Mercurial Introduction

Why Mercurial

Mercurial is a *Distributed version control system*.

A version control is used to

1. Keep track of changes to source code
2. To work cooperatively on the source code
3. Protect against errors

Some vocabulary

repository

This is the place where file current and historical data is stored

commit

This is the action of writing the current changes in the history

changeset

This is a set of changes made in a single commit (often represented by a patch). A changeset has a set of parent, the set of all changesets makes the repository tree

tip

This is the most recent commit in the history

Creating a repository

The first thing to do is to create a new repository

```
$ hg init myrepos
$ ls -a myrepos
.   ..   .hg
```

A special directory `.hg` has been created to store the history of the project. Nothing should ever be changed in this directory without knowing exactly what you are doing.

Status of the repository

Let's say I have some file in *myrepos*

```
$ ls myrepos
A  B  C
```

Files can be *tracked* or *untracked*. The `hg st` command will display the status of all the files in the myrepos directory

```
$ hg st
? A
? B
? C
```

The `?` in front of the file names means that those file are untracked by mercurial.

Adding file to the repository

To track the file a simple *hg add* can be used

```
$ hg add
adding A
adding B
adding C
```

For now the changes are not in the repository history yet. If you want this to happen then you should **commit** your changes.

```
$ hg commit
```

… an editor is spawned so that you can attach a message to the changeset you're about to create.

The repository history

Of course you can see the history of the changes

```
$ hg log
changeset:   1:26071121f84f
tag:         tip
user:        Nicolas Évrard <nicoe@b2ck.com>
date:        Tue Nov 08 13:29:34 2011 +0100
summary:     removing line in C file

changeset:   0:b3759e89231e
user:        Nicolas Évrard <nicoe@b2ck.com>
date:        Tue Nov 08 13:21:37 2011 +0100
summary:     Adding some files
```

This will display the changesets in **this** repository in anti-chronological order.

Making changes to files

Let's say we modified some files. The `hg diff` command will display the changes to the current repository between the file and the **current revision** of the repository.

```
$ hg diff
diff -r 26071121f84f A
--- a/A Tue Nov 08 13:29:34 2011 +0100
+++ b/A Tue Nov 08 13:31:34 2011 +0100
@@ -1,3 +1,4 @@
 B1
-B2
+B3
```

File status and undoing

M	Modified
A	Added
R	Removed
C	Clean
!	Missing
?	Not tracked
I	Ignored

The `hg revert` command will remove all the changes done to a set of files and restore the content of the files to an unmodified state.

Exchanging with others

Now somebody else also want to work with you on your project. This person should use the `hg clone` to fetch you tree of changes

```
$ hg clone path
updating to branch default
3 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Obviously this user should have access to *path*. Please note that *path* can be accessed through different means: **http**, **ssh**, he can also use a **local path on the filesystem**.

This user has received your repository with all the commited changes.

Pulling / Pushing changes

Now we're ready to share our changes with others. There are two ways of doing so:

pushing

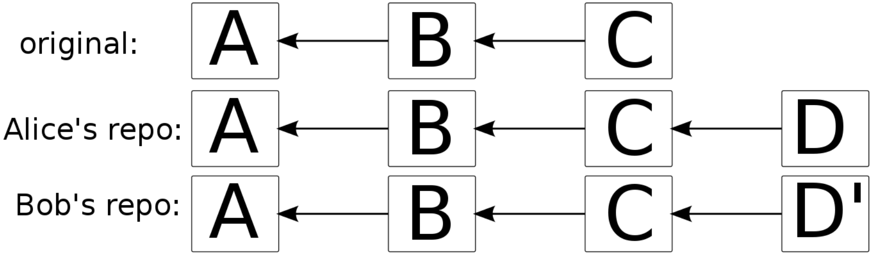
you are sending your changes to another repository.

pulling

you bring other people changes into your repository.

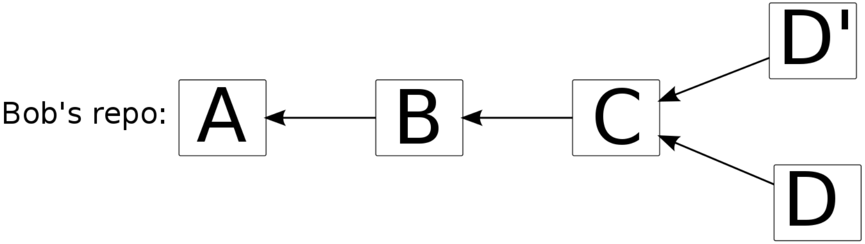
Of course a lot of stuff can (and will) happen when you're sharing your repository.

The starting situation



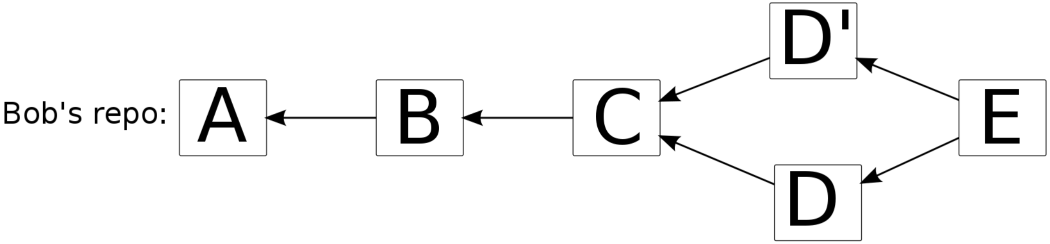
Alice and Bob has *branched* the original repository and commited their own changesets into their respective repositories.

After Bob pulled from Alice's repository



Now Bob must make a merge between both heads with the command `hg merge`. After this command Bob must make sure that everything is well by examining the files (think about `hg st`).

After Bob has commited the merge



Bob can push his tree to the original repository. Another way to synchronise would be for Alice to pull from Bob's repository.

Using mercurial for Tryton development

Organising your code

You should organise your code the way you are used to.

Here is how I do stuff:

1. I have a `projects` directory
2. In the `projects` directory, I have a `tryton` and a `trytond` directory
3. In each of those directory I have another `tryton` (resp. `trytond` directory) which is the clone of the repository on `hg.tryton.org`
4. Both those repositories should be updated daily

5. When I work on a feature I make a branch:

```
$ hg clone tryton tryton-issueXXX
```

6. When the patch is ready for inclusion:

```
$ hg pull -u
… fix eventually broken stuffs …
$ hg commit
$ hg push
```

7. `hg push` from the trunk clone. Beware that your trunk should be up to date otherwise you might have a merge to do!
8. Keeping your trunk repositories up-to-date is a good idea. A tool like `mr` (http://kitenet.net/~joey/code/mr/) can help with this task.

(Almost) required extension

`hgnested` is a mercurial extension that will allow you to fetch a tree of mercurial repositories. For this to work, the source repository must also use the `hgnested` extension.

Install hgnested with pip:

```
$ pip install --user -M hgnested
```

`hgnested` provides a bunch of nested commands nclone, ndiff, npull, npush, nstatus, …

Making your life easier: hgreview

rietveld websites are used in the Tryton development to share patches about proposed features or bugs.

Those websites allows people to upload patches, comment on them, published revised version of the patches, comment on them again until the patch is ready for inclusion.

`hgreview` is an extension that will make your use of rietveld websites easier.

```
$ pip install --user -M hgreview
```

Using hgreview

With hgreview you can submit your patch for review:

```
$ hg review
Server used http://localhost:8080
New issue subject: Removing a line
Issue created. URL: http://localhost:8080/1
Uploading base file for C
Uploading current file for C
```

You can also review patch made by others:

```
$ hg review --fetch -i 1
Looking after issue http://localhost:8080/1 patch
applying http://localhost:8080/download/issue1_2.diff
```

Mercurial Setup for Tryton

You should edit your `$HOME/.hgrc` in order to contribute effectively to Tryton. Those settings will be the default for all your mercurial repositories. You can always have repository-wise setting by editing the `.hg/hgrc` file.

First define your username:

```
[ui]
username = Nicolas Évrard <nicoe@b2ck.com>
```

Then activate the `hgreview` and `hgnested` extensions:

```
[extensions]
hgnested =
hgreview =
```

The `hgreview` extension use by default the appspot server. To use the tryton server just add those line in the mercurial configuration:

```
[review]
username = nicolas.evrard@b2ck.com
server = http://codereview.tryton.org
```

Submit your patch

You first need to write a good commit message:

```
Fixed computation of gantt chart width

The computation was off by one in some very specific case
for example when a task was in low priority and the assignee
is selling some Dunder Mifflin paper

issue42
review28
```

Then you should export your changeset using this simple mercurial command:

```
$ hg export tip > gantt_fix.patch
```

And attach this file to the bug you're fixing. Do not forget to reassign the issue to nobody.