guenter.kolousek@gmail.com>
date:        Mon Jul 01 21:22:07 2013 +0200
summary:     Add forgotten ! when saying Goodbye
```

3. The changes should be put back to the first repository!

```
$ hg push
pushing to /home/gntr/repo1
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 2 files
$ cd ../repo1
$ ls
hello.txt  hello.txt~
```

Where is the new file `authors.txt` and what about `hello.txt~`?

Let's start with the "missing" file `authors.txt`? It is in the repository but – up to now – not in the working directory! And this is good because it would be possible that we already changed something and that shouldn't get lost. A simple `hg up` will do it!

Let's proceed with `hello.txt~`, it's just a backup file produced by a text editor. Possibly, your editor does not create backup files. Maybe your editor of choice produces backup files but stores it to another location on the file system. However, sometimes such backup files are really handy.

Mercurial produces annoying output from the `hg status` command: Such files will be marked with a question mark. You can really easy suppress this output by creating a ".hgignore" file inside the working directory with the following initial example content (extend this file later if you want)

```
syntax: glob

*~
*.class
*.log
*.out
*.o
```

Now type `hg st` and... Hmm, now the line with the `~` has disappeared but ".hgignore" shows up instead. Surely, you may append another line to ".hgignore" to ignore ".hgignore" but it is much better to add it your repository! Go!!

Let's try something new:

```
$ echo Forever >> hello.txt
$ hg commit -m "Add Forever"
$ cd ../repo2
$ echo Maxi Mustermann >> authors.txt
$ hg commit -m "Add forgotten Maxi"
$ cd ../repo1
$ hg pull ../repo2
pulling from ../repo2
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)
```

Now we use the other way to get the changes back using `hg pull ../repo2`. In this case it is necessary to specify the other repository because repo2 knows that repo1 is the one from which is cloned but the other way around is unknown.

Furthermore, you see the message about one additional head and that you should run the command `hg heads`:

```
$ hg heads
changeset:   6:1d3f10888070
tag:         tip
parent:      4:4554438fe821
user:        Guenter Kolousek <guenter.kolousek@gmail.com>
date:        Mon Jul 01 22:03:49 2013 +0200
summary:     Add forgotten Maxi

changeset:   5:f5ca617b659c
user:        Guenter Kolousek <guenter.kolousek@gmail.com>
date:        Mon Jul 01 22:03:12 2013 +0200
summary:     Add Forever
```

Do you see it, now we have two "heads"! You see it better using the command `hg glog` or even better by typing `thg log`. Try it!

Next, the message after pulling says that we should merge. Here we go!

```
$ hg merge
1 files updated, 0 files merged, 0 files removed, 0
files unresolved
(branch merge, don't forget to commit)
```

Now, we merged the changes back into the working directory. Look at the history using `hg glog` or `thg log`! Then, obey to the order and commit the changes and look at the history again.

4. What if we make changes to both repositories at the same time at the same position?

```
$ cat authors.txt
Günter Kolousek
Maxi Mustermann
$ echo Mini Mustermann >> authors.txt
$ hg commit -m "Added Mini Mustermann"
$ cd ../repo2
$ echo Susi Mustermann >> authors.txt
$ hg commit -m "Added Susi Mustermann"
```

So, we have changes in both repositories at the same time and the same positions. Let's get the changes of repo2 back to repo1:

```
$ hg push
pushing to /home/gntr/repo1
searching for changes
abort: push creates new remote head fa0bd12d4785!
(you should pull and merge or use push -f to force)
$ hg push -f
pushing to /home/gntr/repo1
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files (+1
heads)
$ cd ../repo1
```

Again, look at the history and try to merge the changes back using `hg merge`. Now, the tool meld will pop up. It is a visual diff and merge tool that allows two- and three-way comparision between files and directories.

It shows the original file in the middle and the left and right panes shows the modified versions. You can click the changes so that the changes move from left to right. Holding the `Shift` key allows to delete changes and holding the `Ctrl` key allows for inserting before or after. In addition, you can use it like a normal text editor.

Perform the relevant changes so that the left pane contains all authors in sequence.

This finishes our introduction to the Mercurial DVCS!