S-PM 140524

Mercurial Version Control

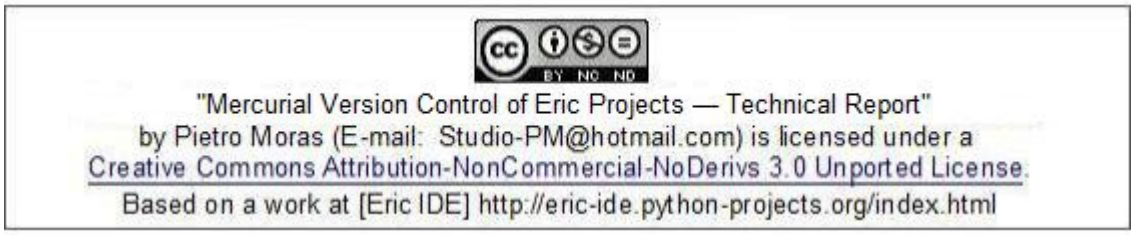
of

Eric Projects

Technical Report

Copyright Page

S-PM 140400



Current edition PDF file, `Eric_Deliver` site URL: http://www.box.net/shared/k64yenrpey under the "Creative Commons License"

Disclaimer The information in this document is subject to change without notice. The author and publisher have made their best efforts about the contents of this book, nevertheless the author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any loss or damages of any kind caused or alleged to be caused directly or indirectly from this book.

Brand Names Brand names such as Linux, Windows, Python are assumed to be universally known, and are here used in full respect of their respective owners.

Planned edition On paper, under traditional copyright

Published by [*not yet—just appointed*] Town & Country Reprographics, Inc.
230 N Main St Concord, NH 03301 (U.S.A.)

- = -

Foreword

S-PM 140000

Dear Reader,

Here is this other Report *to enrich the collection of publications centered upon the Eric developing environment [see: Eric Web Site], produced under the same basic principles characterizing each and all of them.*
That is, in summary:

- *All what you'll read here is the result of a test & documentation project carried on in fair collaboration with the Eric producer but, nevertheless, in total independence from him.*
- *Besides positive technical documentation and instructions, also our possible doubts, perplexities and difficulties have been here fairly reported.*
- *Technical dialogue with you is strongly encouraged, to the obvious benefit of all involved parties. To some extent, we'll measure the effectiveness of this* Report *by the amount of feed-back originated, as an evidence of the active role that the user community of such advanced s/w product, as Eric is, should be entitled to play. Active role that we strongly advocate and welcome.*

See you.
The Author

- -

What & Where in the Internet

List of Eric "Technical Report" so far published:

Eric Python 3 (& 2) Integrated Development Environment	[*S-PM 130900*]
Eric Project Packager	[*S-PM 130500*]
Eric 4 as Python 2.x Integrated Development Environment Version Control	[*S-PM 120800*]
Eric 4 as Python 2.x Integrated Development Environment Web Browser	[*S-PM 120200*]
Eric 4 as Python 2.x Integrated Development Environment	[*S-PM 110800*]

Web references:

Tech. Reports	Deliver:	http://www.box.net/shared/k64yenrpey
	Specimen ("*AsIs*"):	https://www.box.com/s/52xae8aac52badewvdn0
	Author:	Studio-PM@hotmail.com
Eric Web Site	Docs:	http://eric-ide.python-projects.org/eric-documentation.html
Eric Discussion	Mailing List:	eric@riverbankcomputing.com

-<>-

Two Phases

S-PM 140500

It is worth knowing that the test activity this Report is based upon underwent two distinct phases.

i. An initial phase, during which we endeavored discussing the results of our tests both with the Eric and Mercurial people, so to clarify doubts, shed light over relevant aspects, smooth asperities and intercept possible bugs. A process that we assumes as equally useful for product's designers, users and future readers and that, by experience, we know may be particularly effective when you first begin using a new product, as a relatively inexperienced and unbiased user. All that, obviously, done before the resulting Report went published.

With Eric people that was the usual way, whereas with Mercurial people such dialog revealed more and more difficult, as virtually any expressed perplexity, or question, or doubt, not to mention possible bug accounts, were considered little less than an affront, a crime of lese-majesty perpetrated by an impudent user.

ii. As a consequence we decided to complete the test process and then write down consequent results as they appeared to us, without any further preliminary discussion. Alas.

Alas, because working this way is against our deeply rooted conviction and feelings, as just a way for regaining the serenity needed so to carry on and complete this work.

– –

Nevertheless happy then to resume such discussion after the Report's publication, provided serenely and correctly intended, in fair pursuit of a better understanding and evaluation of the product.

– = –

Essentials

Essential reference data, so to precisely identify the subject and product we are talking about, along with its prerequisites and hosting environment.

S-PM 140500

Subject "Mercurial Version Control of Eric Projects", short form: "E-Hg"

Product Eric IDE and related prerequisites [*see: "*Eric Technical Report*"*[1]], plus the optional Mercurial package and the interpreter Python ver. 2, as required by this tool.

Version Reference – Eric IDE

No	*Item*	*Version*	*Note*

1	Windows XP Pro
	Windows Vista Ultimate
	Windows 7
		`5.1 (SP 3)`
		`6.0 (6002:SP 2)`
		`6.1 (7601:SP 1)`
			Host Operating System (x86, 32 bit platform)

2	Python 3	`03.03.02`	Ver. 3, as required by Eric
3	PyQt4	`04.10.02`	GUI library
4	Eric	`5.4.4` `(rev 13324c5b2353)`	IDE Application

Version Reference – Mercurial SCM[2]

No	*Item*	*Version*	*Note*
1	Python 2	`02.07.01`	Ver. 2, as required by Mercurial
2	Mercurial	`02.08.00`	SCM (Source Code Management) system

– –

1 As available at URL: http://eric-ide.python-projects.org/eric-documentation.html, or: http://www.box.net/shared/k64yenrpey

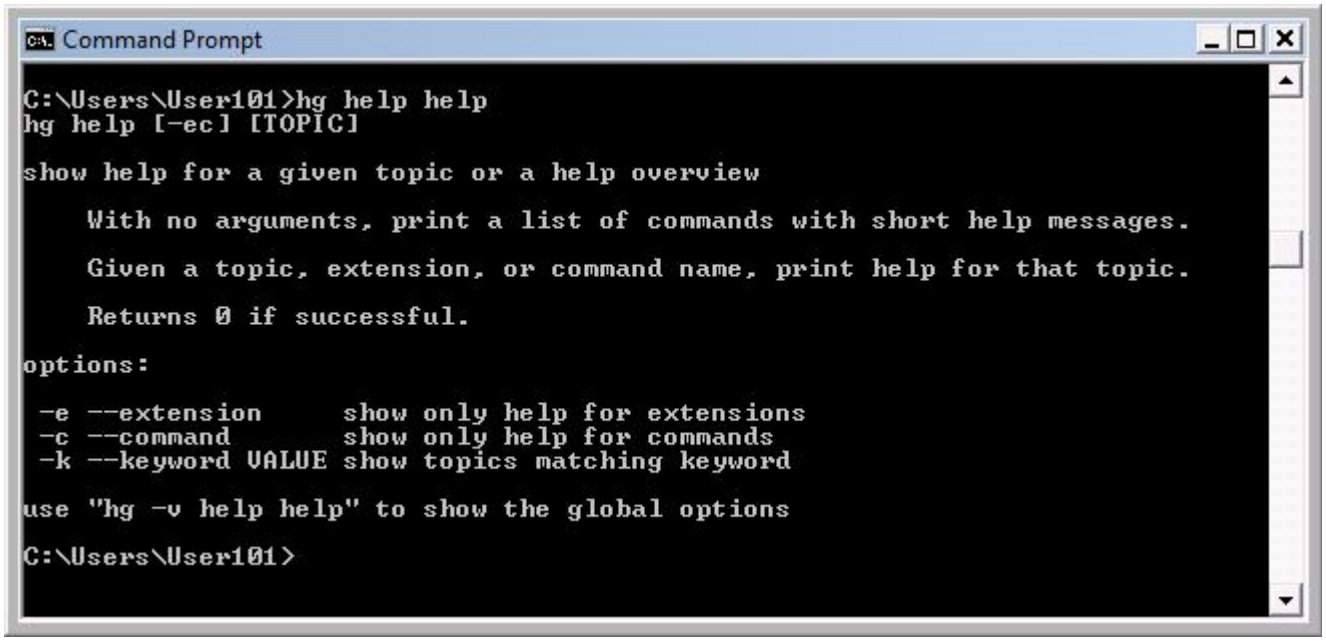
2 <~> SCM "Source Code Management" system is precisely how Mercurial defines itself [*cf.:* Mercurial Command Reference*, at* Mercurial Package*, in* Appendix], with explicit and unambiguous reference to the *Source* material as the object of its action. Whereas Eric—and, therefore, this Report too—uses the more usual and by far more generic diction VCS "Version Control System".

Scope of this *Report*

Scope and purpose of this Report is exclusively the usage of Mercurial considered as a tool integrated into the Eric developing environment.

That is to say that this is not a paper about Mercurial considered in itself, and in general. To that purpose you may refer to:

- Standard Mercurial documentation [*ref.: at* Mercurial Package, *in* Appendix]
- Mercurial command: `hg help`



- Reference book: "Mercurial: The Definitive Guide", by Bryan O'Sullivan

– = –

Terms and Concepts

An explicit list of terms and concepts as here intended is a necessary prerequisite for the correct and unambiguous understanding of this Report.

You may well know or find elsewhere other equally valid or better definitions, such as those at the "Glossary" section of the official "Mercurial Command Reference"[3] [*ref.: at* Mercurial Package*, in* Appendix]. The point here is purely that of granting coherence between terms and intended meanings throughout this very Report.

Viewpoint

Were you going to a desert island, forced to observe tremendously strict luggage limitations, from this entire Report we'd suggest you to strip off and bring with you just this page, as all the rest is little more than a simple and fairly obvious consequence.

– –

S-PM 140000

Eric Project — It is what you create with the command `Project > New...` [*see*] and then develop using this IDE. "Project" here conventionally written with initial capital letter when intended in this specific, i.e. not generic, sense.
<~><~> Note that "Project" is a specific Eric concept, with no precise correspondence in Python terms.

Version Controlled — Or, simply: *versioned*, is the term here adopted to briefly qualify a Project for which a Version Control System has been activated. That is, here, Mercurial, if not otherwise specified.

Revision — Each recorded and identifiable stage of a versioned Project. The act of recording a Revision is called Commit. The set of recorded Revisions constitutes a topological structure, a graph ordered and connected. Each Revision of a Project is identifiable and equally available to be handled individually, one at a time.

Remark

<~> This is how *Revision* is intended in Mercurial terms, not in general. In general, to specify different editions of a s/w product, it is normal to use such a numeric code: ***m.n***, where ***m*** is usually called *version*, and ***n*** a *revision* (of that main version).

Of course also Mercurial versioned products can be freely specified in such a customary way, for external purposes; that is, independently from the specific Mercurial Revision identifier. With such

3 <~> Anyhow a Glossary on which we do not rely that much, as too often obscure, misleading, and with some rather peculiar choice of terms that we are here happy to ignore.

"*m.n*" custom identifier that can be associated also to any Mercurial Revision in form of a so called Mercurial "*Tag*" [*see:* `>> Tag in Repository...`].

– –

Changeset A term referring to the identifiable set of metadata defining a Revision, and stored into a Mercurial Repository as the consequence of a Commit action.

Remark

<!> *Changeset* and *Revision* are substantially equivalent concepts, just a conveniently different way of looking at the same thing: Changeset as an aspect internal to Mercurial, Revision as its external expression and usable result.

Changesets are intrinsically incremental, Revisions integral. Both items are identified with the same code [*see next point:* Revision Number, Revision Identifier].

– –

Revision Number, *Revision Identifier*

Each Revision is unique and unalterable, and can be referred to by means one of these two indexes:

Revision Number (in short: Rev. No, or `rev(number)`)

A whole number (`0`, `1`, ..., *n*), automatically assigned to a Revision when Committed; consecutive and unique within the same Working Directory, that is with a *local* scope. The same Revision in different Kin Working Directories can assume different Development positions and, therefore, a different Revision Number.

Revision Identifier (in short: Rev. Id, or `id(string)`)

A hexadecimal code derived from the very Revision contents by means of a Secure Hash Algorithm[4], computed and assigned to a Revision when Committed; unique within all related Kin Working Directories, that is with a *global* scope. The same Revisions in different Working Directories can have different Development positions, but are anyhow associated to the same hexadecimal Identifier.

Preference for one of the two identifying indexes is, obviously, just matter of scope and convenience. Within the same Working Directory Rev. No is more handy, within different Working Directory Rev. Id is a must.

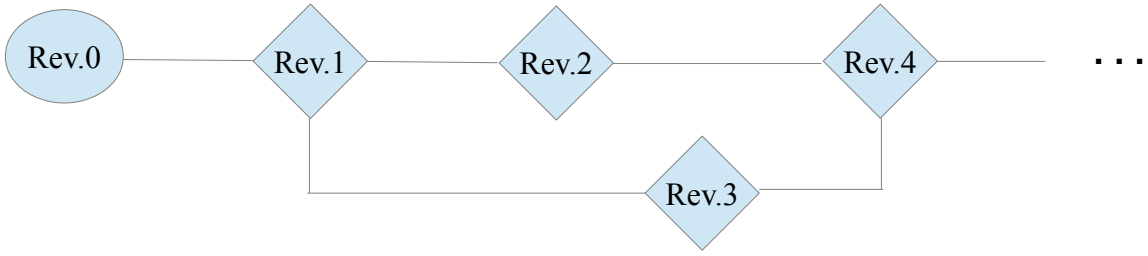
– –

Development Ordered set of Changesets leading to a given Revision. In other words, each given Revision can be considered as the ordered composition of all Committed Changesets,

4 More precisely: a "Secure Hash Algorithm" *SHA-1*, 40 hex digits, all used internally, first 12 in display. Odds of collision for the 12 digit are one in 16 million, and relatively harmless (i.e.: easy to recover).

from the Origin to the desired Revision Id.

More precisely the Changesets of a Development constitute such a Graph:



As you see, the Graph of the Revisions representing a Development can easily become fairly complex. In such a Graph, these are the most notable elements:

- An Origin Revision, followed by other Revisions. Each Revision has associated a unique identifier [*see:* Revision Number, Revision Identifier*, above*].
- Named Branches. All Branches in this Graph have got a name, either the initial "`default`", or a custom assigned one.
- Topological Branches, that is Branches forked out of the same Named Branch.
- Directed (i.e.: oriented) segments, orderly connecting all Revisions.
- Parent Revision, followed by a Child or Descendant; with one Revision that can be the Descendant of two Parents, as the result of a "`merge`".
- Ancestor is called any Revision preceding a given one; from its Parent down, backward, to the Origin.
- Head Revision, with no Descendant. In a given stage of a Development there may be more than one Head; that is one per Branch, either Topological or Named.
- Most recent Revision is conventionally called "*Tip*". Necessarily only one in a Repository, and not necessarily belonging to the currently active Branch.

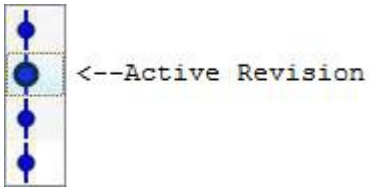
Viewpoint

<!> As here stated, the definition of "Heads" and "Tip" are equally neat and unambiguous; whereas, throughout the Mercurial product, they tend to assume different and contradictory meanings and roles with different commands [*see:* `>> Switch...`, `TIP` *selector*]. In this Report we'll firmly stick to the concepts as here stated [*cf. next section:* Dealing with Mercurial Tip, Heads and Active Revision].

– –

Active Revision "Active" is the term here adopted for the current pending Revision, that one you are currently working on. Typically, but not necessarily, the Tip, as any other Committed Revision can be "Activated" at any given time [*see:* `>> Switch...`].

Schematically:



Active Revision is represented on the "`Log Browser`" window with a larger dot [*see:* `>> Show Log Browser`]. The Current Active Revision is necessarily also the Parent of the current Working Revision [*see next point*].

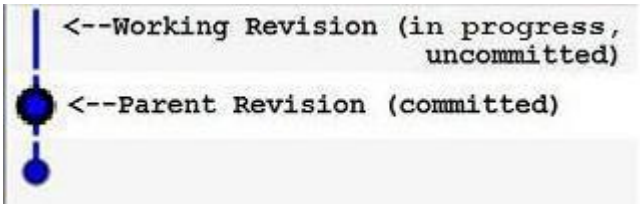
– –

Working Revision

This is the term here adopted to qualify the condition of a Revision under development, i.e. *in progress*, to be then possibly Committed[5]. A Revision in progress is necessarily the Child, or Descendant, of a Committed[6] Revision, called Parent.

One and only one Revision can be "in progress" in a Working Directory at any given time.

Schematically:



Intuitively: a Working Revision can be seen as what you have between consecutive Committed Revisions.

Being customary to identify a Working Revision with the same identifier of its Committed Parent, in this Report we'll make systematic and explicit use of this "*Working*" specification, so to avoid any nasty ambiguity and confusion between the

5 As such sometimes referred to also as a "un-Committed" or even "dirty" Revision [*where "*dirty*" stands for "*molding*", "*undergoing a creativity process*"*].

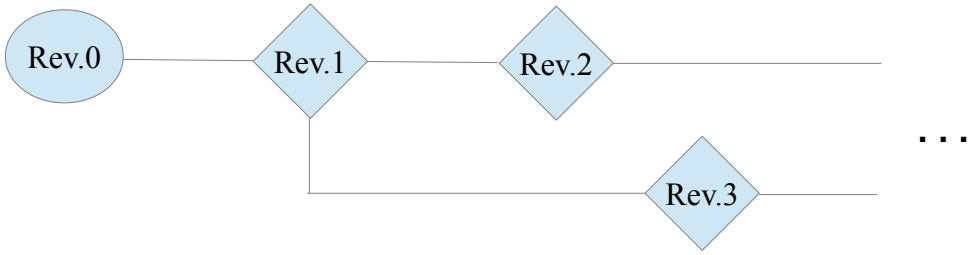
6 <~> In this Report normally we do not specify "Committed Revision", as opposed to Working Revision, if not for extreme clarity's sake.

Committed and not yet Committed Revision you're currently still working on[7].

Once Committed, a Revision will take a new identifying code of its own.

Remark

Working Revision need not necessarily be derived from the most recent one, the Tip. That's usual, but not mandatory, as you can well switch "back", and work on any former Revision. What will then happen, when you possibly Commit the introduced changes, is the creation of a so called "*Topological Fork*".



That is a new Tip Revision, but on a new Topological Branch, which is a forked Branch with the same Branch Name, and with two Heads. Distinct Heads can then well be merged into one successive unique Tip Revision.

– –

Commit Act of permanent recording of a Revision in progress into an identifiable new Revision[8]. It is an *incremental recording process*, as only changes are recorded in the so called Changeset [*see above*].

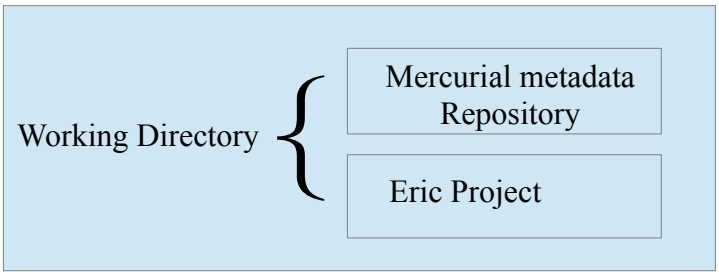
– –

7 <~> Possible cause of uncertainty and errors for many users, growing unsure and distrustful; as we've so often realized.

8 By the way, note that, of course, not all possible intermediate stages of development of a Project can be identified, and then possibly recalled; but only those that have been explicitly "Commited".

Working Directory

Usual Mercurial term for the container of the versioned project you are currently working on, corresponding here to the Project Directory of a versioned Eric Project. "*WDir*" hereafter for short.
<!> Note that it is ***inside*** this directory that, in case of a Mercurial Version Controlled Project, you'll find the so called "Repository" of Mercurial metadata [*see hereafter*].



Project Directory

Root directory where an Eric Project is stored.
<!> For Eric versioned Projects the Project Directory assumes also the role of a Working Directory, in Mercurial terms.

Remark

In this Report, as a rule, with "Working Directory" we'll refer exclusively to Mercurial versioned Eric Projects; whereas, with "Project Directory", indifferently to versioned or not versioned Eric Projects.

– –

Repository Sub-directory within a Working Directory aimed at containing the Mercurial specific metadata, as a rule automatically named: "`.hg`".

<!> Note that a Repository is NOT the container of any Eric Project data. In fact you could erase a Repository without losing a bit of your Eric Project: you'd just lose "only" its Version Control metadata. And it is precisely to make this concept evident that, in this Report, we'll often use the expression "metadata Repository", instead of simply "Repository".

Remark

Clear distinction between the whole "Working Directory" and the inner "Repository" is necessary for the understanding of specific Mercurial actions, possibly aimed at the metadata Repository only, or the Project data only, or both.

This the clearest statement found in the original Mercurial Glossary about such concepts:

"*The repository metadata exists in the .hg directory inside the working directory.*"

and this other, somehow more reticent:

"*A repository is usually (but not always) found in the* `.hg` *subdirectory of a working directory*".

In particular note that Repository in NOT a synonym for Project storage, rather a common misunderstanding, easily leading to nasty confusion[9].

– –

Viewpoint

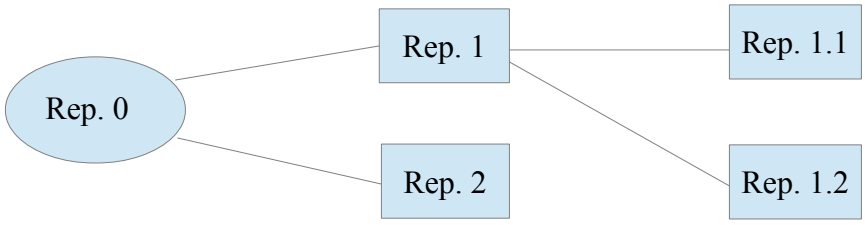
<!> In this Report we'll make clear and unambiguous distinction between Eric Project, Working Directory and metadata Repository as here above defined. And clearly and unambiguously consider "`.hg`" nothing else than the conventional name assigned to the Mercurial metadata storage of a Version Controlled Project, located within its very Working Directory. For clearness' sake.

– –

Network of Working Directories

Or, equivalently, *Repository Network*. Because, as already noted, a single Mercurial operative unit can be equivalently referred to as a Repository or as a Working Directory; which is perfectly acceptable, provided these distinct terms are not confused.

Mercurial is defined as a "distributed" Version Control System because each Working Directory is autonomously operative, independently from any other one. Which is true, but it is equally true that if a set of Working Directories are concurrently aimed at the development of the same compound project, there are crucial moments when they must operate connected in network, as hereafter represented[10].



– –

9 Actually it could be defined as a synecdoche, that is: "a figure of speech in which a part is used for the whole, as in *sail* for *ship*". Nice in rhetoric, may be misleading in technics.

10 Note that the No.s in this figure are mere labels, as no standard identifying code is associated to Mercurial Repositories.

Notable aspects of such a Repository Network are:

- A "*Main*" Working Directory, created first, possibly the final container of a compound target project.
- *Child* or Descendant WDir, "cloned" from another *Parent* WDir, typically but not necessarily the original "Main".
- Changes independently generated in each WDir that can be transferred and interchanged with any other WDir connected in network; that is, in Mercurial terms: *Pushed* or *Pulled*. That said, provided respective WDir access permissions would permit it[11].
- Changesets is what is actually interchanged. This implies that identical Changesets can be orderly part of different Working Directories, each one singularly identified with unique identifying codes. As a rule, such interchange of Changesets, when referring to the same source files, can generate contents "conflicts", conveniently intercepted and managed as a normal aspect concerning the development of a versioned Project.
- Each Repository can be connected to another Repository through a single URL address as stored in each Repository. It is in this sense that Repositories can form a Network. By default a cloned repository results connected with its Parent, in a sort of informatics umbilical chord. But that's simply a default, well changeable at will.

Network of Kin Projects

All Working Directories, Projects and Repositories stemmed out of a common Origin, or out of subsequent Revisions, within the same *Network* of Working Directories, Projects and Repositories will be hereafter generically referred to as "*Kin*".

– –

"*Version Control System*"

In short: VCS, is usually how Eric refers to such tools as Mercurial in all its menu, captions, labels and messages. And, therefore, how we'll also do throughout this Report. Sometimes also as: VC System.

A "*Source Code Management*" (SCM) tool

Is how Mercurial defines itself. A definition here considered equivalent to "*Revision Control System*" (RCS), or the above cited "*Version Control System*" (VCS), as other similar tools call themselves[12].

11 <~> Read/write access permission is a subject well off the scope of this Report.

12 Notably Subversion, as treated in the similar "Eric 4 – Version Control" Tech. Report, already published [*see Eric web site*].

Viewpoint

But here, saying "*Source Code Management*", there is something that goes beyond names and conventions, reaching semantics.

<!> As a matter of fact "*Source Code*" is not a generic expression, applicable to a vast class of things, it means a precise and well defined type of file, nothing else.

<?> Nothing else? Does it mean that Mercurial is intended, by design, as a versioning system to be used uniquely for Source Code files? Even considering "Source Code" in the broadest possible sense, that is as plain not-formatted text, this would certainly imply a rather drastic scope delimitation.

<!> But how relevant, in principle, such a question may be, for this very Report it doesn't make any significant difference, as E-Hg is anyhow intended for Eric Projects only, that is to say typically for Python source files only. That is, of course, not considering the versioning of any other possible resource file an Eric Project may comprise.

Naming Convention

Note that in this Report we generally refer to Mercurial, in accordance with Eric, as a "Version Control System" (VCS), instead of as a "Source Code Management" (SCM) system, as the very Mercurial does [*cf.:* Mercurial Command Reference, *at* Mercurial Package, *in* Appendix].

<~> And this even though *we have strong practical evidence* that the original Mercurial denomination is more accurate and prudent, as it delimits more precisely its scope to the class of *Source Code* files only. That is, fairly avoiding to nourish any illusion that other kind of file items could be equally managed with this tool. Just imagine, for instance, what would be the managing of conflicts when merging encoded or binary files.

– –

Mercurial vs. *E-Hg*

<!> Of course the Mercurial tool considered in itself, or considered as integrated into Eric, are ***not*** the same thing. In this Report, when required to make the distinction, we'll refer to the former as "Mercurial", to the latter as "E-Hg".

–<>–

Dealing with Mercurial Tip, Heads and Active Revision

Looking at this matter from Mr. End User's point of view you may experience disconcerting experiences. You'll have that, in some cases, as with command `>> Show Log Browser`, the Tip is actually declared where expected, but not so in some others.

<~> Just for instance, consider entering the command `>> Switch...`, with the option "`TIP`" enabled. As a result you'll be switched to the Head of the currently active Branch, which is not necessarily the same thing as the Tip.

The point is that in a plain Mercurial Repository you may well be having, as the "Active" Revision you're currently working on, the most recent one, that is the so called Tip; which may be also the "Head" of the only existing Branch. Rather a common condition where Active, Head and Tip Revision mean the same thing.

<!> But in case you have got several Branches, each one with its own Head, and where you've possibly activated a Revision older than the Head, you have that Tip (N.B.: necessarily one and only one), Heads (N.B.: possibly more than one) and Active (N.B.: one, at most) are different things altogether, amongst which Mercurial commands—and E-Hg's as a consequence—sometimes make a mess of confusion. Many examples have we stumbled on, some of which annotated throughout this Report, as in the above <~> point.

<~> As another confounding example, try a Bookmark definition, with the "`TIP`" option enabled, when being currently positioned at a Changeset preceding a Branch Head. Just try, and you'll see the action remaining on the current Changeset, not even going to the current Head.

<~> Or, in case you like consulting the related original Mercurial documentation, consider this definition of "Branch Tip" (N.B.: *not* "Tip", but "Branch Tip"): "*The head of a given branch with the highest revision number*". So confusing that the only serious solution is to accurately avoid using it, preferring the equivalent and unambiguous term "Head"; as we've constantly done throughout this Report.

– –

We've no precise idea whether all this lexical and conceptual confusion amongst "Tip, Heads and Active Revision" is a Mercurial issue, an E-Hg issue or a bled of both. As for any Mr. End User all what we can do is to bear witness, as we are doing with this Report, and hope for the best.

-<>-

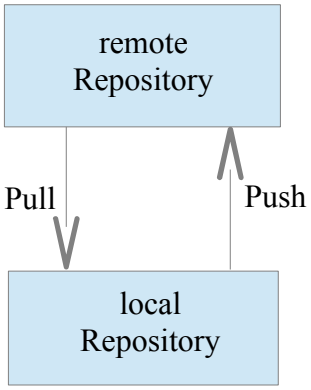
Terms and Data Flow

A schematic representation of such specific Mercurial terms as: Commit, Update, Push, Pull, referred to the interchange of version control data, can be of help for the correct and immediate interpretation of the E-Hg commands.

– –

Project <—> Repository Data Interchange

Local <—> Remote Repositories Data Interchange



As a typical data-flow example, to transfer the result of a source editing action from your local Project to a central master Project, you have first to *Commit* it to you local Repository, and then *Push* from your local Repository to the Remote one. Or, inversely, the owner of the remote Repository could *Pull* the new metadata from your Repository into his one.

Then, there, to complete the transfer, the owner of the remote Repository will operate locally so to *Update* his Project from that metadata Repository. All that, of course, leaving out of consideration the read/write permission on each involved data storage, which is another story altogether.

– = –

E-Hg Primer – Versioning in Practice

Purpose of this section is that of illustrating what is the essence, the spirit and use of this E-Hg sub-system through the description of typical operative actions, easy to grasp and repeat. Nothing of exhaustive, of course, it's not that the purpose, but fairly illustrative, yes. A tutorial hands-on primer complementary to the general and detailed treatment as can be found on the other sections of this Report.

– –

Putting an Eric Project under Version Control

Eric Projects can be "version controlled".

Why?[13]

- So to keep ordered and verifiable track of its Development history, that is all of its Committed Revisions. This way, each recorded stage can be revisited so to check what has been done, when, by whom, and how it worked, bugs included.
- So to orderly collect distinct contributions coming from distinct designers, concurrently working on the same Project.
- So to keep and manage ordered set of different flavors of the same Project, variously tailored and customized. Up to the Forking of a Project into distinct lines of Development.
- So to enhance the possibility of collaboration and data interchange within the s/w design community, where the adoption of Version Control System is becoming common practice.
- For these and many other reasons, as described in dedicated technical papers and books, such as "Mercurial: The Definitive Guide", by Bryan O'Sullivan, specifically focused on the Mercurial system here considered.

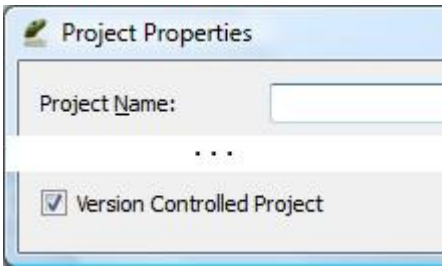
– –

How?

- Ticking on the "`Version Controlled Project`" check-box[14], at the very inception of a new Eric Project.

13 <~> Then see the extra "***Off Report***" section, if interested in some "*Why Not*" reasons...

14 Note that this check-box is consequent the actual installation of a Version Control System [*see:* Setup and General Management, *in* Appendix].



- Turning versioned an existing unversioned Project, by means of the command:
  `Project > Version Control > Add to Repository...`

Then, if you change you mind, you can always turn a versioned Project back to "normal", with command: `Project > Version Control > Export from Repository...`

– –

Resetting the Development History of a Versioned Project

A fresh restart for a versioned Project can be obtained combining the two aforementioned actions, this way.

`Project > Version Control > Export from Repository...`
so to turn a versioned Project unversioned, and then:

`Project > Version Control > Add to Repository...`
so to turn it versioned again, with a Development history restarted from scratch.

– –

"Cloning" of a Versioned Project

As a typical initial action when joining a team of developers working on a versioned Project, you make a local copy of that Project, at its current stage of Development. In Mercurial terms this copy is called a "clone", and is to be done with the E-Hg command:
`Project > Version Control > New from Repository...`

– –

Addition of New Files to a Versioned Project

Even at this "primer" level the addition of a new file to a versioned Project requires a special mention. Point is that current Version Control Systems, while capable of intercepting automatically changes performed on existing files, are incapable of realizing automatically the very presence or absence of a file inside the Working Directory of a versioned Project.

Therefore it's up to the designer, after having possibly added a new file to a versioned Project, to inform the Version Control System of such an event, trough a dedicated context command:
`Project-Viewer(^)> Add to Repository / Remove from Repository`

And analogously, in case of deletion: `Project-Viewer(^)> Remove from Repository`

Viewpoint

<~> Of course, whatever sound justification may be brought, this is a manifest sign of immature technology.

– –

"Committing" of a Versioned Project

Each new significant stage of a versioned Project can be permanently saved with a so called "Commit" action, to be performed with the E-Hg command:
`Project > Version Control > Commit Changes to Repository...`

What "significant stage" means is left to the free interpretation of each designer[15].

– –

Switching Among Revisions

Central feature of a Version Control System is that of navigating back and forth on the Development history of a Project, among different Committed Revisions; here with command:
`Project > Version Control > Switch...`

Then, at each different Revision, you can test your Project, execute changes, generate and store new version of the same Project, and display overall versioning info with several E-Hg commands, such as:

`Project > Version Control > Mercurial`

`Project > Version Control > Show Log / Show Status... / Show Difference`

– –

Working in Team

You can team up with another versioned Kin Project by means of the "`default` `[paths]` parameter of the "`.hg\hgrc`" configuration file, located into each Project's Repository. For cloned Projects this parameter is defaulted to the origin's; anyhow it can be also freely set at will [*see:* Configuration Files*, in* Appendix].

15 <~> Our attitude is: better not too often...

Once connected through `[paths]` parameters, new Revisions independently Committed in distinct Kin Projects can be explored with commands:

`Project > Version Control > Show Incoming / Outgoing Log`

and possibly interchanged with commands: `Project > Version Control > Pull / Push Changes` That is, respective read / write permissions permitting.

Once a new Revision received, a: `Project > Version Control > Update from Repository` (or equivalent `Project > Version Control > Switch...`), so to complete the process from the metadata Repository upto the actual Project level.

– –

Remark

<!> Besides the usual `Pull` / `Push` commands, E-Hg offers to the Project designers operating in team other ways for sharing their contributions, suited to be used in different circumstances. Such as:

`Changegroup Management`
: With no direct connection required, sort of an "off-line" Pull / Push equivalent system (a specific E-Hg tool based upon the original Mercurial "`bundle`" command).

`Import / Export Patches...`
: For source fragments, distinct from standard Revisions. Can be used independently from Mercurial too.

`New / Export from Repository...`
: For entire Projects.

– –

<~> That is, without forgetting the neatest of all ways for distributing s/w contributions and updates:

`Plain source file interchange`[16]

– = –

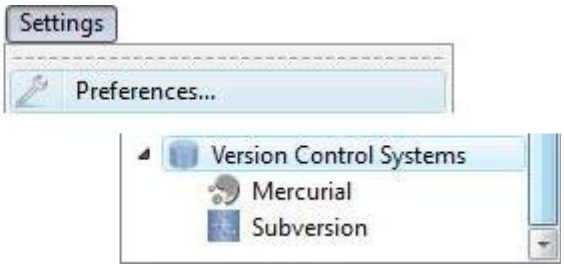
16 [<~> *Frankly our preferred …*]

E-Hg Command Reference

Once installed as described with section "Setup and General Management" [*see: in* Appendix], the Mercurial tool results smoothly integrated into Eric, usable and manageable through the commands found available in the distinct E-Hg main menu [*MM*] and context menu [*CM*] sections, as hereafter listed.

– –

E-Hg Main Menus



Main Menu: Project

`Project > New...`

Where a new Project can be created, version controlled since the beginning;

`Project > Version Control`

Where to perform version control actions on a Project.

Main Menu: Settings

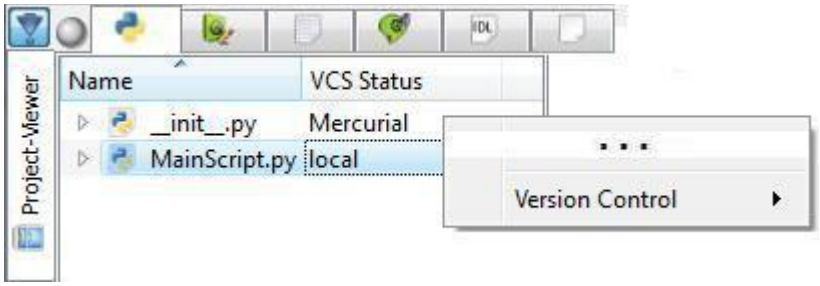
`Settings > Preferences... - Version Control Systems > Mercurial, ...`

Where to manage version control configuration and parameters.

– –

E-Hg Context Menus

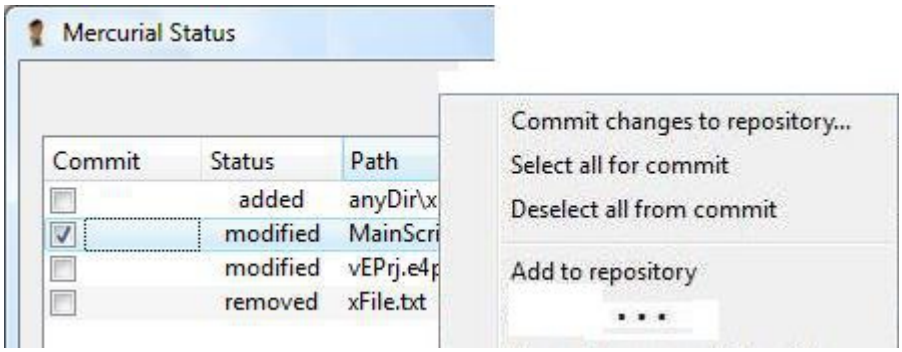
Project-Viewer(^) Version Control



On the Left Pane, Tabs: `Sources`, …, `Others`

– –

Mercurial Status(^)



On the form of the *MM* command `>> Show Status...`

– –

VCS Status Monitor "LED"(^)

At right on the bottom Information Bar.

– –

<~> For the general description of these same command menus, considered independently from the presence of a Version Control System, please refer to the dedicated "Eric Technical Report" [*ref.: Eric Web Site, in* Foreword], on account that the following sections here are mainly focused on the E-Hg functions only.

– = –

[*MM*] E-Hg Main Menus

[*MM*] Main Menus vs. [*CM*] Context Menus

Scope, operative environment, of the E-Hg Main Menus [*MM*] is the entire Eric Project, considered as a whole; with no distinction between single items—that is: files and directories—belonging to the same Project. Whereas scope, operative environment, of the E-Hg Context Menus [*CM*] is the single file or single directory of an Eric Project.

Eric Users must be well aware of such distinction, and primary purpose of this Report is to offer valid technical reference for both classes of commands, in dedicated sections so flagged: *MM* and *CM*.

– –

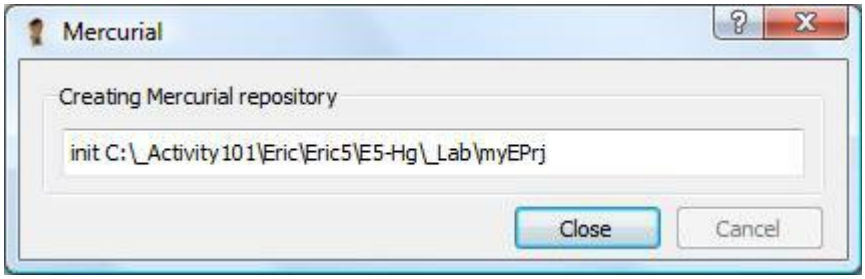
Common Action List

Some E-Hg commands do perform similar "atomic" *Actions*, as hereafter listed [*A01*, …], which are executed making use of similar dialog boxes and which it is convenient to consider preliminarily, in dedicated sections.

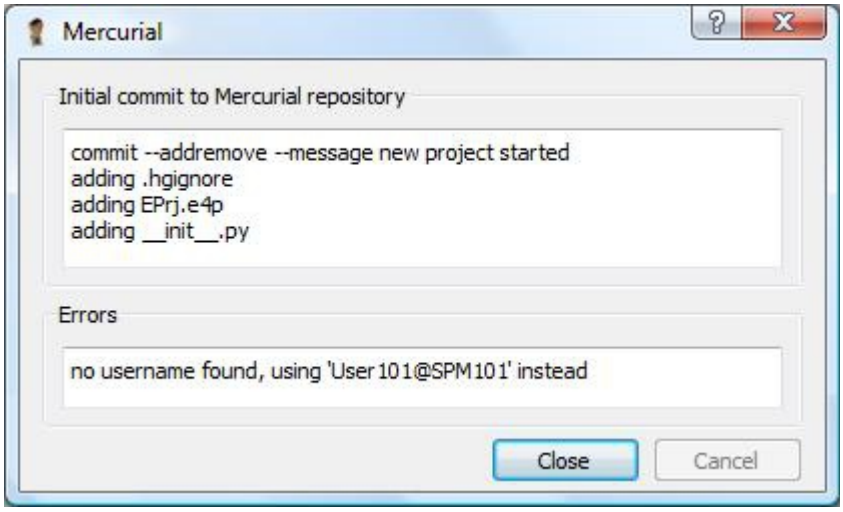
– –

➢ [*A01*] *Display of Execution Log*

Most of E-Hg menu commands are a sort of short-hand for the actual Mercurial command sequence to be executed, and whose full syntax, and resulting response, will be then anyhow explicitly shown on such dialog boxes as this one:



… or this other one:



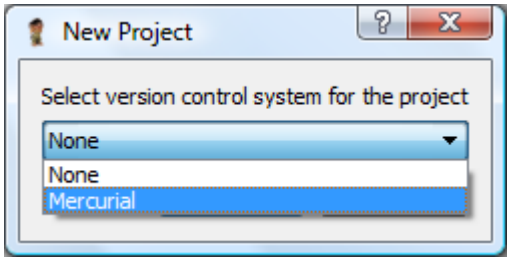
so to gain knowledge of how an E-Hg menu command is actually converted into Mercurial terms.

Viewpoint

<!> In accordance with what already declared [*see:* Scope of this Report, *and also next* [*A03*] *Edit Mercurial Options*], as a rule, the direct contents of these dialog boxes will be hereafter commented only if useful to complete the description of the related E-Hg menu command. Otherwise it will entirely be left to the free interpretation of the possibly Mercurial cognoscenti users.

– –

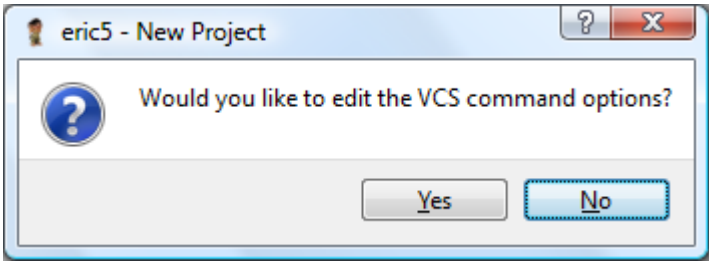
➢ [*A02*] *Select Version Control System*



Dialog box where to select on a drop-down list the desired VC System among those shown as currently available to Eric. With "`None`" as a possible choice.

– –

➢ [*A03*] *Edit Mercurial Options*



Dialog box where to possibility activate a direct editing of options for the Mercurial commands that are going to be generated automatically by Eric.

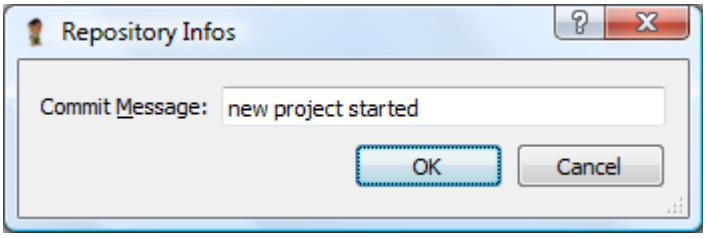
Viewpoint

<!> As already declared [*see:* Scope of this Report], focus of this Report is not on the Mercurial tool as such, but on E-Hg; that is on Mercurial as integrated and available in Eric. Therefore, whenever possible, we'll ignore the here offered possibility for a direct editing of the command options, as we'd rather fully rely onto the equivalent E-Hg menu commands.

– –

➢ [*A04*] *Edit Commit Message*

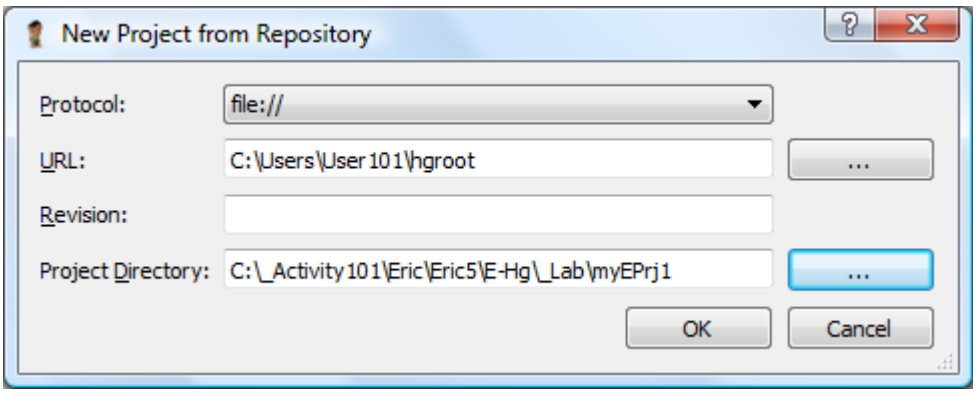
Dialog box where to possibly edit the default "`Commit Message`" to be entered, as required by Mercurial.



– –

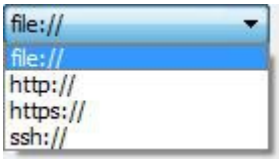
➢ [*A05*] *Copy of "Working Directory"*

Such E-Hg copy action is similar, but not identical, to the so called Mercurial "*clone*" action, as hereafter better explained when describing the E-Hg commands making use of this same dialog box [*see:* `>> New from Repository...`].



Control Box

`Protocol` Drop-down list where to chose the connection protocol to be used use for the data transfer. Available choice:



`file`	Local file system. Default protocol.
`http`	Usual "Hypertext Transfer Protocol", to transfer information over the Internet.
`https`	"Secure" `http`.
`ssh`	"Secure Shell" cryptographic network protocol, for secure data communication.

`URL` Locator of the source Working Directory, according to the chosen `Protocol` [*see*].
<~> Strictly speaking this is not a URL, as it can be either a usual directory path or a trailing URL segment to complete what selected on the preceding `Protocol` field.
<~> Here anyhow we'd appreciate a friendlier default value[17].

`Revision` Optional code to possibly identify a specific Mercurial "Changeset" other than the default Tip, as it is conventionally called in Mercurial terms the most recent one [*see:* Terms and Concepts][18].

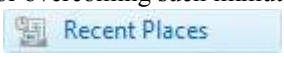
Remark

<!> This "`Revision`" is an option that could be conveniently used to generate a partial Clone of a given Repository, that is an exact copy up to a given Revision, instead of a whole Repository.

- -

`Project Directory`
Destination Working Directory, assumed necessarily local, and existing.
<~> Here too we'd appreciate a friendlier default value.

-<>-

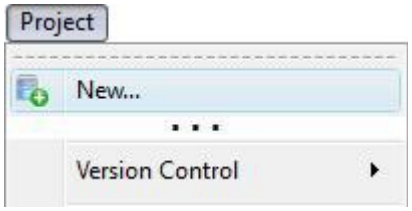
17 <~> A possible way for overcoming such immutable and uncomfortable default path value is to use the "`Recent Places`" 

option as available on the Windows file search control box [*see*].

18 <!> Note that here this parameter implies: "from the Origin *up to* this Revision", whereas in another context the same parameter may well imply: "precisely *this* Revision".

Main Menu: **Project**

This the Eric "`Project`" command menu, where the two first E-Hg commands can be found and used, as hereafter described[19].



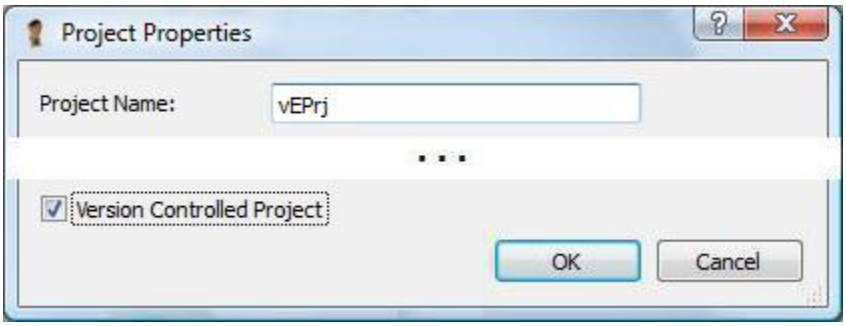
Command List

`New...` `Version Control`

– –

Project > **New...**

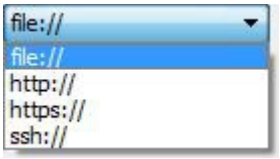
At the bottom of the related "`Project Properties`" control form a check-box is provided to possibly activate the desired Version Control System, at the very inception of a new Eric Project.



19 For a general description of this "`Project`" menu, independently from the presence of a Version Control System, please refer to the dedicated "Eric Technical Report" [*ref.: Eric Web Site,* Foreword].

Remark

<!> Note that here there is no such choice for a "`Protocol`", as with command `>> New from Repository...` [*cf. also:* [*A05, pg. 27*] *Copy of "Working Directory"*]:



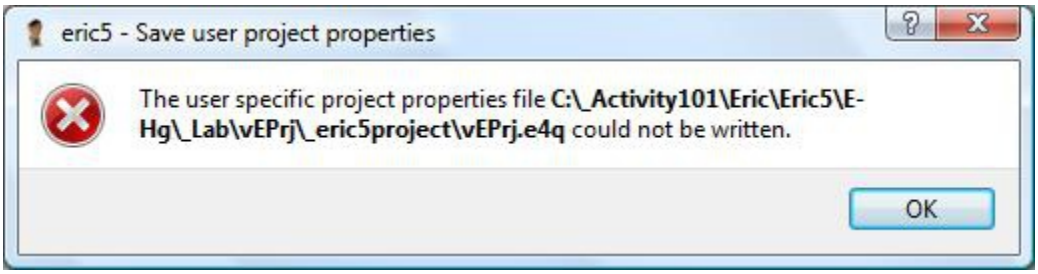
That is to say that the sheer *initialization* of a new versioned Project can be done with the same rules, and limits, of the creation of a usual Eric Project; whereas the *use* of an existing versioned Project can have a much wider scope [*cf.:* `[paths]` *at* Configuration Files*, and* Working in the Internet*, in* Appendix].

– –

E-Hg Execution Sequence

Starting from scratch, that is selecting a brand new directory where to create such new "versioned" Eric Project, this the consequent execution sequence.

<~> Note that if, for any reason, you'd abort and then restart this same sequence, you would encounter such an error condition:

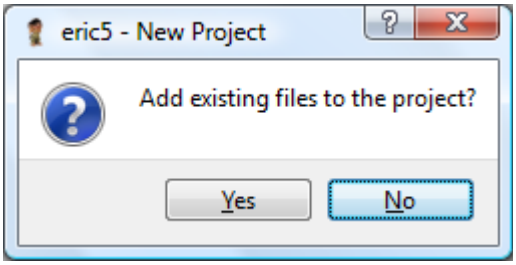


Probably because of some incomplete reset action.

– –

Step 1

First you'll be asked whether to add to the new Project the files possibly already existing within the target directory.



Viewpoint

<~> It's curious that such a possibility is offered even with a brand new Project, with no source file ever developed.

– –

Step 2

- [*A02, pg. 26*] *Select Version Control System*

  Action of explicit selection of the desired VC System.

– –

Step 3

- [*A04, pg. 27*] *Edit Commit Message*

  Action for entering the "`Commit Message`", Mercurial required.

– –

Step 4

- [*A03, pg. 26*] *Edit Mercurial Options*

Here you can intervene directly onto the options to be enabled at the very Mercurial command level. A possibility that we'll be happy to decline, confident that the E-Hg automatism will do anyhow a good work[20].

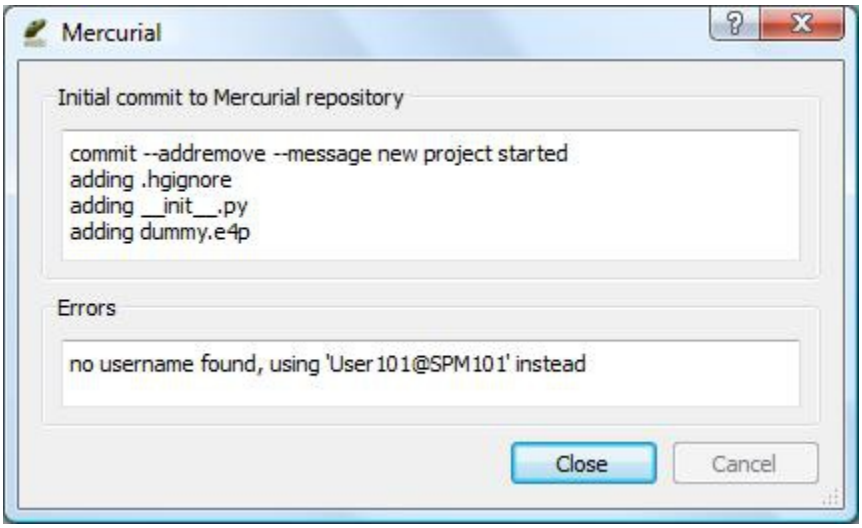
– –

20 <~> Our declared preference in this Report, whenever possible, is to privilege the usage of Mercurial as integrated into Eric, not in itself.

Step 5

- [*A01, pg. 25*] *Display of Execution Log*

Action of showing the log of the specific Mercurial command execution. Which is, in this case, the log of the Commit of the Eric Project at its current stage, that is as the initial Changeset after the creation of the Mercurial metadata Repository [*see:* Terms and Concepts].



This last action is executed under the given "`username`", Mercurial required, here conveniently generated automatically with format: '*<User Name>*`@`*<Computer Name>*'[21], if not already preset as a "`Mercurial.ini`" parameter [*see:* Configuration Files, *in* Appendix].

Viewpoint

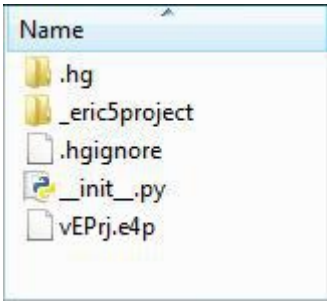
<~> The "`username`" is here displayed as it were to fix an error, as expressed in Mercurial terms. Whereas, if considered in E-Hg terms, it could be well considered simply as a convenient automatism.

– –

E-Hg Command Result

At successful conclusion of this sequence, this the resulting versioned Eric Project directory.

21 [*cf. same* `username` *format at:* Eric Tech. Report, *section* ***R-Pane***: Cooperation]



That is to say, expressed in Mercurial terms, the so called "Working Directory" [*see:* Terms and Concepts].

Viewpoint

In this list, as expected, we see two distinct elements:

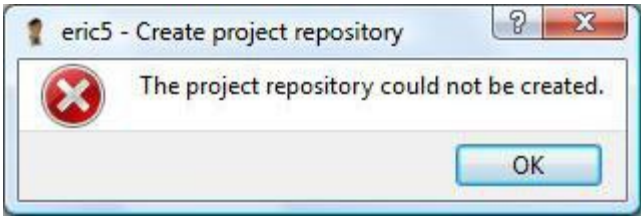
i. The "`\.hg`" Mercurial metadata Repository sub-directory.
<!> With inside yet no default "`hgrc`" configuration file [*see:* Configuration Files, *in* Appendix]; as to say that this is an Origin Working Directory. We found then anyhow convenient to define a user-global "`Mercurial.ini`" configuration file, valid for all Kin Working Directories here around, Origin included; plus a local "`\.hg\hgrc`" file for each specific Kin Working Directory, Origin included.

ii. The usual Eric Project initial stuff.
<~> Plus, to our surprise, as third element, a "`.hgignore`" file, which is certainly not any Eric stuff, but Mercurial's. And rather confusing too, as we assumed that a neat distinction would have been kept and maintained between the two environments. No exceptions admitted.

What it is, and the justification for such a hybrid element, can be hereafter found looking at the `>> Repository Administration > Create .hgignore` command [*see*], but the real honest explanation, in spite of any possible specious justification brought, is that, under this respect, Mercurial technology is still confused and immature.

– –

Whereas, if anything went wrong, you'll be informed this way:



As, for instance, in case of such a declared Mercurial error:

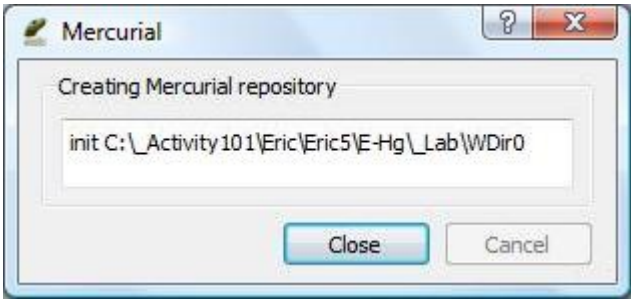
```
Abort:  no username supplied (see 'hg help config')
```

[*see: next* **Case Log**]

- -

Hg Execution

An initial:



followed by:

```
commit --addremove --message new project started
adding .hgignore
adding __init__.py
adding vEPrj.e4p
```

Viewpoint

<~> Note that the above "`Creating Mercurial repository`" heading is rather misleading, as the Mercurial "`init`" command argument is NOT a Repository, but the whole Project's directory; that is the whole Working Directory, said in Mercurial terms.

It is *inside* such main directory that the "`\.hg`" sub-directory is created, as the Mercurial metadata Repository. That said for clarity's sake [*cf. above section:* Terms and Concepts].

- -

Case Log

```
no username found, using 'User101@SPM101' instead
```

Remedy action: configure a `username` parameter as with section Configuration Files [*see in:* Appendix].

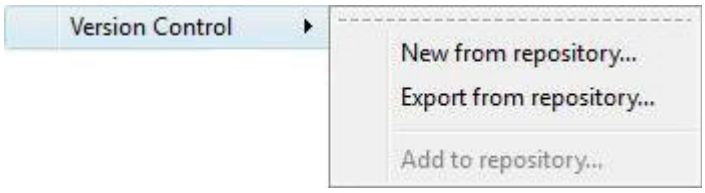
-<>-

Main Menu: Project > **Version Control** – [*3 Cases*]

Three distinct configurations for this menu command, according to the versioned condition of the Project currently opened, with each distinct configuration orderly described hereafter [*see*].

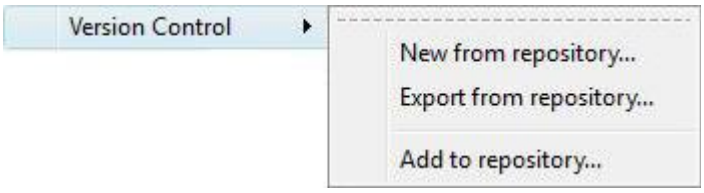
– –

[*C1/3*] No Project Opened



– –

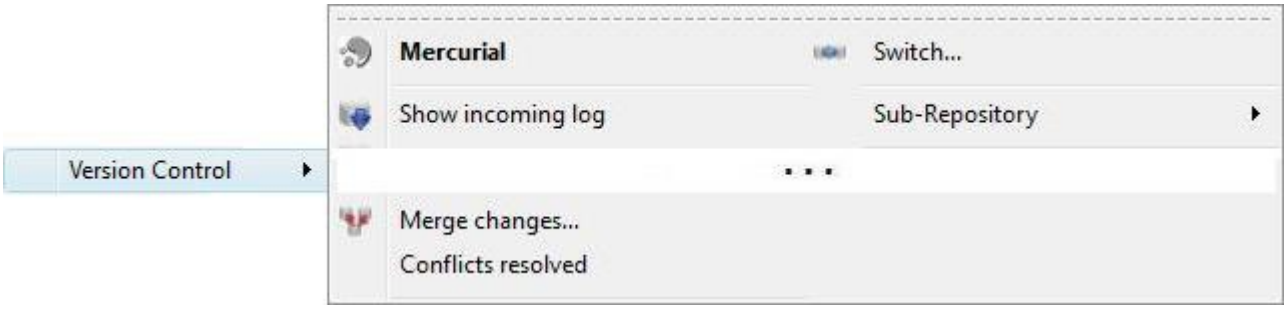
[*C2/3*] Project Opened, Not Versioned



– –

[*C3/3*] Project Opened, Mercurial Versioned

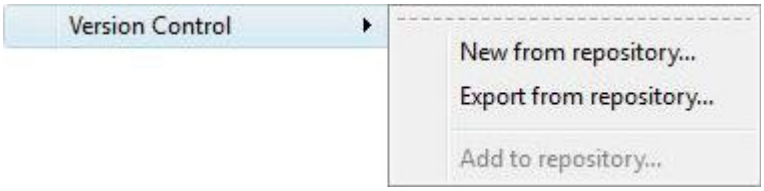
Being this last one by far the most rich `Project > Version Control` menu configuration of the three [*see*].



-<>-

Project > Version Control

[*C1/3*] No Project Opened



Command List

`New from Repository...` `Export from Repository...`

– –

>> New from Repository...

Designed to create a new versioned Eric Project as the Mercurial "Clone" of a master Working Directory, existing somewhere and readable[22].

E-Hg Execution Sequence

Step 1

- [*A02, pg. 26*] *Select Version Control System*

Here the "`None`" option is not available, being equivalent to cancel the sequence.

Step 2

- [*A05, pg. 27*] *Copy of "Working Directory"*

Step 3

- [*A03, pg. 26*] *Edit Mercurial Options*

That we'll be happy to ignore.

Step 4

- [*A01, pg. 25*] *Display of Execution Log*

A look at the resulting sequence log, of course expressed in strict Mercurial terms, could be anyhow useful.

– –

22 <~> Then, why not explicitly call it: "`Clone from Repository...`"?

Hg Execution

```
clone file:///C:/_Activity101/Eric/Eric5/E-Hg/_Lab/WDir0
      C:\_Activity101\Eric\Eric5\E-Hg\_Lab\WDir1
updating to branch default
3 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

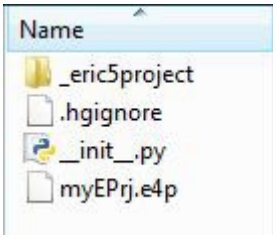
- -

>> Export from Repository...

Similar to the preceding command `>> New from Repository...` [*see*], but for the fact that only the sheer Eric Project is here copied, NOT the Mercurial metadata Repository sub-directory. So that the resulting new Eric Project is NOT versioned.

Result

And this the resulting Eric Project Directory:



<~> Where, again, here still we have the odd "`.hgignore`" file, as a vestigial remnant of the Mercurial past, now having nothing to do with a sheer Eric Project [*cf.:* Result *of command* `Project > New...`].

- -

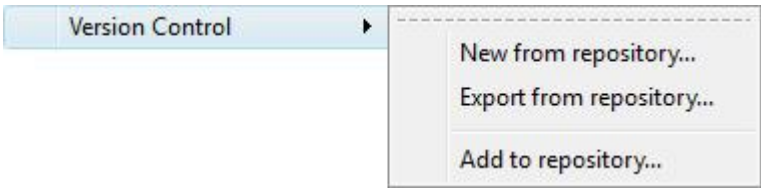
Remark

This is an E-Hg command functionally similar to the corresponding original—and more versatile—Mercurial version, here also available as: `>> Specials > Create Unversioned Archive...` [*see*].

-<>-

Project > **Version Control**

[*C2/3*] Project Opened, not Versioned



Command List

`New from Repository...` `Export from Repository...` [*see:* [*C1/3*] No Project Opened]
`Add to Repository...`

– –

>> **Add to Repository...**

Designed to turn "versioned" an unversioned Eric Project, currently active. It's very similar to the E-Hg command `Project > New...`, with which it shares an equal command sequence [*see*].

Viewpoint

<~> We deem that the name of this command—probably kept for "historical" reasons [*cf. other Tech. Report:* Eric4 Version Control]—is clearly not in line with the actual function here performed, as we know it. That said also in consideration of the similar, but not identical, `Project-Viewer(^)> Add to Repository` *CM* command [*see section:* Context Menu: Project-Viewer(^) Version Control] for which the name, on the contrary, is absolutely appropriate.

Remark

<!> The inverse action of this command, that is turning unversioned a Mercurial versioned Project, can be carried on simply deleting the "`\.hg`" metadata Repository sub-directory. Anyhow noting that:

- <~> We'd like to be confirmed that this is the "correct" way of doing it.
- Its true that the same Project, once unversioned, could be then turned again versioned, but, obviously, this way the entire versionig story will be lost, being reset to the sheer last Tip Changeset.

– –

Step 1

- [*A02, pg. 26*] *Select Version Control System*

  Action to select the of the desired VC System.

Step 2

- [*A04, pg. 27*] *Edit Commit Message*

  Action to enter the "`Commit Message`", Mercurial required.

Step 3

- [*A03, pg. 26*] *Edit Mercurial Options*

  Happy to ignore this action of possible direct editing of the Mercurial command options.

Step 4

- [*A01, pg. 25*] *Display of Execution Log*

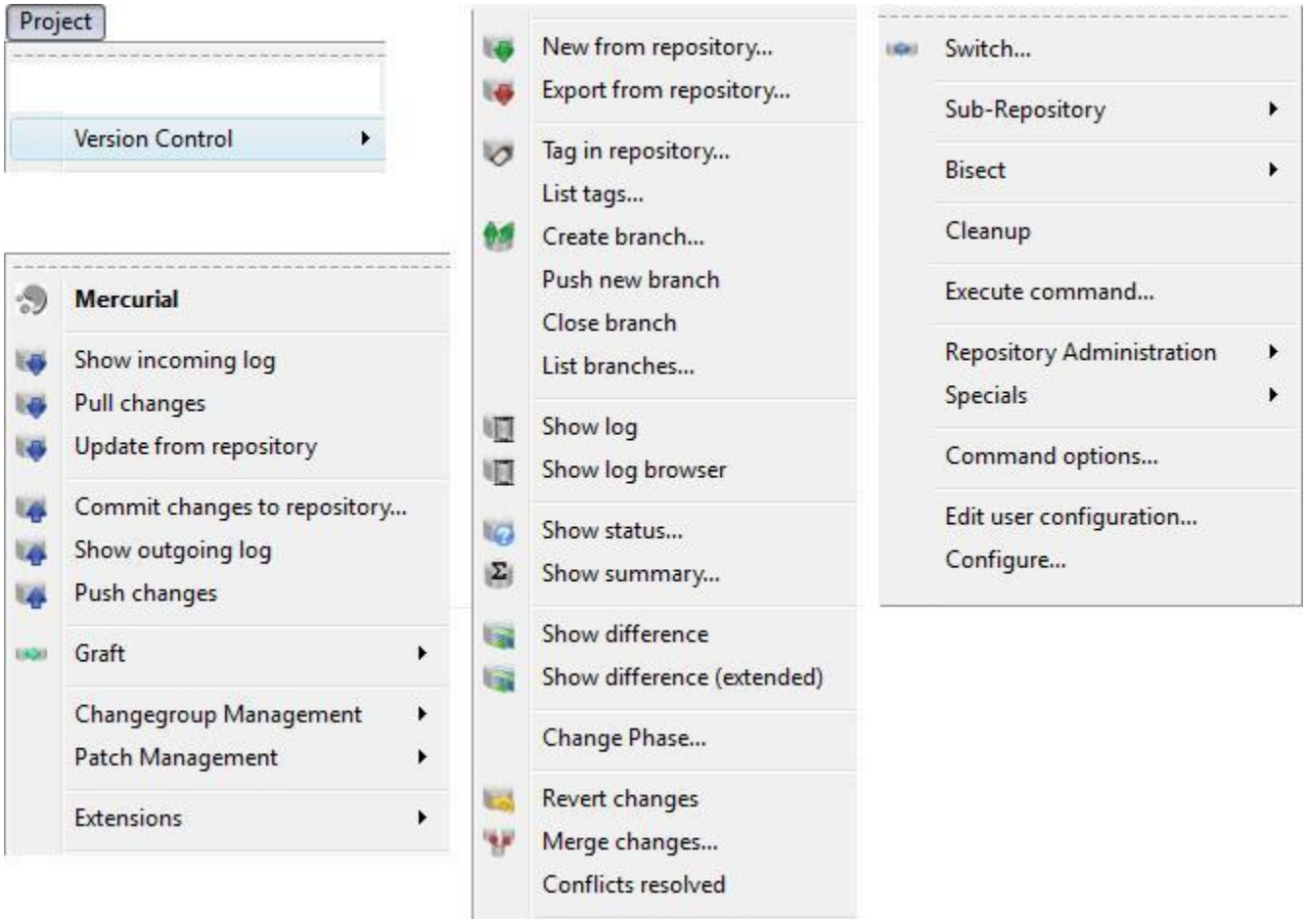
In the end a look at the resulting sequence log, of course expressed in strict Mercurial terms, could be anyhow useful.

-<>-

Project > **Version Control**

[*C3/3*] Project Opened, Mercurial Versioned

Main `Project > Version Control` sub-menu configuration, enabled with an open versioned Eric Project.



Command List

Mercurial[23]

`Show Incoming Log`		`Pull Changes`
`Update from Repository`	`Commit Changes to Repository...`	
`Show Outgoing Log`	`Push Changes`	`Graft`
`Changegroup Management`	`Patch Management`	`Extensions`
`New from Repository...`	`Export from Repository...`	[*see:* [*C1/3*] No Project Opened]

23 "**Mercurial**" the only menu command here written in boldface, as for a title, a heading; no other reason.

```
Tag in Repository...        List Tags...                 Create Branch...
Push New Branch             Close Branch                 List Branches...
Show Log                    Show Log Browser             Show Status...
Show Summary...             Show Difference              Show Difference (Extended)
Change Phase...             Revert Changes
Merge Changes...            Conflicts Resolved           Switch...
Sub-Repository              Bisect                       Cleanup
Execute Command...          Repository Administration    Specials
Command Options...          Edit User Configuration...   Configure...
```

- -

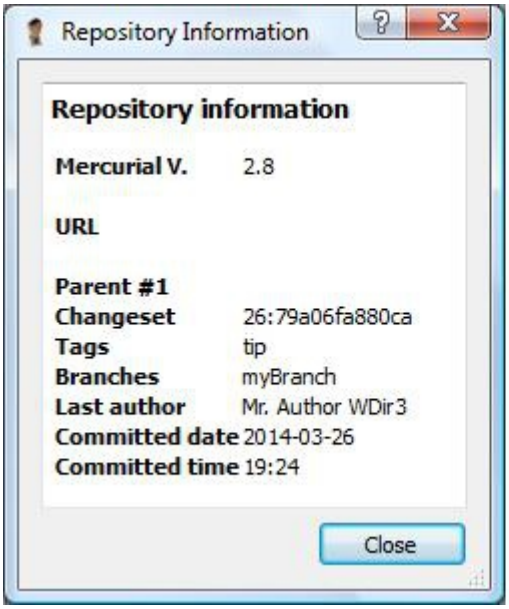
>> Mercurial

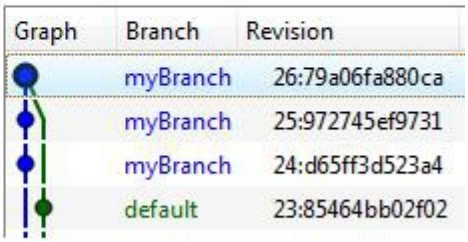
Designed to show an essential "`Repository Information`" box about the current Working Revision and the Parent(s) from which it is derived.

Parent Revision, which usually is the most recently Committed one, but not necessarily, because any other Revision can be selected with command `>> Switch...` [*see*] so to become Active, and also because the current Working Revision can be the place where *two* distinct Committed Revisions can be merged [*cf.:* `>> Merge Changes...`]. Note that it is in this case that a Working Revision will happen to be Descendant of *two* Parent Revisions, and with no Active Revision currently defined [*as can be verified via:* `>> Show Log Browser`].

Remark

<!> Note also that the not-so-subtle distinction between Working Revision (which is shown by this command) and Active Revision (which is NOT shown by this command) is here particularly evident when an Active Revision is declared the only Parent of the related Working Revision even if, in itself [*as with next example*] it is actually the Descendant of *two* Parent Revisions.





In other words, the fact that this command is about the Working Revision, and not about the current Active Revision, is shown in this example by the enumeration of just *one* Changeset, the No 26, as Parent, not *two*, the No 25 and No 23, which are the current Active Revision's Parents [*see*].

– –

Control Box

`Mercurial V.` Product and version No

`URL` Current "`default`" remote Repository's URL[24], as defined in "`Mercurial.ini`" or "`.hg\hgrc`" configuration file [*see:* Configuration Files, *in:* Appendix], valid for such commands as `>> Pull` / `Push Changes` [*see*], usually defaulted to the address of the Parent Repository. A field here left blank if not defined, typically in case of an Origin Repository, as not cloned from any Parent Repository.

24 N.B.: Strictly "`default`", not the other similar "`default-push`" parameter, here ignored.

Viewpoint

This "URL" is a typical example of label so generic to be completely meaningless, if not even misleading. The point is not to inform that this is a URL, which, when not void, results pretty obvious simply looking at it. To say that a URL is a URL is tautological, therefore meaningless. The real point is: what kind of URL is this? Pointing to what? What for? THAT should be the precious information conveyed by a meaningful label.

Anyhow, by try-and-error we realized that:

- In case of a Child Repository, this "URL" is defaulted to the relative Parent's address—that is, unless manually changed as described at section Configuration Files [*see in:* Appendix]—, here written with a different format according to the protocol in use. That is, not necessarily a proper URL, as it can be well a directory path [*cf.:* `Protocol` *field, of command* `>> New from Repository...`].
- In case of an initial Origin Repository, therefore without a Parent of its own, this "URL" field is simply left void [*as in the box hereafter*];

Reaching this way an apex of obscurity: a inexpressive label followed by a void field.

– –

`Parent Changeset`
: Ordinal No and Changeset identifiers of up to two possible Parents of the current Working Revision, with format: <local `No`>:<global `Hex`> [*cf.:* `>> Show Summary...`].

`Tags`
: If present, it is a text label assigned as a name to the Parent Changeset. [*see:* `>> Tag in Repository...`]. The most recent one is automatically named: "`tip`".

Remark

N.B.: Plural "`Tags`", as there may be more than one single Tag associated to the same Changeset. That is: the automatic Tip plus possibly another one assigned manually with command `>> Tag in Repository...` [*see*].

– –

`Branches`
: Sometimes this other field, after the "`Tags`" label.

Viewpoint

<~> About this "`Branches`" label, and the related info, we have some serious Perplexities:

P1) Which is the current Branch is a relevant information that we'd like to see *always* here shown, as there is *always* defined a Branch, the so called "`default`", when not a custom one.
<~> Indeed what's the current Branch is a crucial information, well worth to be here always in display as it is, for instance, with command: `>> Show Log` [*see*].

P2) Maybe this info is here offered only with the very single Changeset where a new Named Branch is defined first. If so, we wouldn't like it, but we'd accept it as a possible explanation of the above P1). Provided that we'd see also the "`default`" Branch shown at the Changeset where defined first, that is the Origin Changeset No 0.

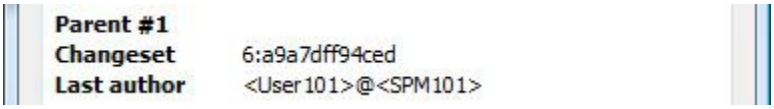
P3) <?> But then we couldn't understand why this label reads plural "`Branches`", as only *one* Branch can be defined in a Changeset (as we've verified with command `>> Create Branch...`, where, in any case, *only the last* of more than possibly one will be actually created).

– –

`Tags`
`Branches`

<?> Sometimes here we've also ...
… spotted *both* these labels [*as with next example*]

<?> Whereas sometimes, between the labels "`Parent Changeset`" and "`Last author`", where "`Tags`" or "`Branches`" usually lies, nothing appears at all [*as with next example*].



<?> This way turning our Perplexity to confusion. We mean, if in case of a void value the rule is that the label should be missing, then, by the same token, also the above empty "`URL`" label should disappear; shouldn't it?

– –

`Last author`

"`username`" of the Parent Revision's Commit [*cf.:* `>> Commit Changes to Repository...`, *and* Configuration Files *section in* Appendix]
<?> Why "`Last`"? A Commit, is a Commit, is a Commit. What's the sense of such specification here? Besides, next parameter is a date / time [*cf.*], obviously without

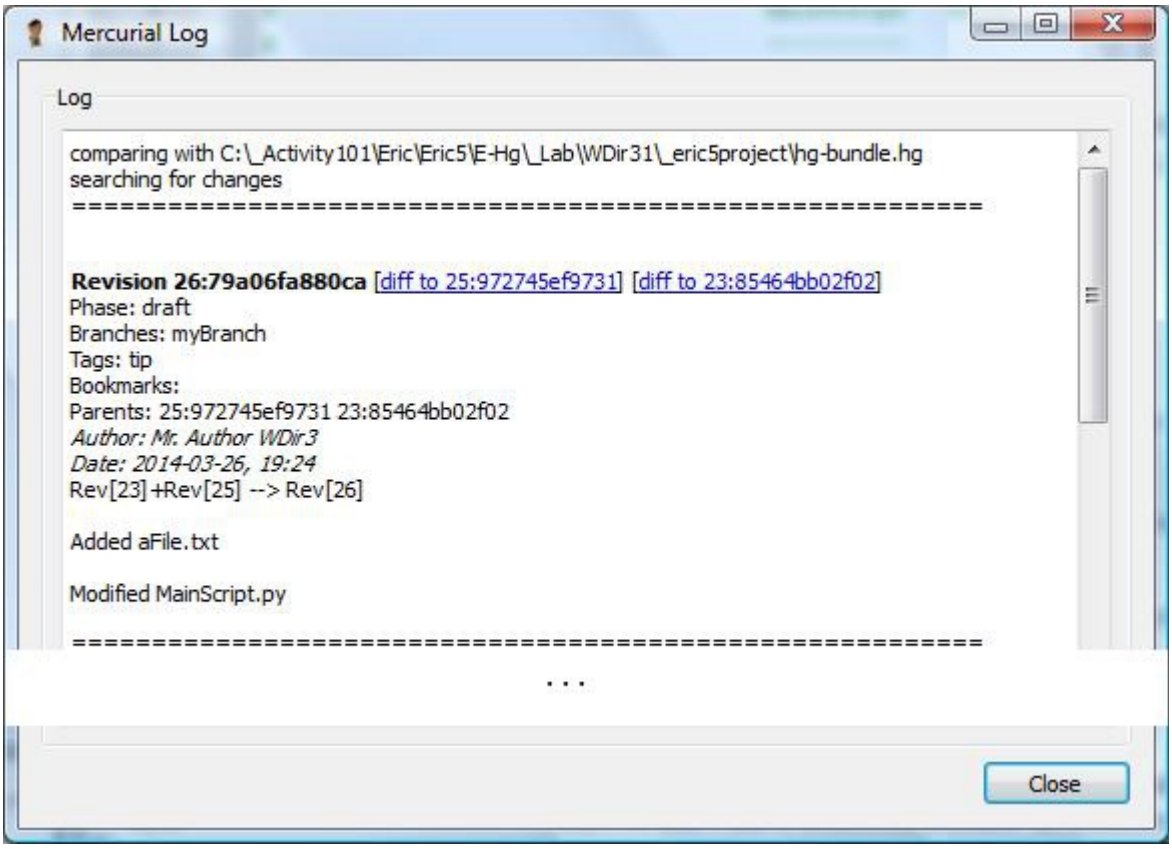
any such "`Last`" specification. What's the difference?

`Committed` date / time.

– –

>> Show Incoming Log

Designed to display a "`Mercurial Log`" box of the remote Kin Repository changes to be possibly downloaded ("Pulled") into the local Repository [*see:* `>> Pull Changes`]. Remote source Repository defined as Sender by the "`default`" parameter, set into the "`[paths]`" section of `Mercurial.ini` and `.hg\hgrc` configuration files [*see:* Configuration Files, *in* Appendix]; usually defaulted to the Parent's URL, at the initial "Clone" action [*see:* `>> New from Repository...`]. Command symmetrical of `>> Show Outgoing Log` [*see*].



Log display with different formats, selectable with `>> Configure... - Use Log Browser for incoming / outgoing log` [*see*]; that is:

- A simple text "`Mercurial Log`" display, as with command `>> Show Log` [*see*] (default condition), or:
- A rich graphic "`Mercurial Log (Incoming)`" display, as with `>> Show Log Browser` [*see*].

Case Log

`No log available for '***'`, or a "blank" display
In case of no new incoming changes to be shown.

`abort: repository default not found!`
In case of no "`default`" path configured [*see:* Configuration Files, *in* Appendix]
<~> In this case it would be nice to have this command disabled, as it simply couldn't apply.

– –

>> Pull Changes

Designed to "Pull" (i.e.: download[25]) locally all new Changesets from a remote Kin Repository, as listed available with command `>> Show Incoming Log` [*see*].

This command, together with its "`Push Changes`" companion [*see*], and the related `>> Show Outgoing Log` [*see*], is the main E-Hg tool for sharing Revisions among developers of the same versioned Project[26], provided they enjoy all required permission rights to read/write their Kin Working Directories[27].

Features

- Sender Kin Repository address is recorded as:
  `default = <URL>`
  parameter into the "`[paths]`" section of the configuration file `.hg\hgrc`; in priority over the same parameter possibly available in `Mercurial.ini` [*see:* Configuration Files, *in* Appendix].
- Initial `default = <Parent URL>`
  parameter is set automatically at the cloning to a Child Working Directory. But then, it can be freely changed with command: `>> Repository Administration > Edit Repository Configuration...` [*see*]. Anyhow note that there can be only one possible "`default`" Sender

25 About actual downloading execution see also the "`Prefere Unbundle`" setting, of `> Configure...` [*see*].

26 But not the only one, as described with section Working in TeamWorking in Team [*see in:* E-Hg PrimerE-Hg Primer].

27 Which, of course, is a specific managerial and technical condition, whose analysis is off the scope of this Report.

Repository at any given time.

- This Pull command cannot execute if no legal URL address is available.

Viewpoint

<~> Therefore it would be nice & friendly to have this "`>> Pull Changes`" and the associated "`Show Incoming Log`" commands automatically "dimmed"-disabled in case of related "`default = <URL>`" parameter undefined. For instance, a master Repository, not having been cloned from any Parent, typically can have no such `default = <URL>` parameter defined.

– –

- This Pull command can be turned dummy (i.e.: operatively disabled) setting the parameter:
  `default = <same current Working Directory URL>`
  Somehow it's a way to "isolate" a Repository.
- New Changesets possibly available to be Pulled from the Sender Repository can be explored beforehand with command `>> Show Incoming Log` [*see*]. The list of Pull-able Changesets begins after the last one in common to both Sender and Receiver Repository.
- In case of changes executed independently at the Receiver side on a given Working Revision, a Pull action will generate a topological Branch, with the current Working Revision unaltered. Topological Branch that can be then reduced with the usual: Merge – Resolve – Commit sequence [*see:* `>> Merge Changes...`].
- The action of Pulling executed at the Receiver side is equivalent to the action of "Pushing" executed at the Sender side towards the same Receiver [*cf.:* `>> Push Changes`], in the sense that the same Changesets are copied.
- Once Pull executed, current Working Revision remains the same, and the "`Incoming Log`" turns empty. Equivalently the `>> Show Outgoing Log` [*see*], seen at the Sender side towards the same Receiver, turns empty.

Remark

In case of a versioned Project developed by a team of developers who, for whatever reason—managerial or technical—, do not have direct access to the Network of Kin Working Directories, in particular for writing, the E-Hg tool that could be used instead of this one is the `>> Changegroup Management` [*see:* Working in TeamWorking in Team*, in* E-Hg Primer].

– –

Hg Execution

```
pull -v
```

– –

>> Update from Repository

Designed to move up to the Head Changeset of the current Branch, and make it the current Active one. Note that this is NOT necessarily the most recent Changeset in absolute, but only that relative to the Branch currently active [*see:* `Create Branch...`].

Command[28] equivalent to the default condition of `>> Switch...` [*see*].

Remark

<!> In case of any changes already introduced on the current Working Revision you're currently moving from, it is important to be aware that:

- Moving to a different Revision necessarily implies a "`merge`" process of the changes already introduced [*see:* `>> Merge Changes...`], possibly with consequent conflicts. Which could not be exactly what you had in mind.
- Two actions for avoiding what at the previous point:
  - Eider first Commit current Working Revision [*see:* `>> Commit Changes to Repository...`], and then move to the other desired one;
  - Or simply drop all changes with command `>> Revert Changes` [*see*]. This same drop action can be executed also after the movement, possibly followed by a declaration of `>> Conflicts Resolved` [*see*].
- Note that in case of a dummy move, that is a move aimed at the same current Revision, nothing will happen, in particular the current Working Revision will not be reset to the original contents.

Hg Execution

```
update -v
```

Case Log

```
warning: conflicts during merge.
merging MainScript.py incomplete! (edit conflicts, then use 'hg resolve -mark')
```

As it may happen with changes introduced into the current Working Revision.

– –

>> Commit Changes to Repository...

Central action of the versioning process, designed to record ("Commit") into the metadata Repository the current state of the Working Directory in form of a new Revision; action executed according to what specified on the related control box.

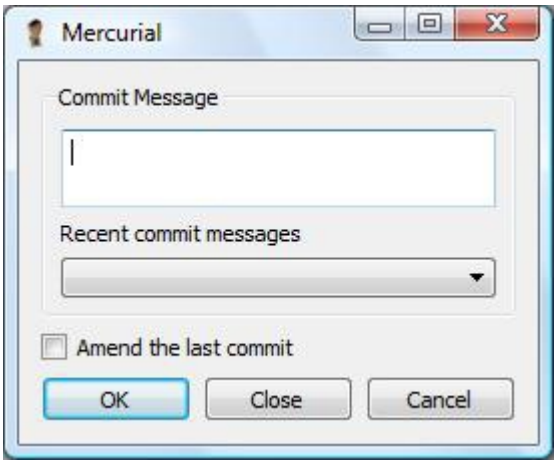
28 <~> That is to say that, in spite of its name, not so explicit, this command is nothing else than a fixed "`Switch...`".

Viewpoint

<~> But then if you think, as we naively did, that this is all what you need to get your current Eric Project actually version controlled, you will be bitterly let down. Indeed, beside this Commit action, you must also:

- Explicitly tell Mercurial of any file possibly added to Eric, in spite of the fact that you have already declared, at its very inception, that a Project is intended to be Version Controlled [*see:* <。。。>]. Do it on L-Pane, with Project-Viewer context command:
  `Project-Viewer(^)> Add to Repository` [*see*]

– –



Control Box

`Commit Message`
: Required text field, possibly derived from a history drop-down list [*see next control*].

`Recent commit message`
: Message history drop-dpwn list [*cf. former control*].

`Amend last commit`
: Check-box to operate in "`--amend`" mode on the very parent Revision, re-Commited with the same `No`, but with a new Hex `Id`, without creating a new one.

Hg Execution

```
commit -v --message ***
commit -v --amend --message ***
```

Case Log

`abort: no username supplied`

Remedy action: configure a `username` parameter as with section Configuration Files [*see in:* Appendix].

– –

Undo and Redo of Commit Changes

The Commit of a Revision is the fundamental recording act for a Versioned Project. In principle Commits, like diamonds, are forever. Well, almost forever, as there are some actions provided to let you change your mind. Here the list.

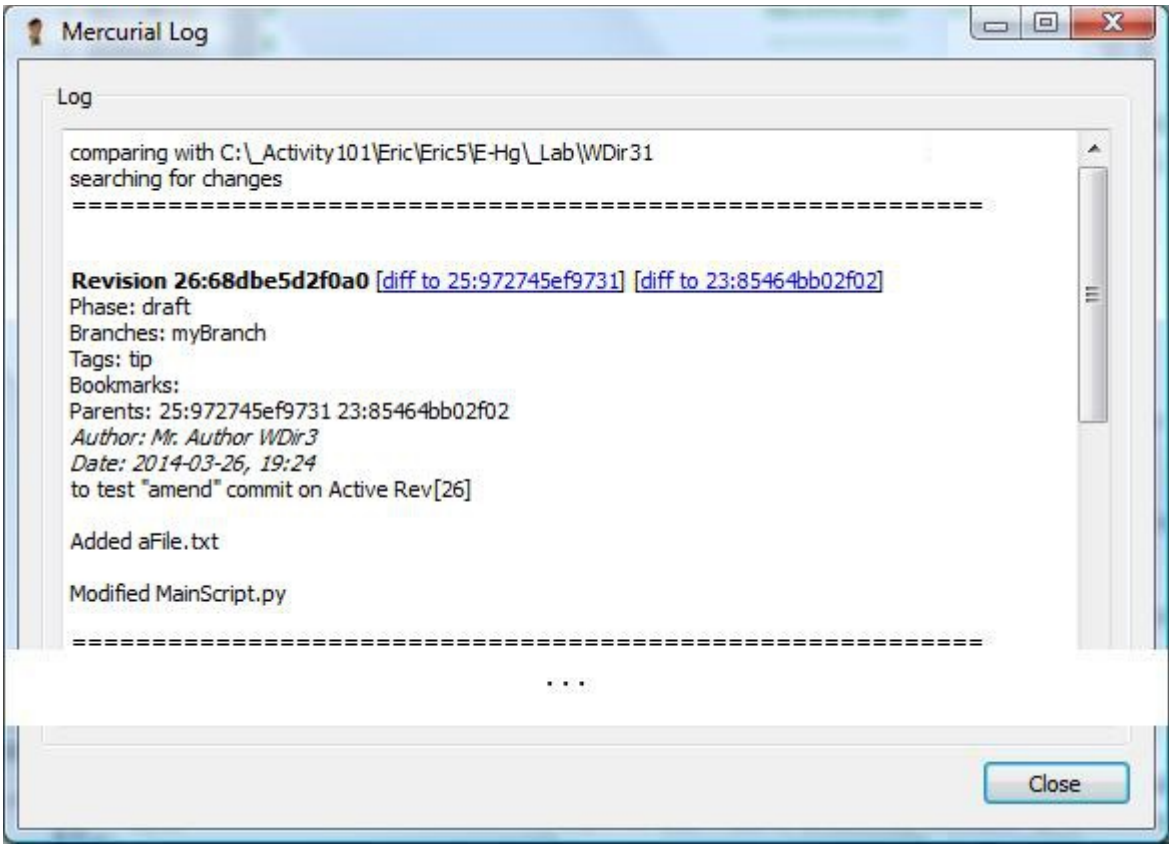
- The first convenient Undo/Redo tool to be considered is simply the standard Eric editing tool. In other words: before an actual E-Hg Commit action, just think twice.
- Then, just *before* a Commit, you could simply drop all changes and revert the current Working Revision to its original state, with command: `>> Revert Changes` [*see*]. Here also think twice, as no "redo" is provided.
- Then, just *after* a Commit, being this a transactional action, you could change your mind via: `>> Repository Administration > Rollback Last Transaction` [*see*]. As a "redo" action, just Commit again the same Working Revision.
- Old Committed Revisions can anyhow be "Branch" edited after being turned again Active, with command `>> Switch...` [*see*]. This Switch and edit action is somehow equivalent to a Do/Undo/Redo change session.
- Old Committed Revisions could be undone with command: >> Repository Administration > Back Out Changeset [see]. To Redo just "Switch..." again to the initial Revision.
- All Revisions Committed after a given one could be dropped by partially cloning a versioned Project making use of the "`Revision`" option with command `>> New from Repository...` [*see*].

– –

>> Show Outgoing Log

Designed to display a "`Mercurial Log`" box of the local Repository changes to be possibly uploaded ("pushed") into a remote Kin Repository [*see:* `>> Push Changes`]. Remote Repository defined as Receiver with "`default`" or "`default-push`" parameters, set into the "`[paths]`" section of `Mercurial.ini` and `.hg\hgrc` configuration files [*see:* Configuration Files, *in* Appendix]; usually

defaulted to the Parent's, at the initial "clone" action [*see:* `>> New from Repository...`]. Command symmetrical of `>> Show Incoming Log` [*see*].



Log display of different format as selected with `>> Configure... - Use Log Browser for incoming / outgoing log` [*see*]; that is:

- A simple text "`Mercurial Log`" display, as with command `>> Show Log` [*see*] (default condition), or:
- A rich graphic "`Mercurial Log (Outgoing)`" display, as with `>> Show Log Browser` [*see*].

Case Log

`No log available for '***'`, or a "blank" display
: In case of no new outgoing changes to be shown.

`abort: repository default-push not found!`
: In case of no "`default-push`" nor "`default`" path configured [*see:* Configuration Files, *in* Appendix]

<~> In this case it would be nice to have this command disabled, as it simply couldn't apply.

– –

>> Push Changes

Designed to "Push" (i.e.: upload) to a remote Receiver Repository all new local Changesets, as listed available with command `>> Show Outgoing Log` [*see*].

This command, together with its "`Pull Changes`" companion [*see*], and the related `>> Show Incoming Log` [*see*], is the main E-Hg tool for sharing Revisions among developers of the same versioned Project[29], provided they enjoy all required permission rights to read/write their Kin Working Directories[30].

Features

These Push ***Features*** are perfectly symmetrical to those seen at `>> Pull Changes` [*see*], but for the following two cases.

- If any change has been independently introduced into the aimed Receiver Working Directory, the Push action should necessarily created a new remote Head on a Topological Branch, which is not feasible, as it would there make unclear which is the Head in use. In such a "`new remote head`" case you'll be informed, with an "`abort`" message, that the Push action has not been executed.

  The suggested remedy is first a plain Pull and merge action executed locally, with possible conflicts resolved, then followed by the desired Push.
  In other words:
  <!> In case of a contemporary not-empty Incoming *and* Outgoing Log, it is advisable first to execute a Pull, and only then a Push.

  – –

- Besides the usual "`[paths]`" parameter: `default = <URL>`
  Push command accepts, in priority, this other parameter [*see:* Configuration Files*, in* Appendix]:
  `default-push = <receiver URL>`

  Note that this Push command can be turned selectively dummy (i.e.: disabled) setting:
  `default-push = <same current Working Directory's URL>`

29 But not the only one, as described with section Working in Team [*see in:* E-Hg Primer].
30 Which, of course, is a specific managerial and technical condition, whose analysis is off the scope of this Report.

Hg Execution

```
push -v
```

Case Log

```
abort: push creates new remote head 3d8a7f60d535!
(pull and merge or see "hg help push" for details about pushing new heads)
```

Not so clear an "`abort`" message, meaning:

"No real Push action has been executed, because it would have created a second Head into the destination Repository, which is not permitted [*see above:* ***Feature***]. Here you are advised first to execute a preliminary local Pull and Merge action."

Remark

<~> For adventurous users, loving risk, we signal the existence of the `>> Specials > Push Changes (force)` command [*see*], provided to execute anyhow such Push action in "`--force`" mode.

– –

>> Graft

A function for selectively coping individual Changesets from different Branches, without altering the distinction between the two Branches [*as with:* `>> Merge Changes...`].

Command List

`Copy Changesets` `Continue Copying Session`

– –

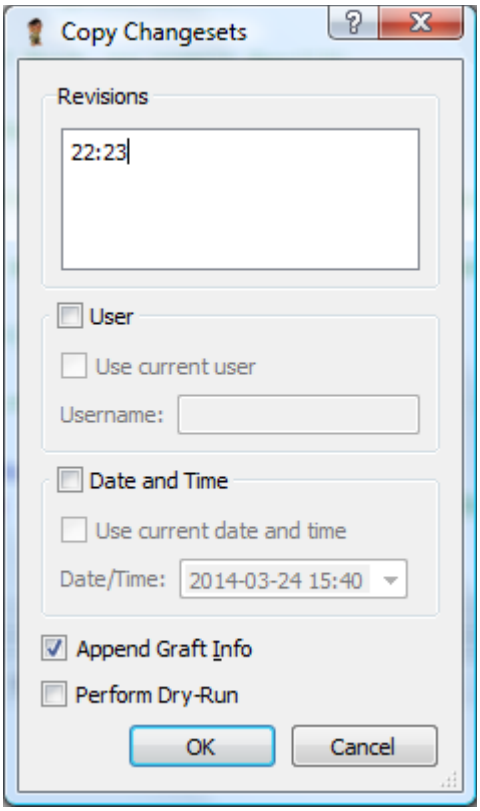
>> Graft > Copy Changesets

Designed to copy and merge into the current Branch the contents of individual Changesets belonging to other Branches[31]. Changesets already copied will be ignored. User name, date and

31 Action commonly referred to as "backporting" or "cherry-picking". Same action of "`Copy Changesets`" button, as with `>> Show Log Browser` [*see*].

description are defaulted from the source Changesets.

In case of merge conflicts, this copy session will be interrupted so to intervene manually, and then resume via command `>> Graft > Continue Copying Session` [*see*].



Control Box

`Revisions` Usual Revision sets selection, as with command `>> Show Log` [*see*].

`User` Check-box to change user info

`Date and Time` Check-box to change date info

`Append Graft Info`
That is, in addition to the Commit message, so to tell from which Changeset the changes were taken.

`Perform Dry-Run`
Check-box for a dummy execution, that is: no actions, just print output [*usual original Mercurial option:* `-n, --dry-run`].

Hg Execution

```
graft --verbose --log 15:17
```

Case Log

`warning: conflicts during merge.`

Conflicts to be solved manually, followed by a `>> Conflicts Resolved` and `Continue Copying Session` action; or a `>> Update from Repository` [*see*] to abort.

`abort: graft in progress`

A further `Continue Copying Session` [*see hereafter*] may be required, or a `>> Update from Repository` [*see*] to abort.

– –

>> Graft > Continue Copying Session

Designed to continue the current `>> Graft > Copy Changesets` session possibly interrupted for solving manually an intervened merge conflict. After actual manual intervention, a `>> Conflicts Resolved` command [*see*] should be entered before this command. More than one such `Continue Copying Session` action may be required.

Hg Execution

```
graft --continue --verbose
```

Case Log

`abort: no graft state found, can't continue`

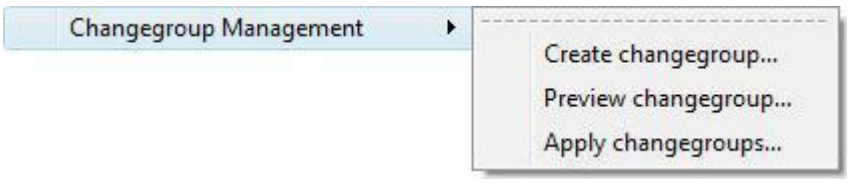
Reason: there is no interrupted `>> Graft > Copy Changesets` session under way. <~> In such a case we'd expect this command be simply disabled.

– –

>> Changegroup Management

Menu item designed for the creation and use of Changegroups, that is compressed archive files of specific sub-sections of a Repository, typically used for distribution purposes in a collaborative development environment where, typically, direct Pull / Push isn't a convenient or available action. That is, a sort of an "off-line" Pull / Push equivalent system [*cf.:* Working in Team, *in* E-Hg Primer – Versioning in Practice].

A specific E-Hg tool based upon the original Mercurial "`bundle`" command [*cf.:* Mercurial Command Reference, *at* Mercurial Package, *in* Appendix].



Command List

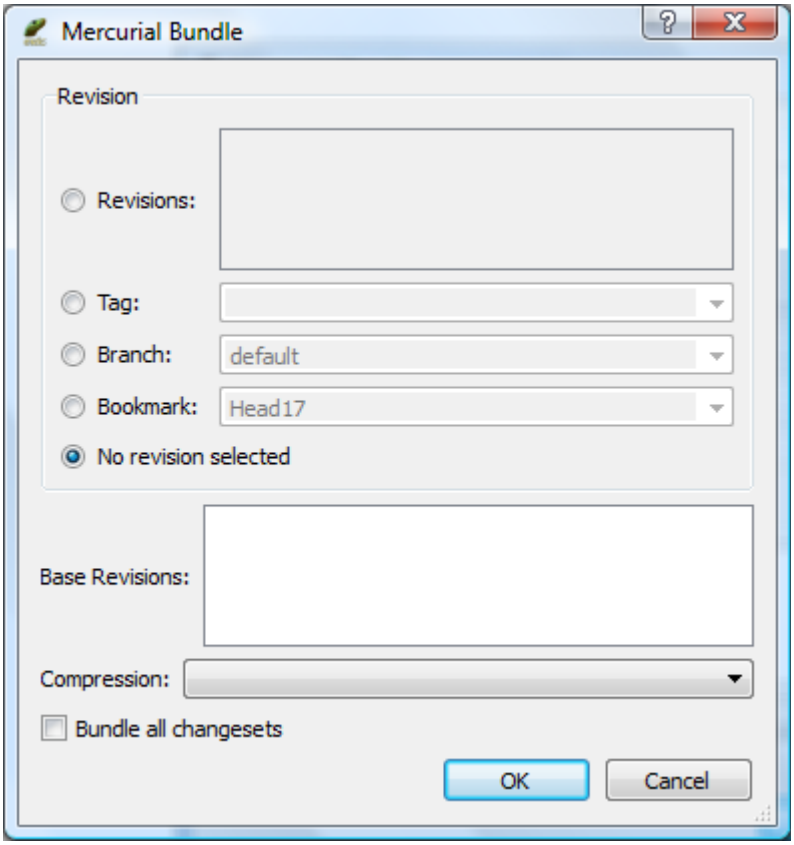
`Create Changegroup...` `Preview Changegroup...` `Apply Changegroups...`

– –

>> Changegroup Management > **Create Changegroup...**

Designed to create a "Changegroup" compressed file where to record a selected collection of Changesets, according to the parameters entered on the command's control box. Default location for the generated file is the current Working Directory, default extension "`.hg`".

Changegroup is a convenient way for storing selected portions of versioned Projects typically to be then transferred where no direct Pull / Push actions are permitted, or desired [*cf.:* `Apply Changegroups...`].



Control Box

`Revisions` Usual Revision sets selection, as with command `>> Show Log` [*see*].

`Tag`, `Branch`, `Bookmark`
Usual Revision selectors, as with command `>> Switch...` [*see*].

`No revision selected`
: Implies all Revision beyond the `Base` [*see next point*]. Default condition.

`Base Revisions`
: Revisions to be ignored, typically as assumed already present in the target Repository, with same Revision sets selection syntax as with `Revisions` [*see above*]. Example: `Base=n` and `Revisions=tip` implies a Revision interval = [`(n+1), tip`]

`Compression`
: Drop-down list where to select a Mercurial standard compression: `bzip2`, `gzip`, or `none` (default: `bzip2`)

`Bundle all changesets`
: Check-box to include all available Changesets (instead of a particular selection).

Hg Execution

```
bundle --rev tip --base 2  ***.hg

bundle --all ***.hg
```

\- -

>> Changegroup Management > **Preview Changegroup...**

Designed to show a "`Mercurial Log`" style preview of a given Changegroup file [*cf. command:* `>> Show Log`], whether applied to the current Repository [*see:* `Apply Changegroups...`].

* * *

A command for identifying the contents of a Changegroup, with some limitations:

- If called while operating on the very Working Directory where it was generated, no other info than: "`No log available for '***'`" will be shown, to say that there is no difference with the local Revisions [<~> *Of course, we should say.*].
- If called while operating on another Kin Repository, you'll be informed about the possible difference with the local Revisions, not about the whole contents of the very Changegroup.
- If called while operating on a not-Kin Repository, all you'll get is an error message of the kind: `abort: 00changelog.i@c2313465bdf8: unknown parent!`

Case Log

`No log available for '***'`

<~> In case of void Changegroup action, for no difference with local Revisions.

– –

>> Changegroup Management > **Apply Changegroups...**

Designed to apply selected Changegroup files to the current Repository [*cf.:* `Create Changegroup...`].

Viewpoint

<~> "`Changegroups`" here in the plural seems to us rather inappropriate. It's true that the original Mercurial "`unbundle`" command assumes as possible the entering of more than one file at once, but here we don't see how it could be done; nor, frankly, if it were a wise possibility.

Hg Execution

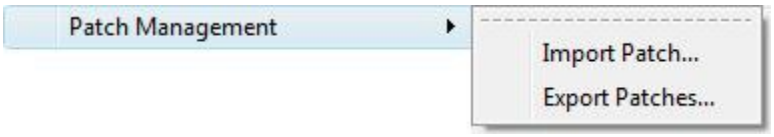
```
unbundle --update --verbose ***.hg
```

– –

>> Patch Management

A mechanism based upon the standard "`*.patch`" files[32], identical to the "`*.diff`" files generated with command `>> Show Difference`, `Save` button [*see*], equally usable as "Patch" files as here considered.

A diverse way of recording and applying changes executed on top of versioned source files, as an independent alternative to such usual Pull / Push actions, would they be unpractical or even impossible [*cf. section:* Working in Team, *in* E-Hg Primer – Versioning in Practice].



Command List

`Import Patch...` `Export Patches...`

– –

>> Patch Management > Import Patch...

Designed to import a standard "`*.patch`" or "`*.diff`" source file into the current Working Directory, and then possibly Commit it right away.

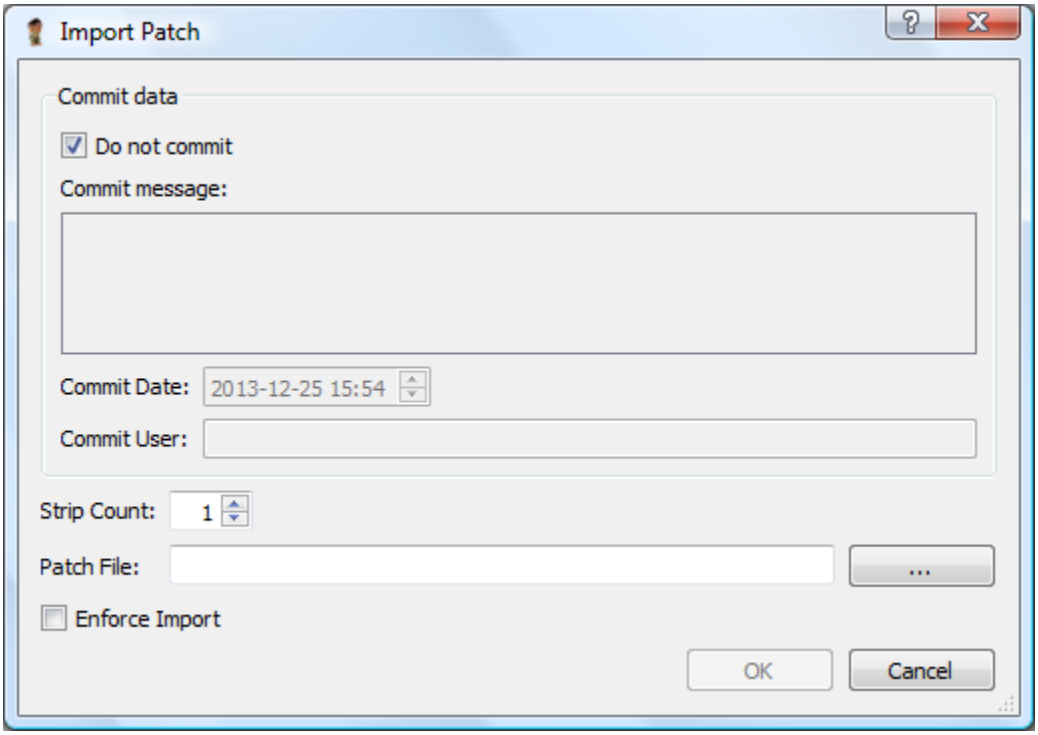
Remark

Note that this is an action indifferent to the synchronization between possibly Kin Projects. For instance:

- A Tip Revision can be well Exported and then Imported as a Patch into any Kin Project, and there as a new Tip Revision, with a *different* hex Id. Even *twice*, that is the same Patch.
- Any Revision can be Exported and then Imported as a Patch into any other Project, "Kin" or not; always on top of the current Working Directory.

With no limits to the "de-syncronisation" [<~> *and also the messing up*] of Projects. That is to say: be aware, and cautious.

32 <~> In case you'd wander, nothing to do, if not the name, with the "Patch" subject as treated in Chapter 12 of the O'Sullivan's "Definitive Guide" (which, by the way, is about the so called MQ "Mercurial Queues").



Control Box

`Commit ...` Optional Commit action, and parameters

`Strip Count` This the original Mercurial definition: "*directory strip option for patch.*"
[*<.?.> questionable (obscure) point, further investigation required*]

`Patch File` Path to the "`*.patch`" or "`*.diff`" file to import[33].

`Enforce Import`
To force the Import action even when some changes have been already introduced on the current Working Revision.

Hg Execution

```
import --verbose --no-commit C:\_Activity101\Eric\Eric5\E-Hg\_Lab\vEPrj2\
      E-HgPatch\vEPrj2_5_4b862abae604_1_of_1.diff
```

33 <~> Not so convenient default, and without memory. At least it could be arranged equal to the `Export Patch...`'s [*see*].

Case Log

`abort: uncommitted changes`

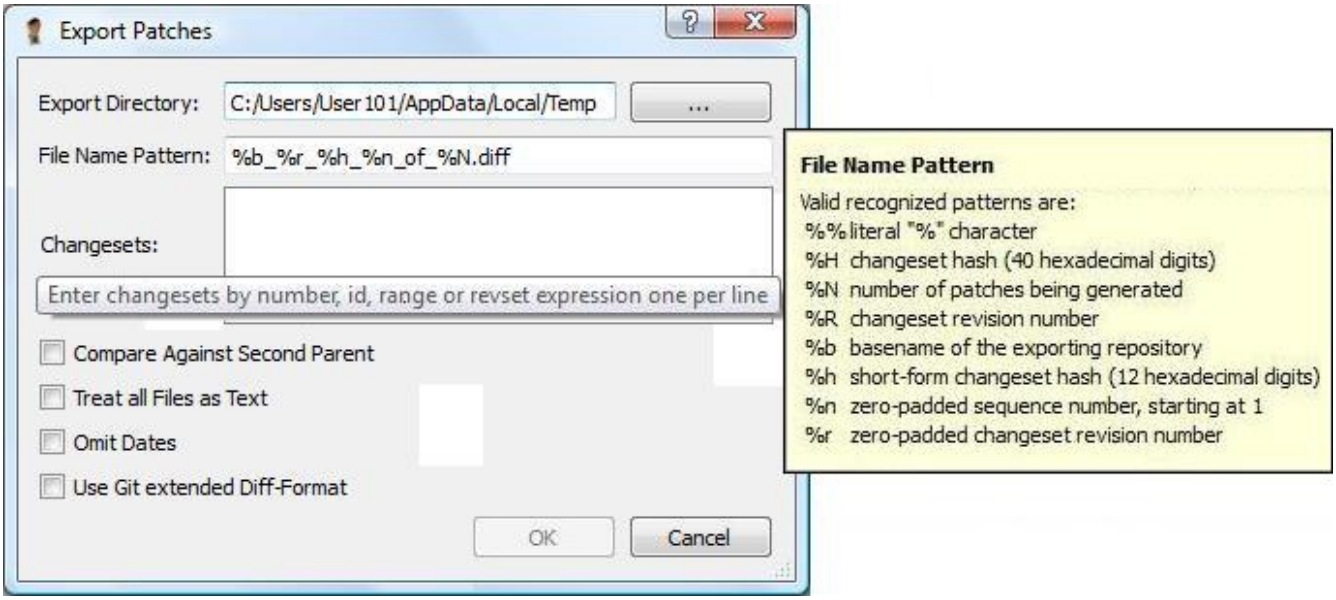
No patch can be imported in case of un-Committed changes on the current Working Revision, unless forced [*see above:* `Enforce Import`].

– –

>> Patch Management > **Export Patches...**

Designed to generate so called "`*.patch`" or "`*.diff`" files out of the current Project, well suited to be imported with the symmetric command `>> Patch Management > Import Patch...` [*see*].

It can be interpreted as an alternative way for interchanging Changesets among Kin versioned Projects, distinct from the usual `Pull` / `Push` commands[34] [*cf.:* Working in Team]. Not so different from the `>> Show Difference (Extended)` command, "`Save`" button [*see*], just more specialized and versatile.



Remark

<!> With this command the incremental nature of the Mercurial Revision or, equivalently, of the Changeset, results particularly evident. Indeed here you see that a given Changeset is nothing else than the changes added to a Project, that is the difference between subsequent Commits.

– –

34 No idea if and how it could really work in case the difference involves also the addition of any new files to a Project. Initial tests have lead to confusing results. [<.?.> *questionable point, further investigation required*]

Control Box

`Export Directory`

Where to store the resulting file
(defaulted to the current user's "`\AppData\Local\Temp`" directory[35])

`File Name Pattern`

Pattern for the name of the Patch file to be generated (conveniently defaulted)

`Changesets` Usual Revision sets selection, as with command `>> Show Log` [*see*].
<~> This same field elsewhere is called "`Revisions`". A better uniformity would be preferable.

`Compare Against Second Parent`
`Treat all Files as Text`

[<。?。> *questionable point, further investigation required*]

`Omit Dates`
`Use Git extended Diff-Format`

Command Options.

Hg Execution

Entering a `Changeset:  rev(2):tip`

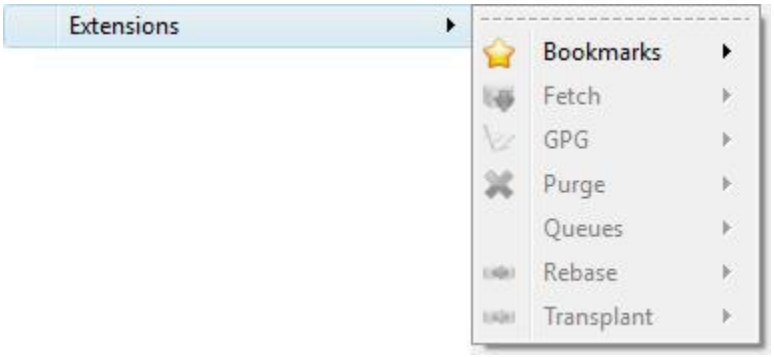
```
export --output C:\Users\User101\AppData\Local\Temp\%b_%r_%h_%n_of_%N.diff
      --verbose rev(2):tip
exporting patches:
C:\Users\User101\AppData\Local\Temp\WDir11_2_73bd8ce4213d_1_of_3.diff
C:\Users\User101\AppData\Local\Temp\WDir11_3_17c1a86c174e_2_of_3.diff
C:\Users\User101\AppData\Local\Temp\WDir11_4_c0052920e8a1_3_of_3.diff
```

- -

35 By the way, note that such directory could be well and safely emptied, after having been used.

>> Extensions

Designed to run extensions, or plug-ins, that can be possibly added to the core Mercurial package.



Of the most popular extensions as here listed there is only one, the Bookmarks, that doesn't require another explicit installation action, and results anyhow here available and enabled because, since version 1.8 (d4ab9486e514), it is part of Mercurial as a core command.

<~> As a consequence, being this a rule for this kind of Tech. Report collection, Bookmarks is the only extension here described, being the only one available after a standard Eric and Mercurial installation[36].

Command List

`Bookmarks`
`Fetch` `GPG` `Purge` `Queues` `Rebase` `Transplant` [*N/A, special installation required*]

- -

36 <~> We could anyhow reconsider this rule in case differently solicited by users encouraging the treatment of some other extension, provided they'd offer their support to that purpose.

>> Extensions > **Bookmarks**

Designed to manage the so called Bookmarks, which are Development pointers referring to given Changesets of a Repository.

Mercurial Bookmarks can play two rather different roles, according to their status, whether "*Active*" ["`current`", *in* E-Hg *terms*] or not. That is:

- All but possibly one Boookmark created in a Repository are Inactive. And their function is virtually the same of a Tag [*see:* `>> Tag in Repository...`], that is plain Changeset pointers to be used with any such command as `>> Switch...` [*see*].
- When created, a Bookmarks can be born either Inactive or Active. That depends upon the Revision selection option chosen on the "`Define`" command control box [*see hereafter*]:
  ➢ If "`TIP`", you'll get an Active Bookmark in correspondence of the currently Active Changeset, independently if a real Tip, or a Head, or not [<~> *rather disconcerting, but true*].
  ➢ In all other cases you'll get a simple Inactive Bookmark.

  Same behavior also with "`Move`" command on an existing Bookmark, possibly turned Active this same way [*see hereafter*].

– –

- <!> Anyhow note that any Bookmark can be turned Active, in any moment, simply switching to it by means of command `>> Switch...`, executed selecting precisely the corresponding "`Bookmark`" option on that control box [*see*].
- Likewise any Bookmark can be also turned Inactive, in any moment, simply switching to its corresponding Revision by means of command `>> Switch...`, executed selecting on its control box an option different from "`Bookmark`"; say: `Number`, or `Id` [*see*].
- In any case, a look at the "`List...`" [*see hereafter*], and you'll verify whether you've got an Active Bookmark, or not.
- As said, at most one Bookmark in a Repository can be set Active and, as such, playing a role comparable to that of a Branch, more precisely a lightweight and much less engaging alternative to Branches [*see:* `>> Create Branch...`]. Main features:
  ➢ As long as a Bookmark is Active it will keep pointing at its Branch's Head, automatically moving to the new Revision after each Commit[37]. Inactive Bookmarks do not move.
  ➢ Active Bookmark can be referred to for transferring the associated Branch to Kin Repositories. Also Inactive Bookmarks can be used for transferring Revisions, but only up to the currently pointed one, that is not necessarily the whole related Branch.
- In short: Active Bookmark points to a Branch (Head), Inactive Bookmarks point to Changesets.

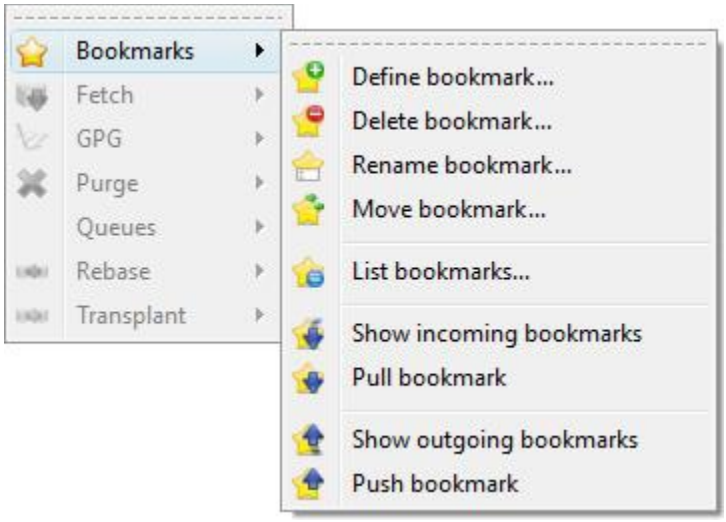
– –

37 Always on top of a Branch, like a cork on water.

Remark

Note that, even though Bookmarks constitutes a local reference[38], their namespace is unique and global, that's why they can be possibly used as reference to `Pull` from / `Push` to Kin Repositories [*see commands:* `>> Bookmarks > Pull / Push`].

– –



Command List[39]

```
Define...          Delete...          Rename...          Move...       List...
Show Incoming      Pull               Show Outgoing      Push
```

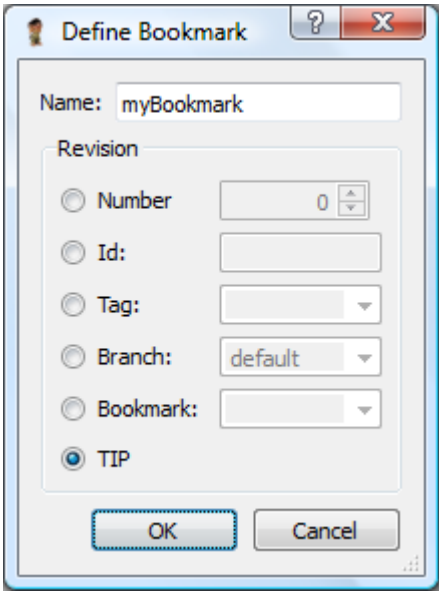
– –

>> Extensions > Bookmarks > **Define**

Designed to create a Bookmark in correspondence with any given Changeset.

38 Just jotted down in a `.hg\bookmarks` file apart, into the Repository.

39 <~> With the trailing "`Bookmark`" term here omitted, for manifest redundancy.



Control Box

`Name` Bookmark identifier

`Number` ... `TIP` Usual Revision selectors, as with command `>> Switch...` [*see*].
<~> Knowing that "`TIP`" might not point to what you'd expect [*see: initial descriptive points here above; and:* Dealing with Mercurial Tip, Heads and Active Revision*, in* Terms and Concepts].

Hg Execution

`bookmarks --rev rev(16) myBMark16`
Selecting an explicit Rev. No, the resulting Bookmark Status is: Inactive

`bookmarks myBookmark`
Selecting default "`TIP`" condition, possibly pointing to an Active Revision with the same Rev. No of the above case, the resulting Bookmark Status is: Active (="`current`").

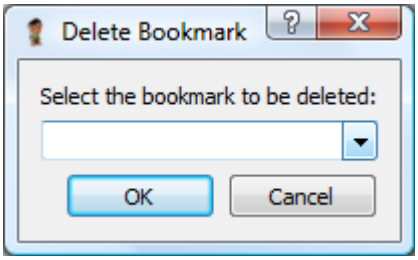
Remark

<~> Be aware that the different initial Status when creating a Bookmark (Inactive vs. Active) is simply the consequence of a different mode for the Revision selection (explicit Rev. No vs. default `TIP`).

– –

>> Extensions > Bookmarks > **Delete**

Bookmarks can be deleted, and with no track left of their existence on the Repository. A feature to confirm how less engaging they are with respect to Branches which, instead, are permanent and global [*see:* `>> Create Branch...`].



Control Box

`bookmark` Name of the Bookmark to delete, with drop-down list

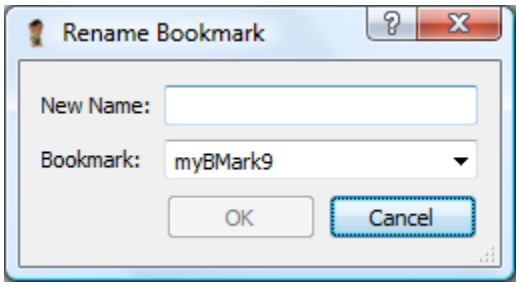
Hg Execution

```
bookmarks --delete myBookmark
```

– –

>> Extensions > Bookmarks > **Rename**

Bookmarks can be renamed. Another convenient element of flexibility.

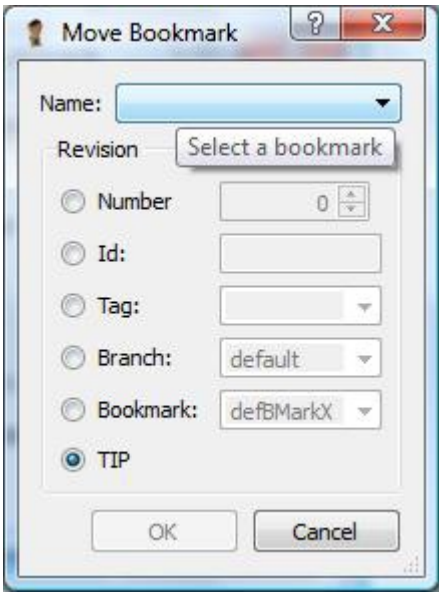


– –

>> Extensions > Bookmarks > **Move...**

Bookmarks can be moved to different Revisions.

<!> If the target Revision is here selected with option "`TIP`", the moved Bookmark is turned Active, as with command "`Define`" [*see above*].



Control Box

`Name` Drop-down list of available Bookmarks

`Number … TIP` Usual Revision selectors, as with command `>> Switch...` [*see*]
<~> Knowing that "`TIP`" might not point to what you'd expect [*see: initial descriptive points here above; and:* Dealing with Mercurial Tip, Heads and Active Revision*, in* Terms and Concepts].

Hg Execution

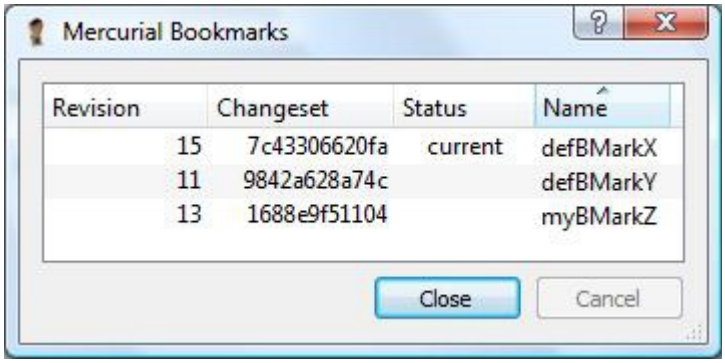
```
bookmarks --force --rev rev(15) myBMark
```

– –

>> Extensions > Bookmarks > **List...**

Bookmarks can be listed, so to check how many there are of them, where they currently are located and, most important, which is the currently Active one; if any. Which is particularly relevant, for two reasons:

- Not more than one Bookmark at a time can be active—or "`current`", as E-Hg flags it.
- It is only the Active Bookmark that, after each Commit, will be automatically positioned on top of the current Branch.



Control Box

`Revision`, `Changeset`
: Identifying codes (local `No` / global `Hex`)

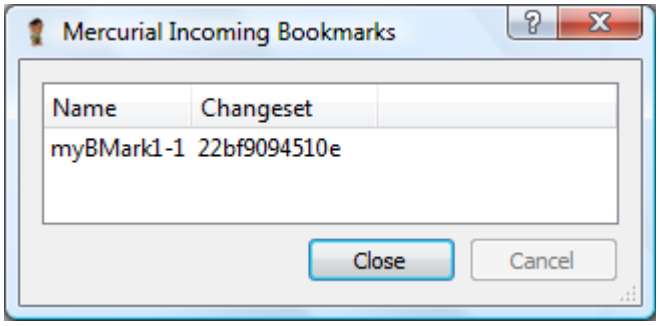
`Status`
: Bookmark status: `blank`(=Inactive) / `current`(=Active, one at most)

`Name`
: Bookmark identifier

– –

>> Extensions > Bookmarks > **Show Incoming**

Designed to show the list of Bookmarks available to be here Pulled from the current Kin Repository configured as source [*see:* Configuration Files, `[paths] default` *parameter, in* Appendix; *and also:* `>> Bookmarks > Pull`].



Control Box

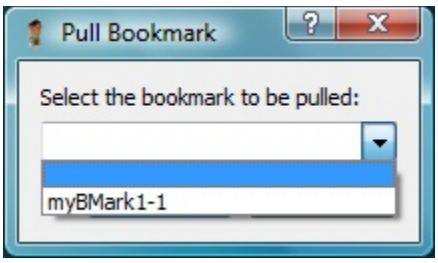
`Name` Bookmark name

`Changeset` Global `Hex Id` code. Note the absence of a Revision `No` which, being a local reference, cannot find correspondence between different Kin Repositories.

– –

>> Extensions > Bookmarks > **Pull**

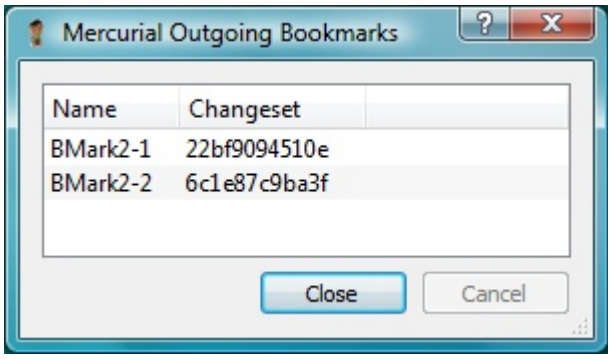
Designed to Pull locally, from the current source Kin Repository, a Bookmark available on the Incoming list [*see:* Configuration Files, `[paths] default` *parameter, in* Appendix*; and also:* `>> Bookmarks > Show Incoming`].



– –

>> Extensions > Bookmarks > **Show Outgoing**

Designed to show the list of local Bookmarks still available to be Pushed to the current Kin Repository configured as destination [*see:* Configuration Files, `[paths] default` *and* `default-push` *parameter, in* Appendix*; and also:* `>> Bookmarks > Push`].



Control Box

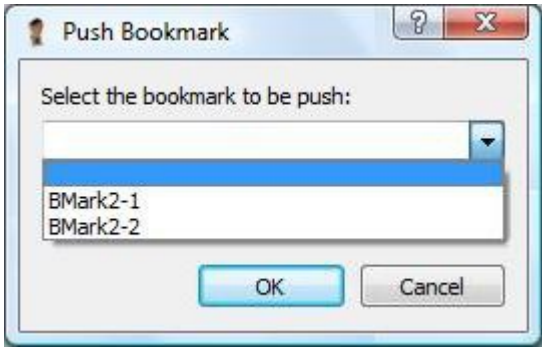
`Name` Bookmark name

`Changeset` Global `Hex Id` code. Note the absence of a Revision `No` which, being a local reference, cannot find correspondence between different Kin Repositories.

– –

>> Extensions > Bookmarks > **Push**

Designed to Push a selected local Bookmark available on the Outgoing list [*cf.:* `>> Bookmarks > Show Outgoing`] to the current destination Kin Repository[40] [*see:* Configuration Files, `[paths] default` *and* `default-push` *parameter, in* Appendix].



40 File write permission allowing, here not considered as out of the E-Hg control.

Control Box

`Select` Selection, from drop-down list of available local Bookmarks [*cf.:* `>> Bookmarks > Show Outgoing`].

– –

>> Extensions > **Fetch**

>> Extensions > **GPG**

>> Extensions > **Purge**

>> Extensions > **Queues**

>> Extensions > **Rebase**

>> Extensions > **Transplant**

Extensions not enabled in case of a standard E-Hg setup and, as such, off this Report's scope.

– –

>> Tag in Repository...

Designed to create—or delete—a "Tagged" Head Changeset, typically to define a Project Revision with an associated a name, to be then conveniently referred to for a possible release of the Project as a public product, out of the factory [*cf.:* ***Remark*** *to term "Revision", in* Terms and Concepts]. This a typical Revision Tag name: `v1.0`

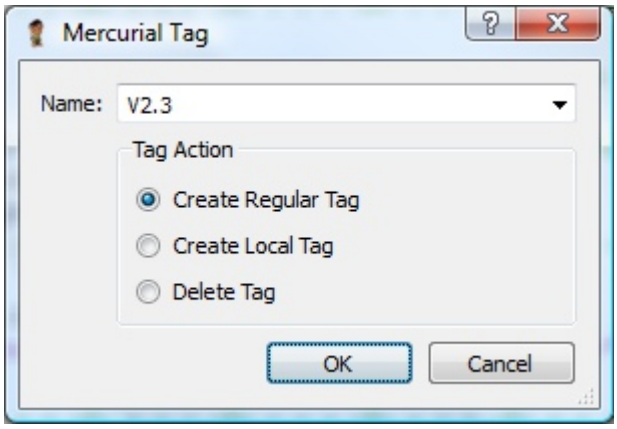
More Tags can be successively defined in a Project, and with a *local* or *global* scope. That is: Working Directory *local*, as is the Revision No, or Repository *global*, as is the hexadecimal Changeset Id.

In case of a regular Repository global Tag, this action will also imply the creation a new Changeset with associated the standard "`tip`" Tag for the new Head Changeset. The deletion of a global Tag [*see:* ***Control Box*** *hereafter*] will also imply the addition of another Changeset [*see:* `>> List Tags...`, *to inspect which the currently defined Tags*].

Remark

<!> Note that these Tags are meant to be assigned only to the Head Changeset of a Branch. So that, in case that's not the condition currently enabled, you should first switch to the Head of the desired Branch [*see:* `>> Switch...`].

– –



Control Box

`Name` Tag name, with drop-down history list

`Regular / Local Tag`
Scope selection: global or local to the current Working Directory only

`Delete` To delete the named Tag

– –

Hg Execution

```
tag --message Created tag <v1.0>. V1.0
tag --local --message Created tag <LocalVx.y>. LocalVx.y
tag --remove --message Removed tag <v0.0>. V0.0
```

Case Log

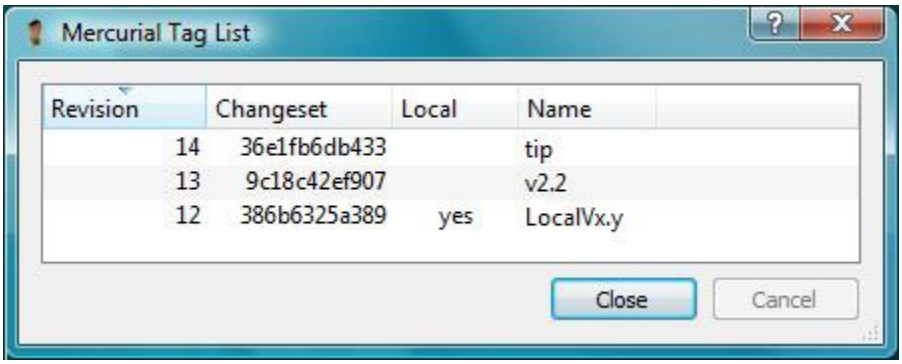
`abort: not at a branch head (use -f to force)`
Tag association to a not Branch-head is discouraged.

`abort: tag 'myLocalTag' is not a global tag`
<~> This is trying to delete a local Tag. Apparently here only global Tags can be deleted. As a work around you may enter a direct Mercurial command:
`tag --remove --local <Tag name>`
[*via:* `>> Execute Command...`].

– –

>> List Tags...

Designed to list all "Tags" currently defined [*see:* `>> Tag in Repository...`]. Note in this list the presence of "`tip`", the automatic Tag conventionally reserved for the most recent Changeset on the Repository.



Control Box

`Revision`, `Changeset`
: Identifying codes (local `No` / global `Hex`)

`Local`
: Tag scope (`yes` = local, `blank` = global)

`Name`
: Tag name, with "`tip`" defined automatically

– –

>> Create Branch...

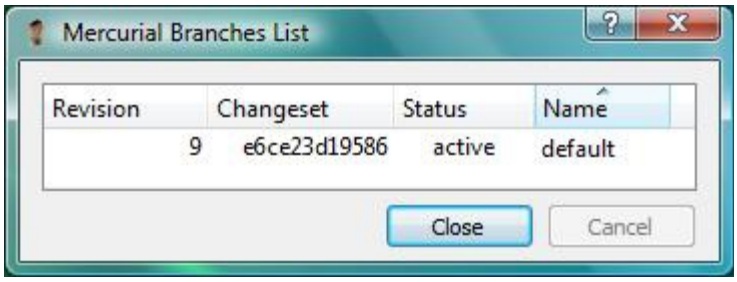
Designed to assign a new name to the currently Active Branch, this way giving origin to a so called new "*Named Branch*"[41] on the Development graph, stemming out of the current Active Revision. Such new Named Branch will actually come into existence at the next Commit of the Working Revision. This action is commonly called "*Fork*".

After a Fork action, the Project Development can proceed independently on each Branch, or be then also re-combined into a unique one, at will, with command `>> Merge Changes...`

41 <~> About the creation of the other so called "Topological Branch", that is a new Branch with unchanged name, see comments to command: `>> Switch...`

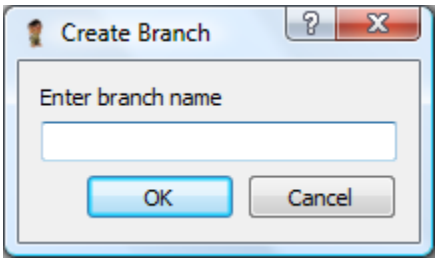
Remark

Note that also the main trunk of the Project is a Named Branch, created automatically from start and named "`default`", as can be verified with command `>> List Branches...`, showing such a result.



That's relevant when using such commands as `>> Switch...`, or `>> Create Changegroup...`, for which "`Branch`" is always a valid option, even if no custom Named Branch has been created.

– –



Control Box

`Branch name` Name to be associated to the new Branch

Hg Execution

```
branch <name>
marked working directory as branch <name>
(branches are permanent and global, did you want a bookmark?)
```

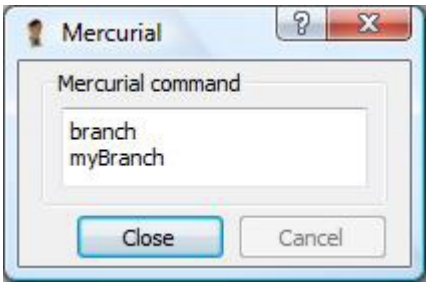
Viewpoint

Note that, as the above message reminds you[42], a Named Branch is a permanent and global structure so that, when creating it, you're advised to consider whether a less engaging structure, such as

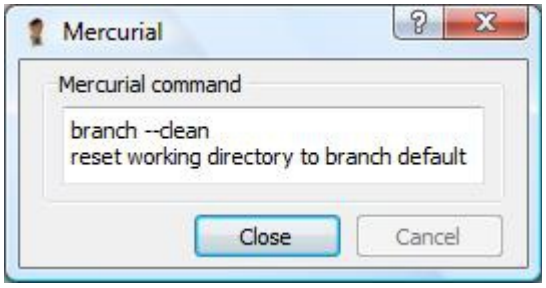
42 <~> Ok: it is a Mercurial message, not E-Hg; but, of course, anyhow worth considering.

a *Bookmark*, could be a better choice [*cf.:* `>> Extensions > Bookmarks`].

Problem is that, at this point in history, the new Name is already there, just waiting to inexorably give origin to a new Named Branch at the next Commit, and with no known E-Hg way to change your mind. Therefore here, if you do change your mid, you have got to apply to such original Mercurial commands [*possibly entered via:* `>> Execute Command...`] as:



`branch` to show the current Working Branch Name



`branch --clean` to reset the Parent's Branch Name

<~> Perhaps two commands worth to be added somehow to the very E-Hg set.

– –

>> Push New Branch

Designed to "Push" (i.e.: upload) a local Named Branch to a target Kin Repository, where it is not yet present[43]. A function that is not executable with the usual `Push Changes` command which, by default, is not allowed to create new Heads on the destination Repository [*cf.:* `>> Push Changes`].

43 Note that "`New`" here is referred to the Receiver Repository, not the Sender.

Target Repository is defined with `[paths]` parameters "`default`" and "`default-push`", as here described with section Configuration Files [*see in:* Appendix].

Hg Execution

```
push -v -new-branch
pushing to C:\_Activity101\Eric\Eric5\E-Hg\_Lab\WDir31
searching for changes
11 changesets found
adding changesets
adding manifests
adding file changes
added 11 changesets with 11 changes to 4 files (+1 heads)
```

Case Log

`abort: repository default-push not found`

Suggested action: check target Repository configuration [*see:* Configuration Files, *in* Appendix].

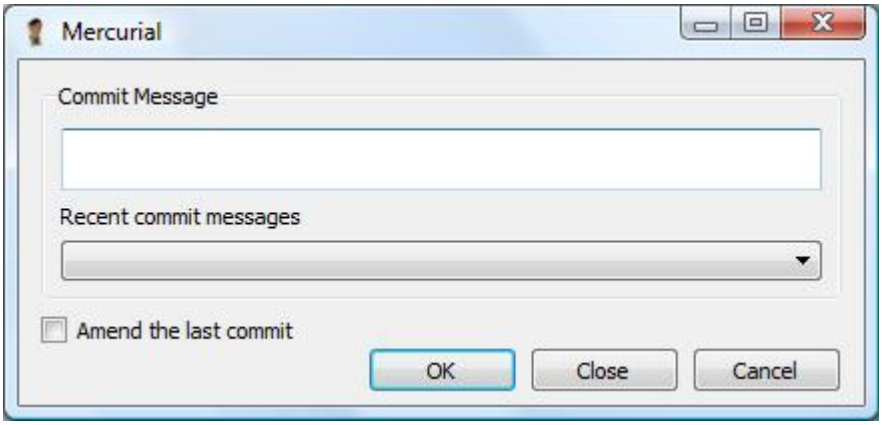
`no changes found`

In case the "new" Name Branch is already present in the target Repository.

– –

>> Close Branch

Designed to Commit a currently Active Head Changeset and also declare its Named Branch as "`closed`", that is: no more needed [*cf.:* `>> List Branches...`].



Control Box

Same as with command: `Commit Changes to Repository...` [*see*].

Hg Execution

```
commit -v --close-branch --message ***
committed changeset 7:e314df182543
```

Case Log

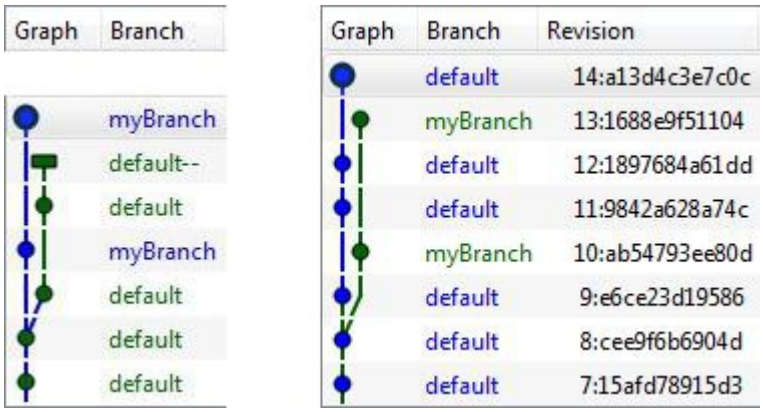
`abort: can only close branch heads`

Suggested action: first switch to a Named Branch's Head.

– –

Remark

Then, to possibly re-open a Branch, simply make its Head Active again [with command: `>> Switch...`], and Commit; as graphically illustrated in the example below.



[*where you see that even "*`default`*" Named Branch can be "*`closed`*"*]

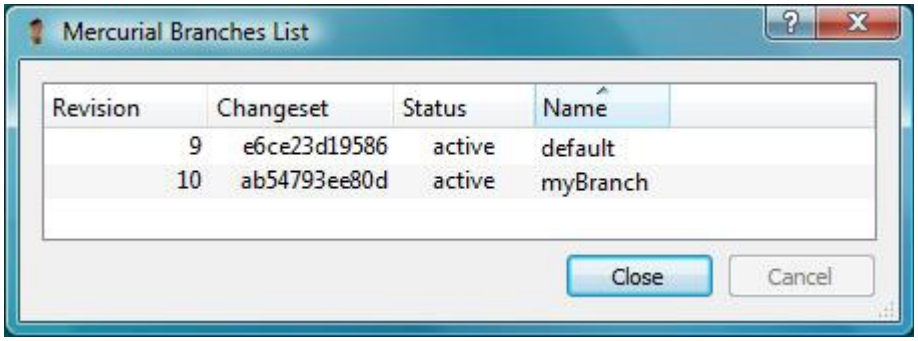
– –

>> List Branches...

Designed to show the list of Named Branches[44] currently defined on the Repository. Note that the sheer fact that a Project is versioned implies necessarily the automatic definition of an initial Named Branch, called "`default`".

44 <~> The distinction between Named and Topological Branches is here particularly significant.

Other Branches can be then defined as distinct “ramification” of the same Project [*cf.:* `>> Create Branch...`].



Control Box

`Revision`, `Changeset`
: Identifying codes (local `No` / global `Hex`) of the most recent Revision on the Branch.

`Status`
: Branch's status:

  `active` when with a proper topological Head, and not `closed` [*see hereafter*];

  `inactive` when without a proper topological Head (that is: merged);

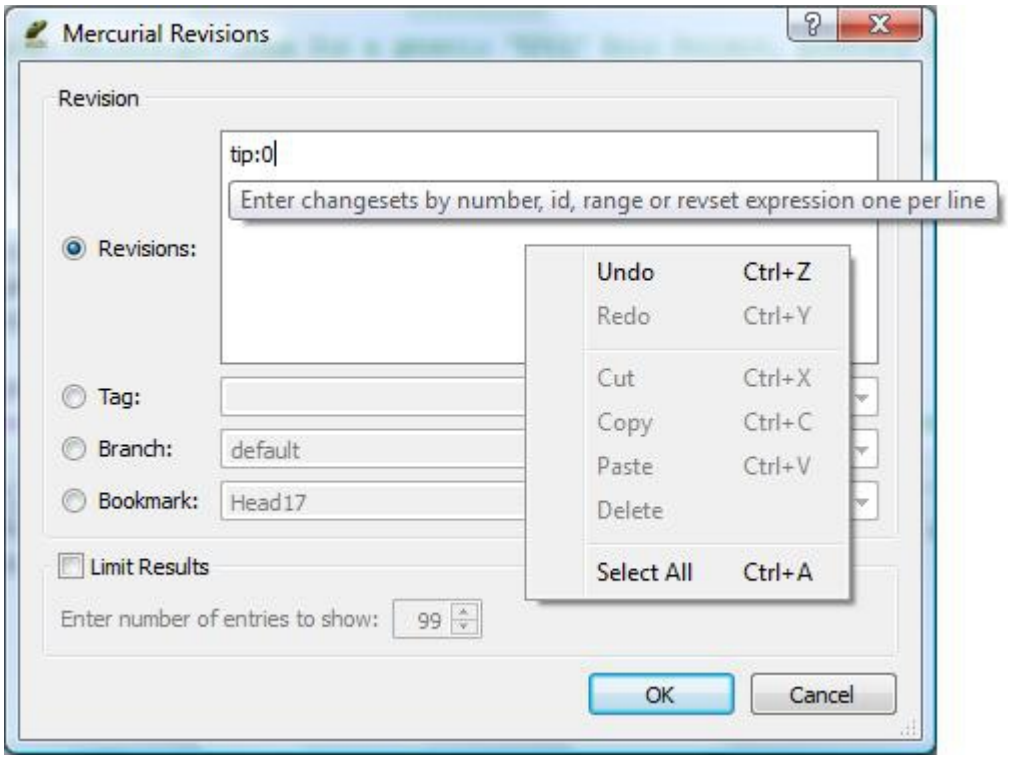
  `closed` when so explicitly declared with command: `>> Close Branch` [*see*], as no more needed (even if with a proper topological Head).

`Name`
: Branch's Name

– –

>> Show Log

Designed to display a section of the Revision Development history according to the values entered on the control box.



Control Box

`Revisions` Revision sets selection, specified with a single Revision `No`, or hex `Id`, one per line; or as intervals according to such Mercurial standard expression[45] as:

`tip:0` all[46];
`m:n` from `No` *m* to *n*[47];
`tip:-n` *n* most recent Revisions.

A right click edit pop-up menu is here of help [*see*].

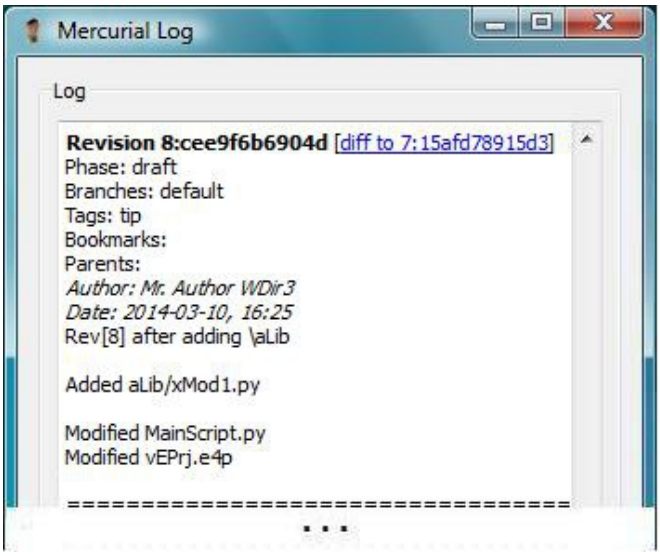
45 See section: Specifying Revision Sets, Mercurial Command Reference [*ref.: at* Mercurial Package, *in* Appendix].
46 <~> Here we'd appreciate such a default, so to get anyhow a visible result, as usual with other E-Hg commands.
47 Possibly specifying a precise `rev(number)` or `id(string)`, if in doubt.

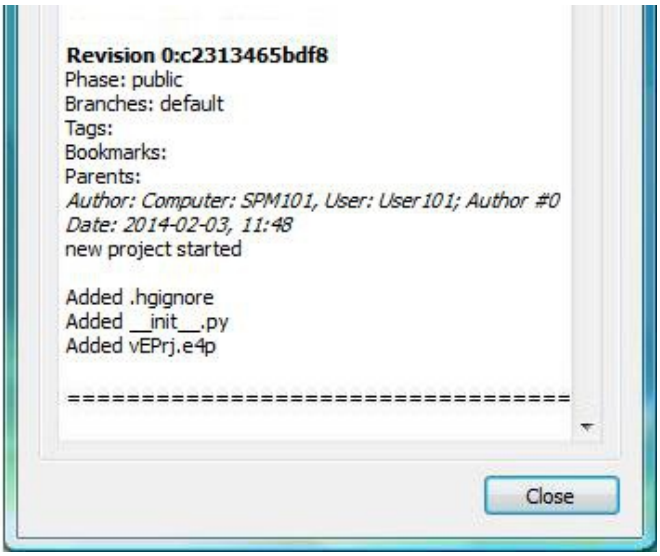
`Tag ... Bookmark`
: Usual Revision selectors, as with command `>> Switch...` [*see, <~> and be informed that "TIP" might not point to what you'd expect; see also:* Dealing with Mercurial Tip, Heads and Active Revision*, in* Terms and Concepts].

`Limit Results`
: Check-box to activate a limit for the number of Changesets to display, set on the control below. With default as set in: "`No of log messages`", with command: `>> Configure...` [*see*].

– –

Case Log





Where:

`Revision` `<No:Id> [diff to <No:Id>]`
: Referred Changeset, with a link to show the "`diff to`" its Parent Changeset [*see:* `>> Show Difference`]. Link necessarily missing in a root (i.e.: Origin) Changeset No 0.

`Phase`
: Changeset's Phase, as with command `>> Change Phase...` [*see*]. Cases: `public`, `draft`, `secret`

`Branches`
: Changeset's Branch name[48] ("`default`", if no custom name assigned)

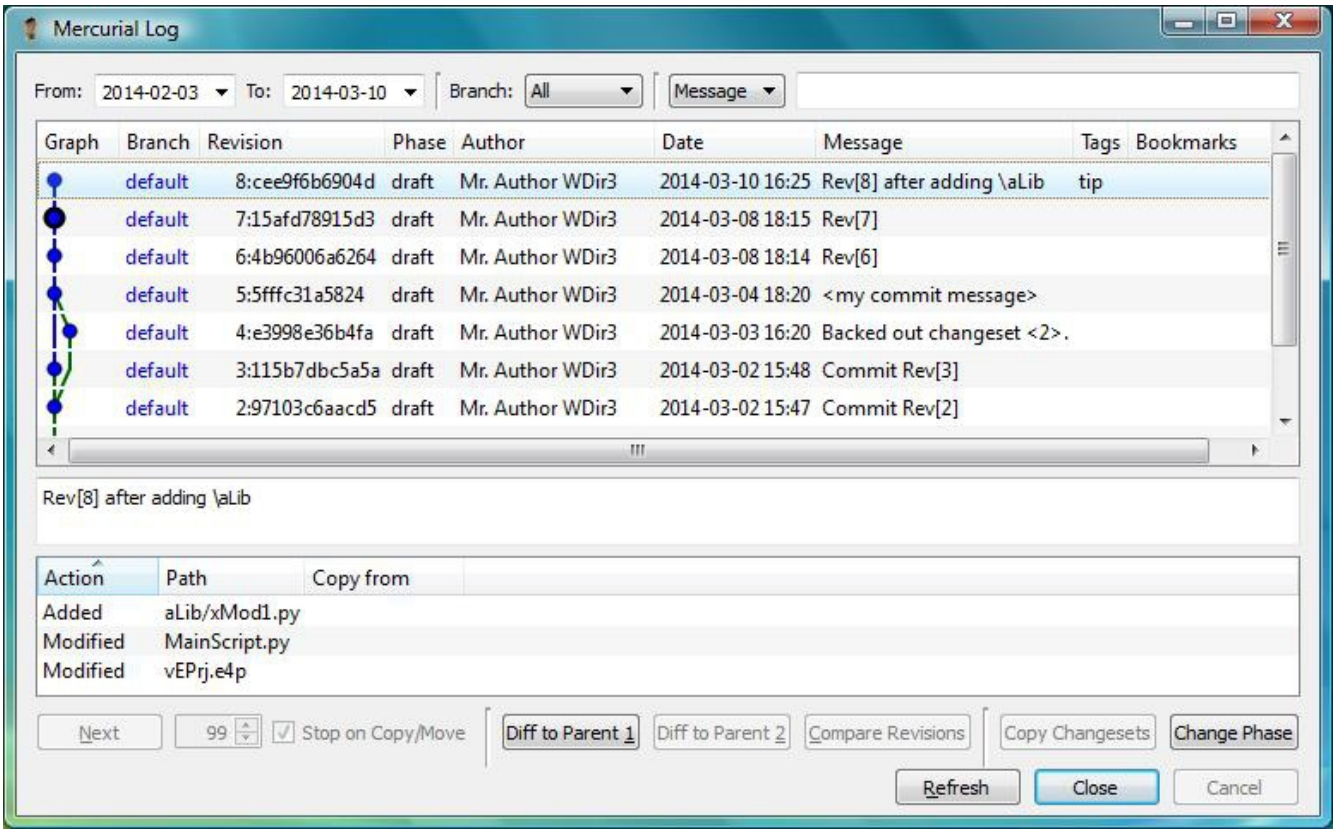
`...`
: [*self-explanatory data*]

– –

48 <~> Why plural? Really a Changeset can belong to more than one Branch at a time?

>> Show Log Browser

Designed to run a tool for browsing the whole E-Hg Development story of the versioned Project. <!> The resulting form can be then kept open while proceeding entering other commands, and refreshed so to grant updated display [*same as with:* `>> Show Status...`].



Remark

In this form look at the presence of a "node" at Rev. 2, followed by two so called "Topological Branches" leading to the Rev. 3 and Rev. 4. Neat example of a Revision graph ramified into distinct branches with the same name (that is, here: "`default`").

Viewpoint

<~> Such distinction between "Topological" and "Named" Branches is functional to Mercurial[49], that's why, in this Report, we'll explicitly make this distinction whenever appropriate.

49 <~> In mathematical terms, all branches in a graph cannot be but topological. The distinction here is to be intended between "purely topological", that is not named, and named branches.

Control Box

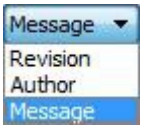
"`Mercurial Log`"

Descriptive header [*cf.:* `>> Show Incoming` / **`Outgoing`** `Log`] of this form, hereafter described into the following sections: ***Managing Controls***, ***Table***, ***Selected Item's***

Managing Controls

`From`, `To` Date interval, drop-dpown controls

`Branch` Name 

and topic 

selection; with a blank text field for: `<all>`

– –

`Next` / `<n>` To control the lines in display [*default No in:* `>> Configure...`]

`Stop on Copy/Move`

<?> An option still obscure to us, renounced. Hints:

- It is an option we've always spotted as checked and disabled; no idea why, nor how.
- We've been told it is meant to "stop the Log" at the point when entered a Context Menu command `(^)> Copy` / `Move` [*see*]. A behavior that, unfortunately, we haven't been able to grasp, nor to verify.
- We presume the E-Hg *CM* command `(^)> Move` corresponds to the Mercurial "`rename`", with "`move`" as just an alias.

`Diff to Parent 1, 2`

To show the "`Mercurial Diff`" between currently selected Revision and one of its (max. two) Parents [*cf.:* `>> Show Difference (Extended)`]. Result may be saved on a "`*.diff`" file [*cf. also:* `>> Patch Management`].
Disabled when no Parent is defined (e.g.: with an Origin Changeset).

`Compare revisions`

To compare two selected revisions (may also be not contiguous, via `Ctrl-Click`). Button otherwise disabled. [*cf.:* `>> Show Difference (Extended)`]

`Copy Changesets`

Enabled only in presence of another Branch from which to possibly copy Changesets over. Same action as with command `>> Graft > Copy Changesets` [*see*].

`Change Phase` To switch from "`draft`" to "`secret`", and back; disable when "`public`" [*see:* `>> Change Phase...`]

– –

Table

`Graph`	Revision history diagram, with graphic symbols characterizing each Changeset:  Currently Active one, typically the Tip  Mid  Origin
`Branch`	Related Branch name
`Revision`	Identifier, format: <local `No`>:<global `Hex`>
`Phase`	Cases: `public`, `draft`, `secret` [*see:* `>> Change Phase...`]
`Author`, `Date`	Commit's
`Message`	Commit's, here just the first 30 char.s [*see next section*]
`Tags`, `Bookmarks`	Possibly associated identifying flags

– –

Selected Item's

`<Text Field>`	Commit Message, full text
`Action`	On referred file: `Added`, `Modified`, `...`
`Path`	File's name and location, relative to the Working Directory
`Copy from`	<?> Same as with the above option "`Stop on Copy/Move`" [*see*], still obscure and not verified. Renounced.

Viewpoint

<~> Strong, very strong perplexities, here reinforced, have we got about this whole ***CM*** `(^)>` `Copy` / `Move` and `(^)>` `Add` / `Remove` matter [*see:* ***Viewpoint*** *after commands* `(^)>` `Copy / Move`, *at section* E-Hg Context Menus].

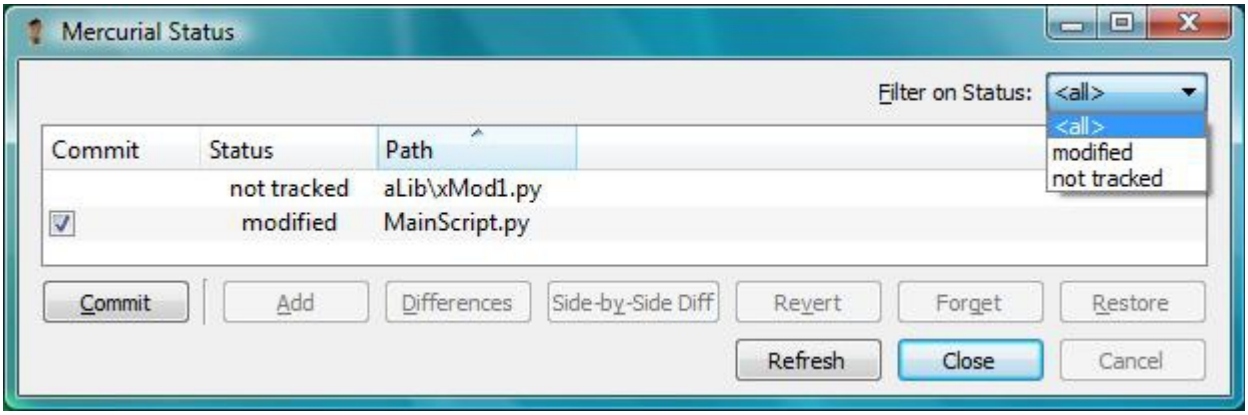
– –

>> Show Status...

Designed to show a control box where to display and manage the "`Mercurial Status`" of the files contained in the Working Directory.

Viewpoint

<~> We consider this command's name really as too generic, virtually meaningless. At least a specification like "File Status" would be here of help, so to have a clue whose "status" is here in display.



An empty form is shown when no changes occurred on the Working Directory.

Control Box

`Filter on Status`
: Drop-down list, to select which the files to display, according to their Mercurial Status: "`modified`", "`not tracked`", or both.

`Commit`
: Check-box for a changed files to possibly Commit. Note that here even single files can be Committed, whereas the command `>> Commit Changes to Repository...` is to Commit all Working Directory changes [*see*].

`Status`
: Mercurial Status of the referred file, can be:
  - `modified` Changed and not Committed
  - `not tracked` To "track" it, see next button "`Add`"
  - `removed` <~> Not in this "`Filter`" drop-down list.

`Path`
: Path of the referred file, relative to the current Working Directory.

\- -

`Commit`
: Execution button, for the selected items.

`Add`
: To add "`not tracked`" item to the Repository [*cf. pop-up* **CM** *(Context Menu) command:* `Project-Viewer(^)> Add to Repository`].
Inverse action with "`Forget`" button [*see hereafter*].

Remark

<!> Note that the "Add" to Repository and the "Add" to Eric Project are totally independent actions, in the sense that an item can be well added to a Repository, and not to the Project, or vice-versa; or added to both. And with such independent commands as:

To Project: *MM* `Project > Add Files / Directory`
CM `Project-Viewer(^)> Add Source Files... / Directory`

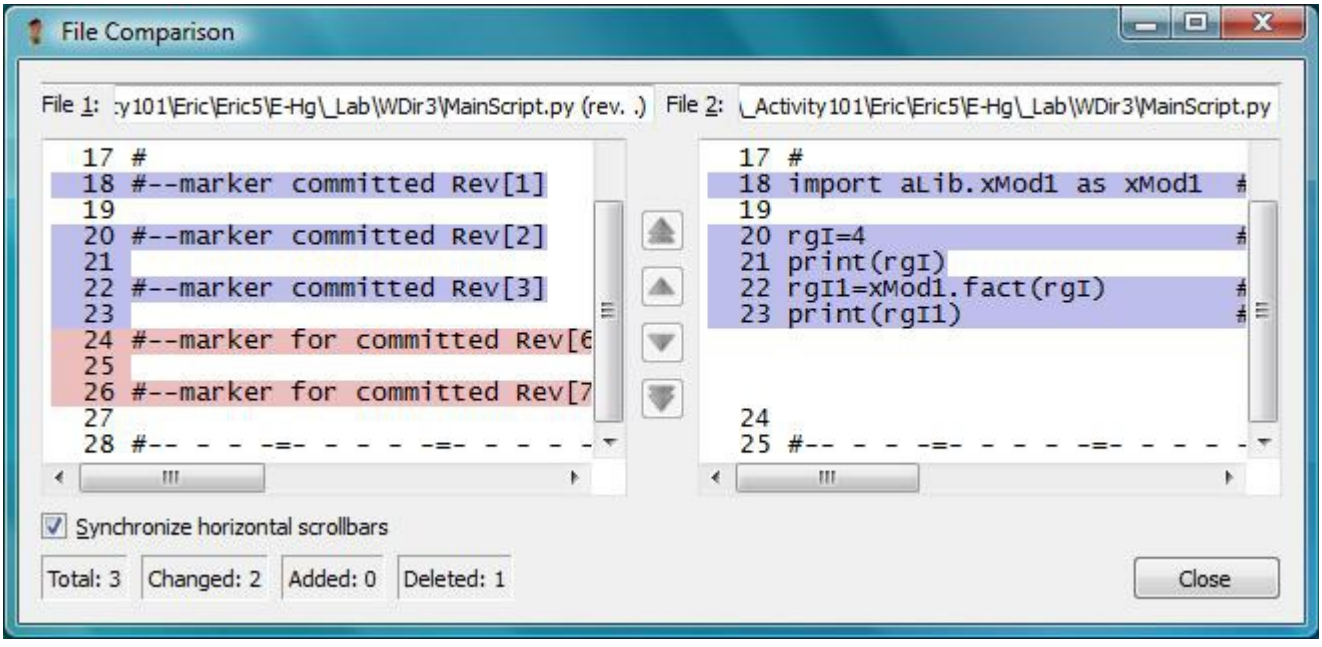
To Repository: *CM* `Project-Viewer(^)> Add to Repository`
This "`Add`" button.

Just as a matter of convenience, the usual sequence is: first add to the Project, if appropriate; then possibly to the Repository.

– –

`Differences` To show the differences between a selected file as now in the Working Revision and as Committed in the Active Revision [*cf.:* >> **`Show Difference`**]
[<。?。> *Is it really so?*]

`Side-by-side Diff`
Same as former "`Differences`" button, but with this different display format:



`Revert` Same as command: `>> Revert Changes` [*see*], for the selected item.

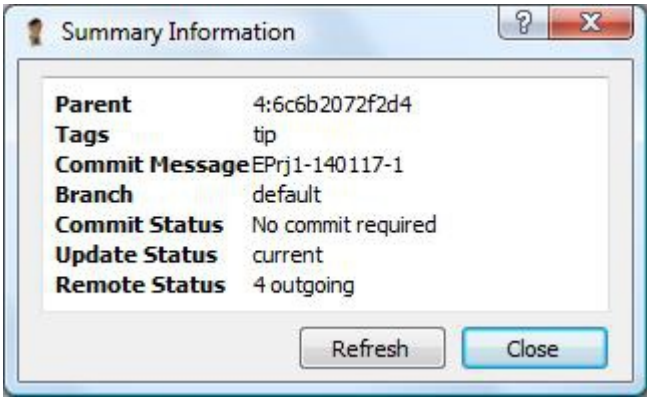
`Forget` Inverse of the "`Add`" action above [*see*]. Selected file will be not tracked any more, after next Commit. Former Commits, and the file in itself, remain unchanged.

`Restore` To restore possibly "Missing Files", that is files deleted out of E-Hg control, either accidentally or by purpose [*cf.:* `Project-Viewer(^)> Remove` *commands*].

– –

>> Show Summary...

Designed to show a descriptive identity card of the current Active Changeset.



Control Box

`Parent` Both identifiers of current Active Changeset, with format: <local `No`>:<global `Hex`>

`Tags` Possible "Tags" [*cf.:* `>> Tag in Repository...`]

`Commit Message`
Initial part of the related Commit message (first 30 char.s)

`Branch` Branch name.
<~> Note that this name is here properly shown also in case it is the "`default`".

`Commit Status` It is to tell if any change has been executed on the Working Revision, so to possibly require a new Commit action. Cases: Not required / Required (e.g.: "`1 modified`").

`Update Status` It is about the synchronization between Repository and Working Revision. Cases:

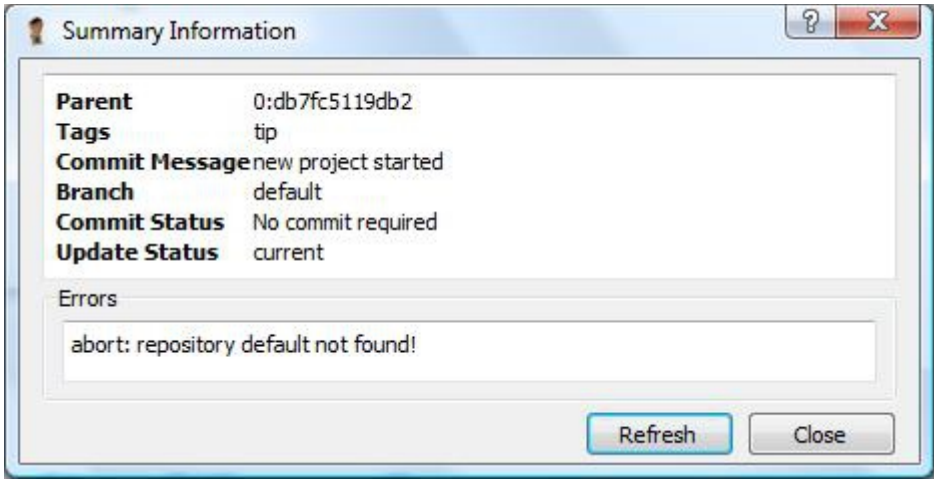
`current`	synced
`update`	Repository is changed, sync action required
`merge`	both Repository and Working revision are changed, sync required

`Remote Status` It is about *Local* vs. *Remote* Repository (as defined with the "`[paths]`" parameters set in `Mercurial.ini` and `.hg\hgrc` configuration files [*see:* Configuration Files, *in* Appendix]; usually defaulted to the Parent's, at "clone" action). Cases:

- `<n> outgoing / incoming` changes and bookmarks that are there to be possibly Pulled / Pushed [*see:* `>> Push / Pull Changes`].

– –

Case Log



`abort: repository default not found!`

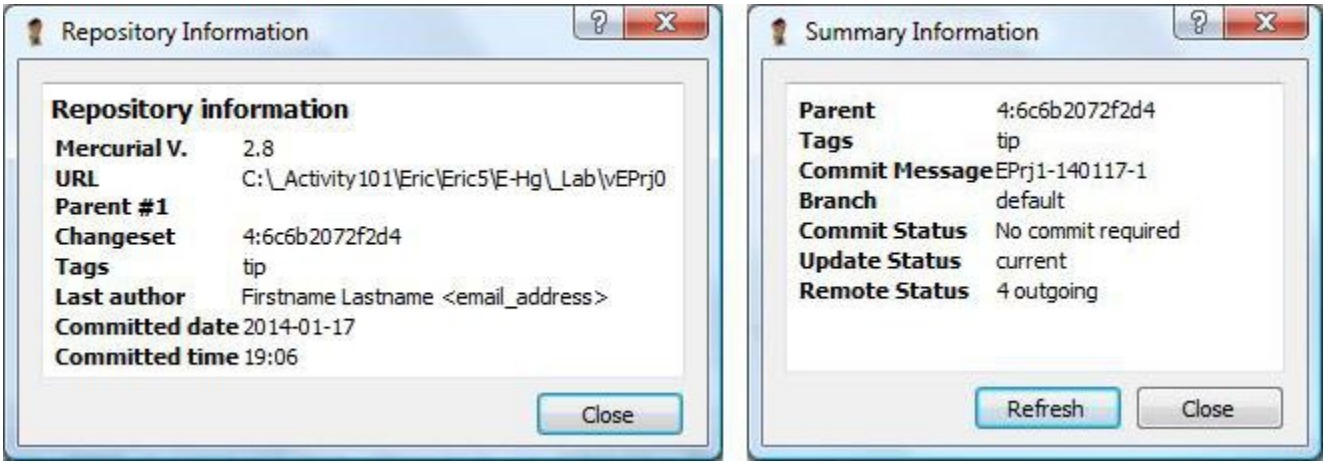
There are cases in which error message can be here shown. Reason, in this case:

- It's a root (i.e.: Origin) Repository, therefore with no Parent as a default "Push" Repository;
- Suggested action: Ignore, or Switch to a "cloned" Child Repository.

– –

Viewpoint

<~> We do not see that much difference between what is shown by this command and by this other one: `>> Mercurial` [*see example hereafter*].



Therefore we frankly wander if these two commands could be conveniently blended into one [*other similar redundancies have we spotted, here and there; perhaps worth considering, after this one*].

– –

>> Show Difference

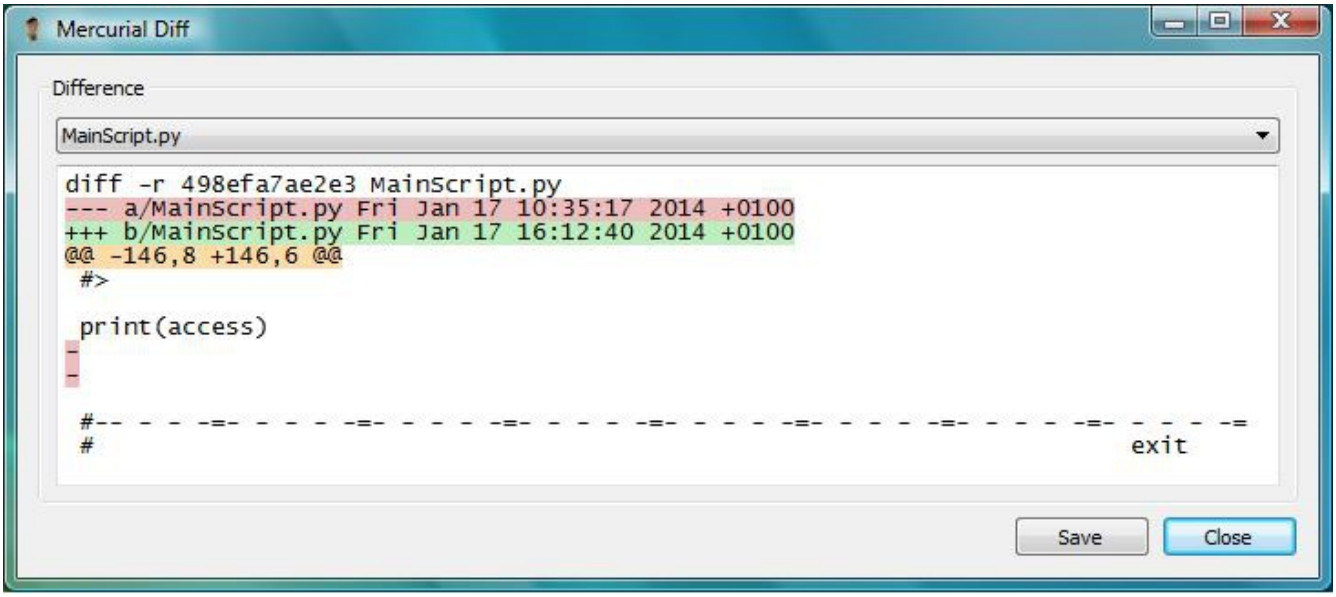
Designed to show the difference[50] between files as in the the current Working Revision and as in its current Active Revision; that is, its Parent Committed Revision. Difference which is exposed in "*unified diff*" (or "*unidiff*") standard XML text file format [*cf. also:* `>> Show Difference (Extended)`].

Remark

- Command `>> Revert Changes` [*see*] is provided to possibly reset the current Working Revision to its original Active Revision state.
- Command `>> Commit Changes to Repository...` [*see*] is to record the current Working Revision into a new Committed Revision, and then possibly have it as Parent of the new Working Revision.

– –

50 <~> As a rule, with *Difference* we intend: (*Final – Initial*) value, in that order.



Control Box

`Difference` Drop-down items selection list, with the Difference intended as:
[(*Working* Rev.) – (*Parent* Rev.)]

<!> Note that Working and Parent Revision necessarily correspond to the *same* identifier [*see the unique "`-r`" global hexadecimal code at first line of the example above*], as it is usual to refer to a Working Revision with its same Parent's identifier.

`Save` To save the result into a standard "`*.diff`" file, possibly usable as a Patch as with command `>> Patch Management` [*see*].

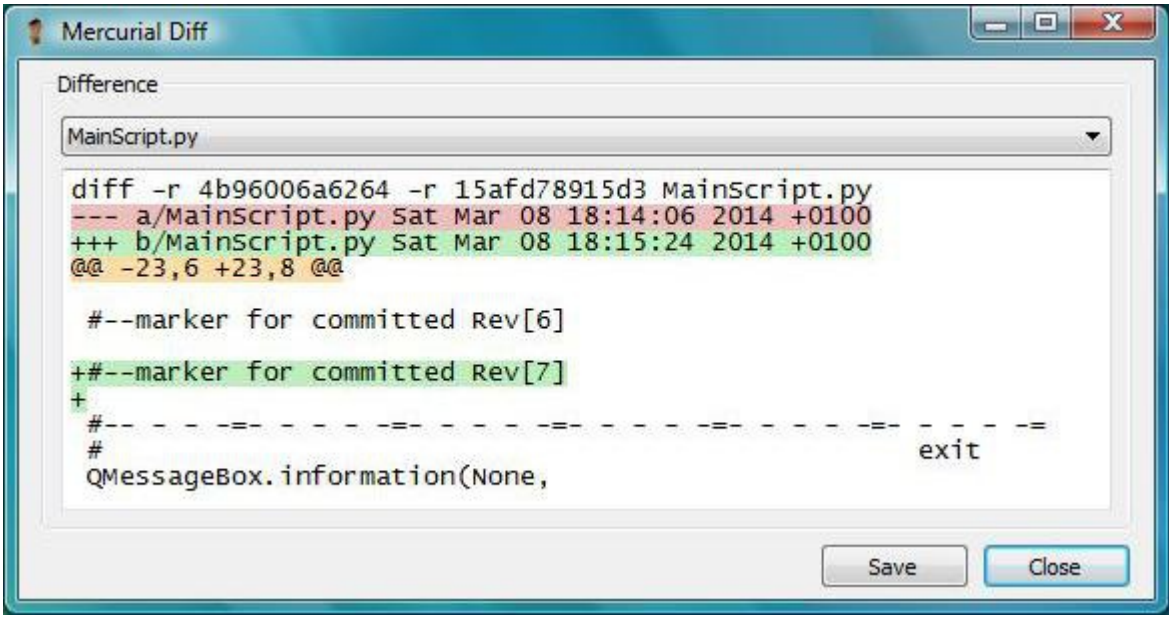
Hg Execution

```
diff -r <Changeset Id> <Source File>
```

– –

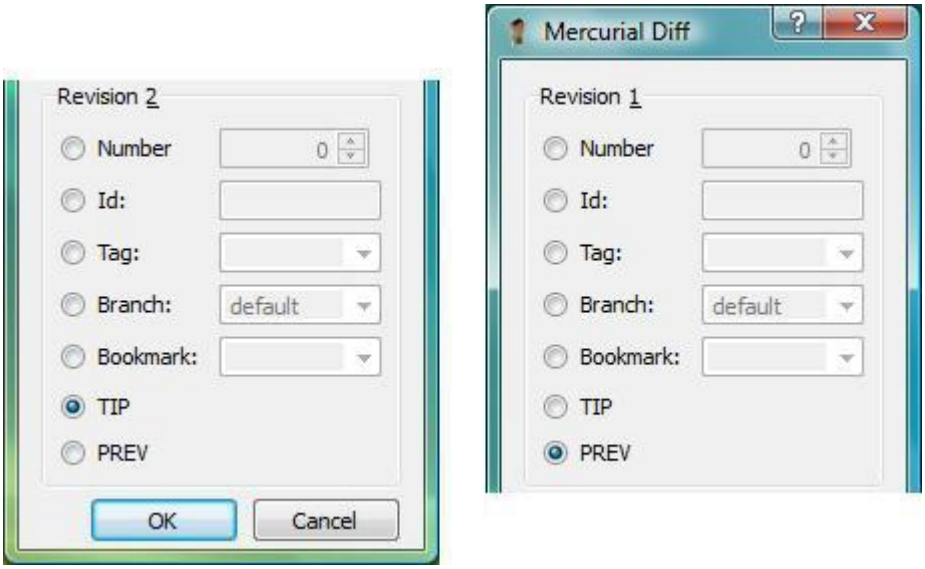
>> Show Difference (Extended)

Designed as a way for inspecting what happened between two distinct Committed Revisions, shown by means of the so called standard "*unified diff*" (or "*unidiff*") XML text file format.



Resulting difference can be then recorded as a standard "`*.diff`" file [*see: "`Save`" button, on the above example*] and also transferred, as a way for dispatching specific Project changes [*see:* `>> Patch Management`, *whose standard "`*.patch`" file format is identical to "`*.diff`"*].

– –



Control Box

`(2nd) Revision` *vs.* `(1st) Revision`

Revisions to compare, with difference intended as: [*end*(`Rev. 2`) – *start*(`Rev. 1`)]
Being as initial default condition: [`TIP - PREV`(=currently Active Revision)]

Remark

This default implies that:

- If you are currently working at the Tip Revision, it necessarily brings about always this result: "`There is no difference`"
- If you are working at any former Revision, what you get is the *inverse* of the changes leading to the Tip.

Whereas if you'd like to compare, say, any given Active Revision with any *former* one, you should first inquiry how to identify that Revision, by means of such other command as: `>> Show Log` [*see*], and then select such an ordered pair: [`PREV` (as `Rev. 2`) – *former*(as `Rev. 1`)].

Viewpoint

<~> A procedure that could be a bit improved if, as we have already proposed, all the Revision selector fields were defaulted with their currently Active values.

– –

`Number ... TIP` Usual Revision selectors, as with command `>> Switch...` [*see, <~> and be informed that "`TIP`" might not point to what you'd expect; see also:* Dealing with Mercurial Tip, Heads and Active Revision*, in* Terms and Concepts].

`PREV` Current Active Revision
<~> Forget this label's apparent meaning, which is *totally misleading*[51].

Remark

Only *Committed* Revisions can here be selected. This means that, with this command, there is no way to compare the contents of a Working Revision, considered as it currently is, to what it was when Committed; not even using the option PREV [<~> *as we frankly expected*]. To that purpose use the other similar command: `>> Show Difference` [*see*].

– –

51 <~> We know the perverse reason for this choice, not even worth to be mentioned. We totally disagree.

>> Change Phase...

Designed to assign to Revisions a "*Phase*" status different from what assigned automatically.

The Mercurial Revision Phase system has the purpose of providing a mild locking mechanism to prevent common data sharing mistakes during critical actions, typically when editing a Working Revision. Usually handled automatically, it can be also controlled manually with this command.

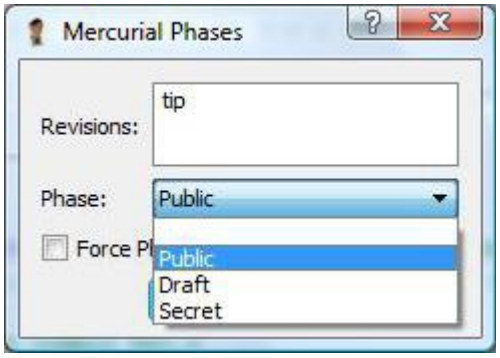
These the rules on which it is based.

- The Phase of a Revision can be: `Public`, `Draft`, `Secret`; in order of decreasing sharing permissiveness.
- By default Working Revisions are set `Draft`, then switched to `Public` when Committed.
- No Child Revision can be more shareable than its Parent, and therefore anyone of its Ancestors.

Current Revision's Phase can be inspected with the `>> Show Log` commands [*see*].

Remark

Note that this is only a Mercurial related mechanism, as the actual data sharing is anyhow subject to the read / write permission rules of the network and file system in use[52].



Control Box

`Revisions` Usual Revision sets selection, as with command `>> Show Log` [*see*].

`Phase` Drop-down list, in order of decreasing permissiveness. Selected value will affect not only each pointed Revision, but also all the Descendant ones in accordance with the related rules [*see above description*].

`Force Phase Change`
To force a normally forbidden change [*see above description*].

52 A subject which is off the scope of this Report.

Hg Execution

```
phase --draft tip
```

Case Log

```
cannot move 1 changesets to a more permissive phase, use --force
no phases changed
```

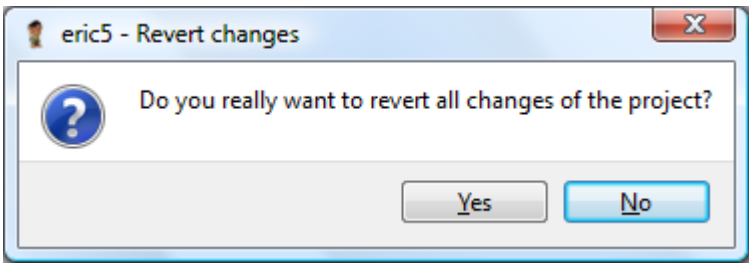
Just a warning, and a suggestion too.

– –

>> Revert Changes

Designed to revert the current Working Revision to the Parent's original Committed status, that is dismissing all possible changes so far executed.

Before proceeding you'll be asked to confirm, as there is no redo for this action [*cf.: section* Undo and Redo of Commit Changes, *after command:* `>> Commit Changes to Repository...`].



Command `>> Show Difference` [*see*] is provided to show the difference between the current Working Revision and its original Parent Revision state.

Remark

<!> Once any changes have been introduced on the current Working Revision, it is important to be aware that:

- Moving to a Revision different from the current one, with such command as: `>> Switch...` or `>> Update from Repository` [*see*], necessarily implies a "`merge`" process of the changes [*see:* `>> Merge Changes...`], with consequent possible conflicts. Which could not be exactly what you had in mind.
- Two actions for avoiding what at the previous point:
  - eider first Commit current Working Revision [*cf.:* `>> Commit Changes to`

`Repository...`], and then move to the other desired one;

- ➢ or simply drop all changes with this very `>> Revert Changes` command.

- Note that in case of a dummy move, that is a `>> Update from Repository` or a `>> Switch...` aimed at the same current Parent revision, nothing will happen, in particular the current Working Revision will not be reset to the original Revision contents. The only way to reset contents is to execute this very `>> Revert Changes` command.

– –

Hg Execution

```
revert --no-backup -v <file>
```

– –

>> Merge Changes...

Designed to merge into the current Working Revision the changes executed into another Revision, so to, typically, reduce two Branches, either Named or Topological, into one. That is, more precisely, combine two Heads into one, the current Working Revision. This way the consequent Committed Revision will be a Descendant of two Parent Revisions.

In this sense, it is the inverse of a forking action as it can be executed:

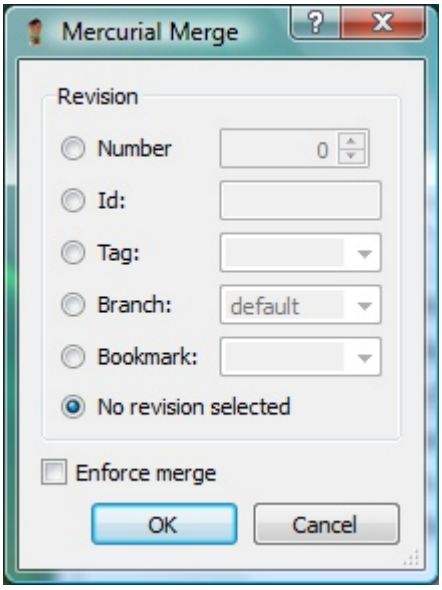
- on a Working Changeset with a `>> Switch...` to an Ancestor Revision, for the Topological;
- or with a `>> Create Branch...`, for the Named Branches [*see*].

– –

Remark

<!> When merging files edited independently, it is commonplace to fall into source text conflicts requiring human intervention. Composition of conflicts can be assisted by dedicated tools, here not considered[53]. In this Report we'll simply consider the operative condition as set with the global `[ui]` option: `merge = internal:merge` [*see:* Configuration Files*, in* Appendix].

53 <!> This kind of merge tools—such as the popular diff and merge program "KDiff3"—constitute a subject in its own right and a rather relevant too, but off scope for this Report as not originally included in E-Hg, nor required as a prerequisite. Anyhow, if present, a merge tool would be automatically run by Mercurial when needed. Whereas, if missing, and with no such instruction as the here adopted `[ui]` option: `merge = internal:merge`, such a message can be shown: "`couldn't find merge tool ***`".



Control Box

`Number ... Bookmark`
Usual Revision selectors, as with command `>> Switch...` [*see*].

`No revision selected`
With no Revision specified, the merge is performed between the current Working Revision and the Head of another Branch, provided there is *only one* other Branch. Otherwise, an explicit Revision with which to merge with must be entered.
This is the default condition.

`Enforce merge` To force anyhow the stated merging action, without any further operator involvement. Option declared "DEPRECATED" by the Mercurial documentation [*cf.:* Mercurial Command Reference, *at* Mercurial Package, *in* Appendix].

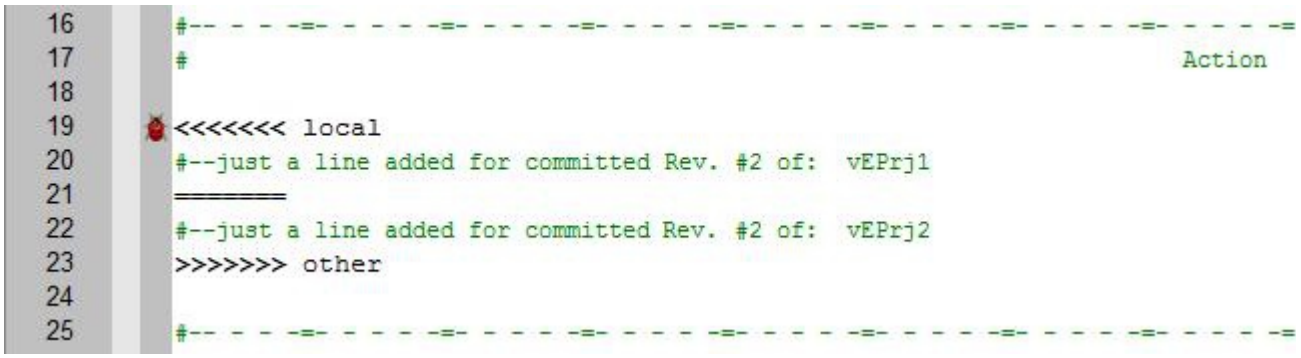
– –

Hg Execution

```
merge
```

Case Log

`(branch merge, don't forget to commit)`
It's just a reminder.



`warning: conflicts during merge.`
`merging MainScript.py incomplete! (edit conflicts, then use 'hg resolve -mark')`

A typical merging conflict case, signaled with the special *conflict markers* generated by the basic Mercurial merge tool; to be solved as this "`warning`" message suggests [*then see also:* `>> Conflicts Resolved`].

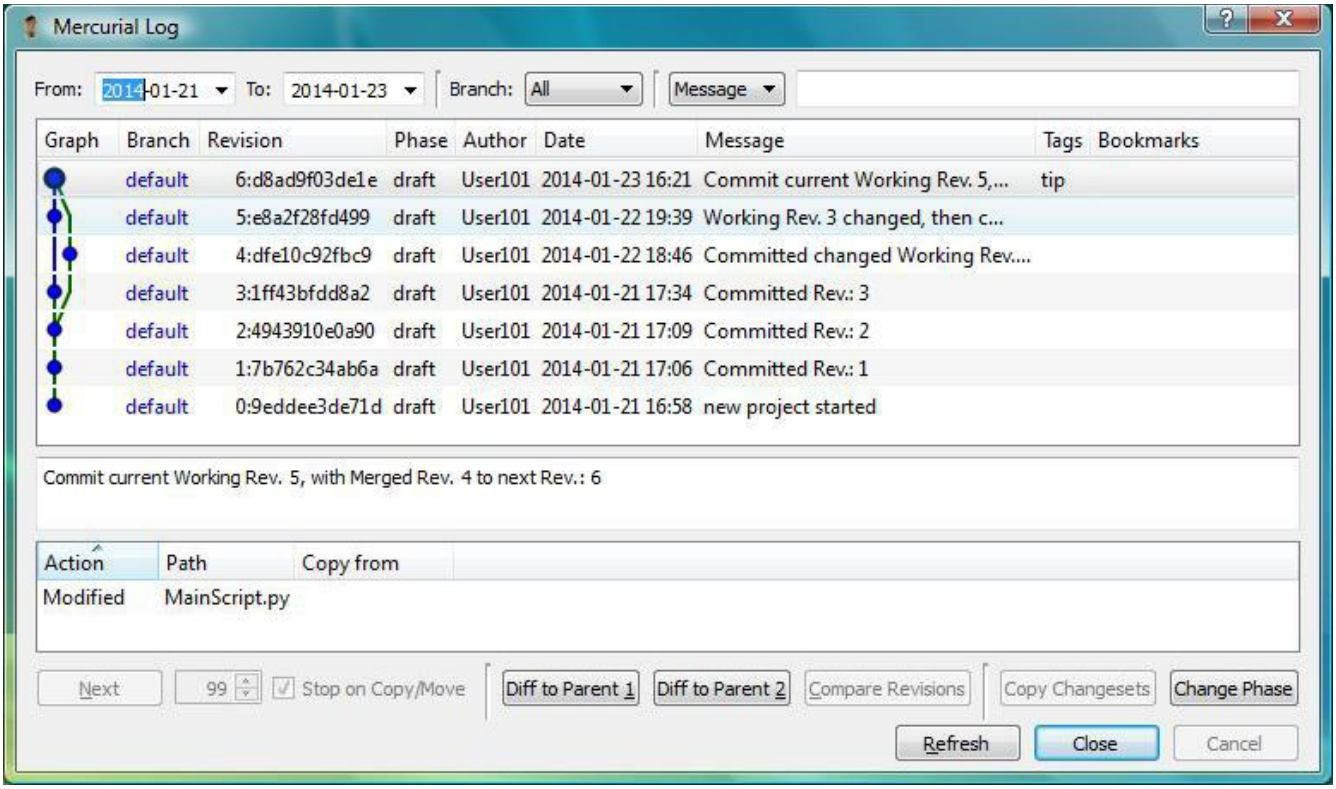
`abort: outstanding uncommitted merges`

Just a reminder, in that case.

– –

A Merging Example

With this `>> Merge Changes...` command you have that two Topological Branches of a Revision Development Graph can be combined into one, as in this example,



where you see a "`tip`" Revision No 6 as the result of the confluence of two Parent Revisions 4 and 5.

Remark

Two remarkable notes:

- Here you see how is that a Revision can have *two* Parents.
- Here you see that a Revision Graph can have closed loops. Whereas, for instance, you have that the topological structure of disc directories is acyclic, that is cannot have closed loops.

– –

>> Conflicts Resolved

Designed to force–declare that possible merging conflicts between the contents of changed files have been resolved manually by the operator [*cf.:* `>> Merge Changes...`].

Declaration required in case of conflicts detected during a merge action, so to be then enabled to possibly Commit the new reduced Revision.

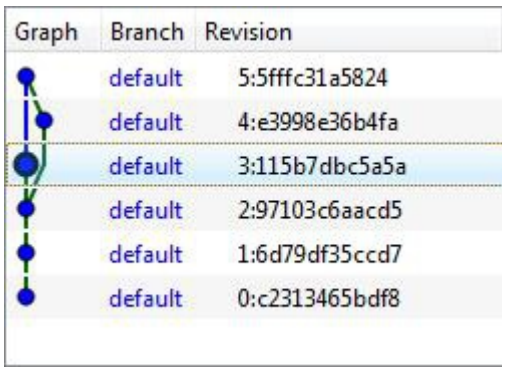
Hg Execution

resolve --mark <*file*>

– –

>> Switch...

Designed to move to a new selected Changeset, and have it as the currently Active one.



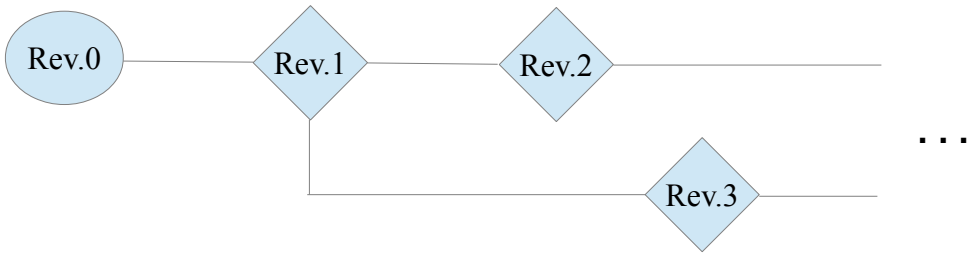
A larger dot is to mark the currently Active Changeset on the "`Mercurial Log`" graph.

– –

<!> A powerful command, acting as a sort of "time machine" for navigating back and forth from the very first to the most recent Changesete so far Committed during the Development story of the current Project [*see:* `>> Commit Changes to Repository...`]. It is the parametric version of the command `>> Update from Repository` [*see*], which is exclusively aimed at activating the Tip Changeset on the current Named Branch (by the way, precisely the default condition for this command).

Which one is the currently enabled Changeset, so to ascertain where you are move from, can be checked with command `>> Mercurial`, or continuously monitored with command `>> Show Log Browser` [*see*].

<!> Switching back to an "old" Revision, and then Committing the changes possibly there introduced, will generate a new Head on a new so called Topological Branch of the same Named Branch. A process commonly called Fork.



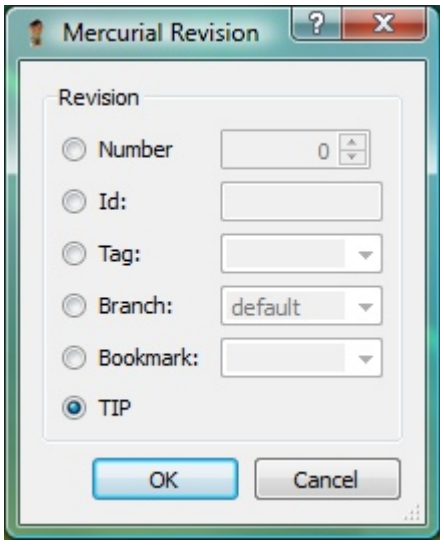
The two consequent Heads of this forked graph can be separately developed, kept separated, or subsequently merged again into one single Tip Revision, at will; with command `>> Merge Changes...` [*see*].

Remark

<!> In case of any changes introduced on the current Working Revision, it is important to be aware that:

- Moving with this command (or also with the similar: `>> Update from Repository` [*see*]) to a Revision different from the currently Active one, necessarily implies a "`merge`" process of the changes possibly already introduced [*see:* `>> Merge Changes...`]; with consequent possible conflicts. Which could not be exactly what you had in mind.
- Two actions for avoiding what in the previous point:
  - Eider first Commit current Working Revision [*cf.:* `>> Commit Changes to Repository...`], and then move to the other desired one;
  - or, if appropriate, drop all changes with the command `>> Revert Changes` [*see*].
- Note that in case of a dummy move, that is a move aimed at the same currently Active Revision, nothing will happen. In particular the current Working Revision will not be reset to the original contents.

– –



Viewpoint

<~> It would be nice here to see, as default, all values from the currently Active Revision; not only the Branch.

Control Box

`Number`, `Id` — Absolute reference to a local Revision No, or to a global hexadecimal Id [*see: Revision Number, Revision Identifier, as defined at section* Terms and Concepts]. Use the `>> Show Log` command to inspect all currently available values [*see*].

`Tag` — Tagged Changeset, with drop-down list [*see:* `>> Tag in Repository...`].

`Branch` — Head of the selected Branch, with drop-down list.

`Bookmark` — Changeset pointer, as defined with command `>> Extensions > Bookmarks` [*see*].

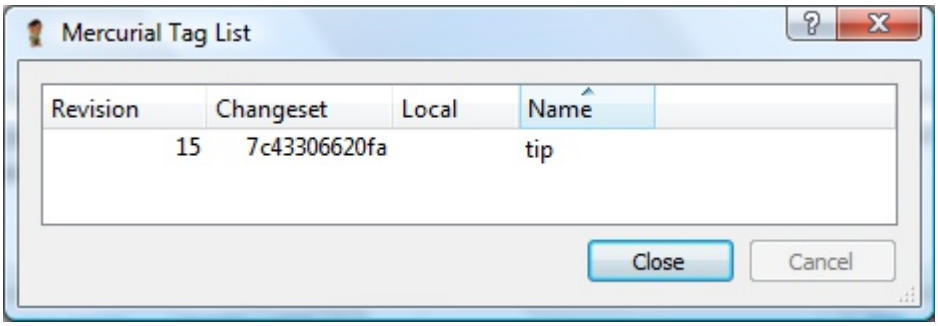
`TIP` — Most recent Changeset relative to the current Named Branch, that is its Head [*see: definition of* Development*, at* Terms and Concepts]. Note that this is NOT necessarily the Repository Tip, as this label implies. Command condition equivalent to `>> Update from Repository` [*see*]. It's the default value for this command.

Viewpoint

<~> This inaccurate "`TIP`" label reveals a rather serious conceptual confusion spread throughout the whole Mercurial and, as a consequence, also the E-Hg product. To ascertain it, just consider such a Graph.

Graph	Branch	Revision	Phase	Author	Date	Message	Tags	Bookmarks
	myBranch	15:7c43306620fa	draft	Mr. Author WDir3	2014-03-14 18:25	"myBranch" commit...	tip	myBMark15
	default	14:a13d4c3e7c0c	public	Mr. Author WDir3	2014-03-13 18:20	commit closed Bran...		
	myBranch	13:1688e9f51104	public	Mr. Author WDir3	2014-03-13 18:08	"myBranch" Head, t...		myBMark
	default	12:1897684a61dd	public	Mr. Author WDir3	2014-03-13 18:02	commit & close "def...		defBMark
	default	11:9842a628a74c	public	Mr. Author WDir3	2014-03-13 18:00	commit as Rev[11] i...		
	myBranch	10:ab54793ee80d	public	Mr. Author WDir3	2014-03-13 16:31	Rev[10], after just de...		
	default	9:e6ce23d19586	public	Mr. Author WDir3	2014-03-12 15:18	after added "aFile.txt...		
	default	8:cee9f6b6904d	public	Mr. Author WDir3	2014-03-10 16:25	Rev[8] after adding \...		
	default	7:15afd78915d3	public	Mr. Author WDir3	2014-03-08 18:15	Rev[7]		

There you see as selected Active Revision the No 12, of "`default`" Branch, and the "`tip`" Changeset as Rev. No 15, of `myBranch`. A fact confirmed with `>> List Tags...` command:



But then, entering command `>> Switch...` aimed at "`TIP`" or, equivalently command `>> Update from Repository`, you'll see the action not going to such a Tip, but to Revision No 14, that is the currently active "`default`" Branch's Head, instead.

Graph	Branch	Revision	Phase	Author	Date	Message	Tags	Bookmarks
	myBranch	15:7c43306620fa	draft	Mr. Author WDir3	2014-03-14 18:25	"myBranch" commit re...	tip	myBMark15
	default	14:a13d4c3e7c0c	public	Mr. Author WDir3	2014-03-13 18:20	commit closed Branch ...		
	myBranch	13:1688e9f51104	public	Mr. Author WDir3	2014-03-13 18:08	"myBranch" Head, to c...		myBMark
	default	12:1897684a61dd	public	Mr. Author WDir3	2014-03-13 18:02	commit & close "defau...		defBMark
	default	11:9842a628a74c	public	Mr. Author WDir3	2014-03-13 18:00	commit as Rev[11] in "...		
	myBranch	10:ab54793ee80d	public	Mr. Author WDir3	2014-03-13 16:31	Rev[10], after just defin...		
	default	9:e6ce23d19586	public	Mr. Author WDir3	2014-03-12 15:18	after added "aFile.txt", ...		
	default	8:cee9f6b6904d	public	Mr. Author WDir3	2014-03-10 16:25	Rev[8] after adding \aLib		
	default	7:15afd78915d3	public	Mr. Author WDir3	2014-03-08 18:15	Rev[7]		

We'll certainly follow with interest any possible evolution of this situation.

- -

Hg Execution

```
update -v -r
```

Case Log

`abort: uncommitted changes`

No switch can be executed in case of un-Committed changes under way on the Working Revision. As suggested remedy action, eider first

a: `>> Commit Changes to Repository...`

or a: `>> Revert Changes` [*see*].

`abort: unknown revision '12'!`

Suggested action: use the command `>> Show Log` to check available revision No

- -

>> Sub-Repository

For reasons hereafter explained [*see: next*  *marked paragraph*] the condition of use, tests and treatment of this command is here kept at a very basic level. That is:

- We'll simply consider a sub-Project inside a usual Eric Project, such as a specific library of Python modules located into a dedicated sub-directory.
- We'll not consider any autonomous external Project, Repository or Working Directory; and no other Version Control System than Mercurial.
- We'll consider the execution of the `Sub-Repository` commands only at their basic performance level, with no further analysis nor tests.

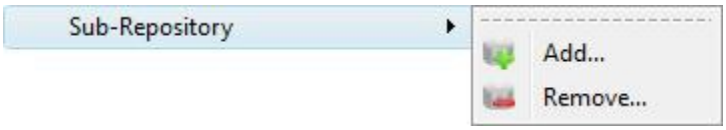
- -

It appears always as a good idea to organize any large and undifferentiated set of entities into some more manageable subsets of selected items. And this is precisely one declared purpose for the Mercurial "Subrepository" structure. Then, if you add to such structure the capacity of being not only treated as a single Mercurial unitary group, but also as an area of compatibility for such other different VC Systems as Subversion and Git, that is a sort of compatibility bridge between these systems, which is another declared purpose for this structure, you're encouraged to conclude that it is not only a good idea, but also necessary.

But then you're also warned that it is not exactly so. Indeed, in the official Mercurial "Subrepository" presentation document [*see it at URL:* http://mercurial.selenic.com/wiki/Subrepository], you're then warned that:

"*This is considered a feature of last resort.*" That is, in other terms, deprecated.

That said, here you have just a uncommitted presentation of such structure, as it is available in E-Hg. Nothing more than that.



Command List

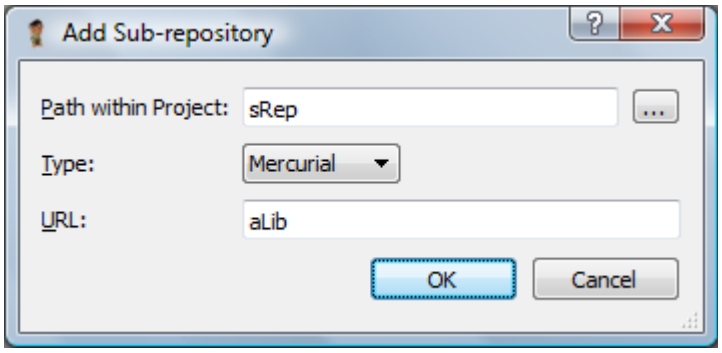
`Add...` `Remove...`

– –

>> Sub-Repository > **Add...**

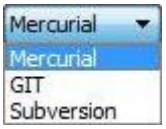
Designed to add a so called sub-Repository into the current Working Directory.

As a result you'll have a "`.hgsub`" file at the root level of the Working Directory, containing the parameters of this sub-Repository [*cf. next:* `>> Sub-Repository > Remove...`].



Control Box

`Path within Project`
Sub-Repository location (assumed existing).

`Type` Drop-down list 

of available VC System types.

`URL` Sub-Project location (assumed existing).

Hg Execution

```
add -v C:\_Activity101\Eric\Eric5\E-Hg\_Lab\WDir1\.hgsub
```

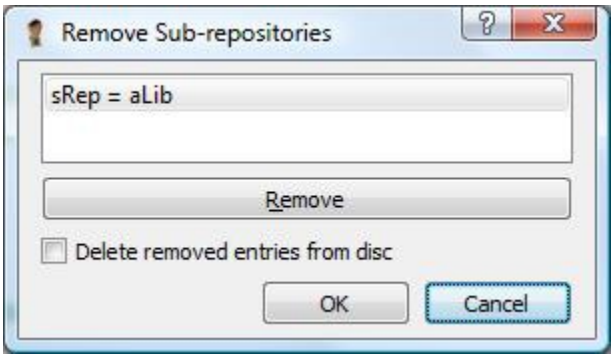
Case Log

`adding .hgsub` A "`.hgsub`" file comprising this kind of lines: `sRep = aLib`
Added at the Working Directory root level (that is, not at the Repository's metadata level), presumably so to be possibly versioned and, therefore, suitably shared if updated, to the benefit of all other related Kin Projects [*cf.:* `>> Repository Administration > Create .hgignore`].

– –

>> Sub-Repository > **Remove...**

Designed to show a box of the sub-Repositories currently defined, to be possibly removed.



<~> Note that this is the only way we've so far singled out for getting any information about the sheer existence of such sub-Repositories [*cf. former:* `>> Sub-Repository > Add...`]. Whereas no way have we found to show anything about their operative status.

– –

Viewpoint

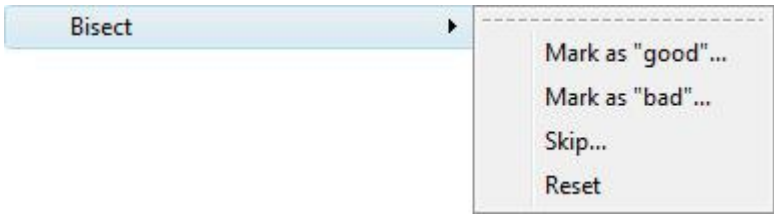
As said, no further investigation is here in order about such `Sub-Repository` feature, unless otherwise solicited, and with some good reasons.

– –

>> Bisect

Designed to offer an efficient method for dramatically restricting the number of tests to be possibly executed so to single out a specific Revision, within a given interval of consecutive Revisions. Typically for pass-fail tests, aimed at finding out where exactly a given bug, or a given unwanted behavior, has been introduced during the Development of a Project.

Once the desired test is defined, and also roughly when the sought behavior first began to appear, instead of applying such test at random, or to all the available Revisions, you will be conveniently guided to apply it within an interval of Revisions automatically selected in dichotomic sequence, according to the result of each subsequent test.



Command List

`Mark as "Good"...` `Mark as "Bad"...` `Skip...` `Reset`

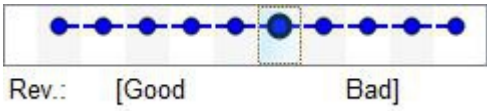
– –

Typical "Bisect" Sequence

- Chose and setup the pass-fail test to be applied. It can be a program call leading to a possible bug or to a measurement of a "suspect" execution time.
- Execute an initial "`>> Bisect > Reset`" call [*see*].
- "Mark" the endpoints of the Revision interval under test [*see:* `>> Bisect > Mark` …]. Typically: execution was "Good" until such Rev. *n1*, then became "Bad" until next Rev. *n2*, possibly the Tip. Bisect mechanism will then automatically evaluate and switch to the best candidate Revision to test between Rev. *n1* and *n2*.
- Possibly declare if any Revisions within this interval are to be "skipped", that is: not tested [*see:* `>> Bisect > Skip...`].
- Run the test.
- According to each test result, mark the current Revision as "Good" or "Bad" (i.e.: pass or fail). Bisect mechanism will then automatically evaluate and switch to the next best candidate Revision to test; or declare completed the dichotomic identification of the sought Revision.

Viewpoint

<~> It would be nice to see somewhere represented the Bisect "Good – Bad" interval, both as initially defined and then, even more important, as it varies in time after each test. Something like:

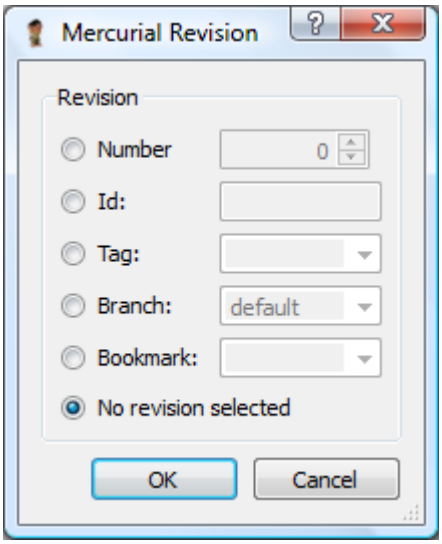


It is true that you can follow the sequence on a `>> Show Log Browser` display [*see*], but frankly, in this case, that's not so friendly.

– –

>> Bisect > **Mark as "Good"...**

Designed to mark one Changeset as "Good" and therefore, usually, as the lower boundary of the interval of consecutive Changesets where to perform the next desired test action [*cf.:* `>> Bisect > Mark as "Bad"...`].



Control Box

`Number ... Bookmark`
: Usual Revision selectors, as with command `>> Switch...` [*see*].

`No revision selected`
: This is to point to the currently tested Revision, as a convenient default during the Bisect process. Default condition.

Hg Execution

```
bisect --good 3
```

Case Log

```
Testing changeset 2:a391900dde8a (4 changesets remaining, ~2 tests)
0 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

This is also to inform about how many "Bisect" test actions presumably remain to be executed on the interval delimited by the two Changesets marked "Good" and "Bad".

– –

>> Bisect > **Mark as "Bad"...**

Designed to mark one Changeset as "Bad" and, therefore, usually, as the upper boundary of the interval of consecutive Changesets where to perform the desired test action [*cf.:* `>> Bisect > Mark as "Good"...`].

Control Box

`Revision` As with `>> Bisect > Mark as "Good"...` [*see*]

Hg Execution

```
bisect --bad tip
```

Case Log

```
Testing changeset 6:b8f1b00b3492 (6 changesets remaining, ~2 tests)
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

– –

>> Bisect > **Skip...**

Designed to possibly skip a Changeset within the ["Good – Bad"] Bisect interval of consecutive Changesets, so not to perform the stated test action [*cf.:* `>> Bisect > Mark as "Good"...`].

Control Box

`Revision` As with `>> Bisect > Mark as "Good"...` [*see*]

Hg Execution

```
bisect --skip 2
```

Case Log

```
Testing changeset 3:326bed97efad (4 changesets remaining, ~2 tests)
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

This is also to inform about how many "Bisect" test actions presumably remain to be executed on the interval delimited by the two Changesets marked "Good" and "Bad".

– –

>> Bisect > **Reset**

Designed to clean off the definition of the ["Good – Bad"] Bisect interval of Changesets.

<!> Action to be anyhow prudentially performed before any new Bisect test sequence.

Hg Execution

```
bisect --reset
```

– –

>> **Cleanup**

Designed to erase the files declared as garbage in the "`Cleanup Pattern`" field of the "`Configure Mercurial Interface`"[54] [*see:* `>> Configure... – Version Control Systems > Mercurial`]. Contributed default `Pattern`:

`*.orig` Old file contents saved by Mercurial "`resolve`" command [*cf.:* Mercurial Command Reference, *at* Mercurial Package, *in* Appendix; *and:* `>> Conflicts Resolved`].

`*.rej` File patch rejects [*see:* `>> Patch Management`].

`*~` Typical Linux suffix for temporary backup files[55].

54 <~> That is, nothing to do with the "`--cleanup`" option of the original Mercurial `shelve` command, in case you guessed otherwise.

55 Unusual for Windows.

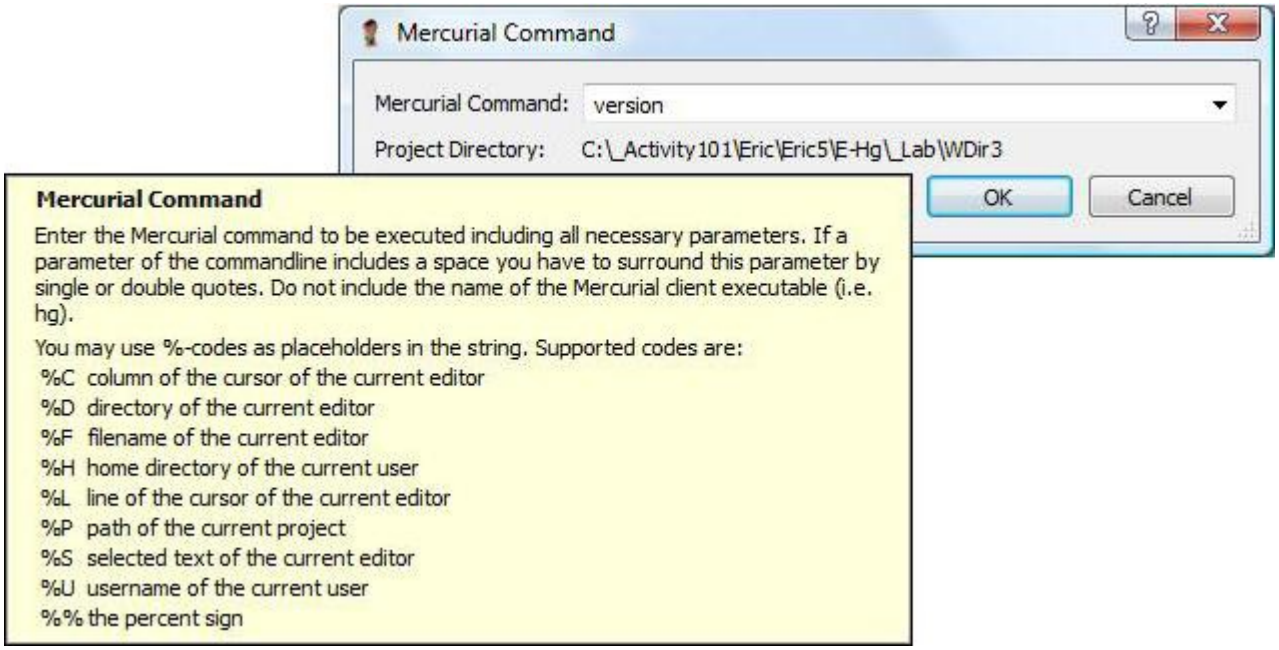
Remark

Note that, to some extent, this is a mechanism similar to the "`.hgignore`" [*cf.:* `>> Repository Administration > Create .hgignore`]. But note also that this is a clear example of a "pure" E-Hg command, that is not wrapping or implying the execution of any proper Mercurial command.

– –

>> Execute Command...

Designed to offer a gate through which direct Mercurial commands—that is: not E-Hg's—can be entered, as shown with an example hereafter [*cf. also:* `>> Command Options...`].



As the presence of this command clearly confirms, E-Hg is not indented to entirely mask and substitute the underlying Mercurial interface[56], whose direct knowledge is anyhow here assumed as useful, if not even required. It is rather intended to make things remarkably easier and friendlier and, most important, fairly integrated into the Eric environment.

– –

56 Compare, for instance, the Windows GUI TortoiseHg version [*see in:* `bitbucket.org`].

Control Box

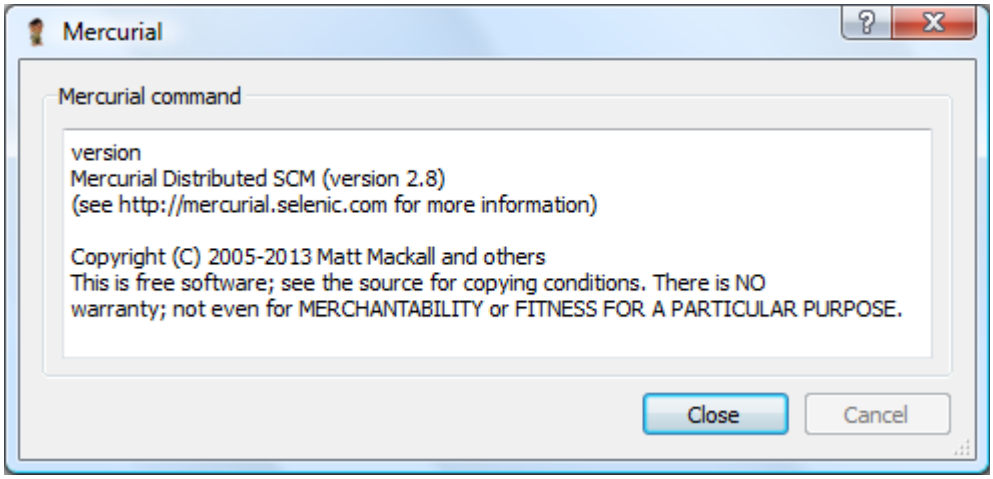
`Mercurial Command`

Without the leading "`hg `" prefix, and with a history drop-down list

`Project Directory`

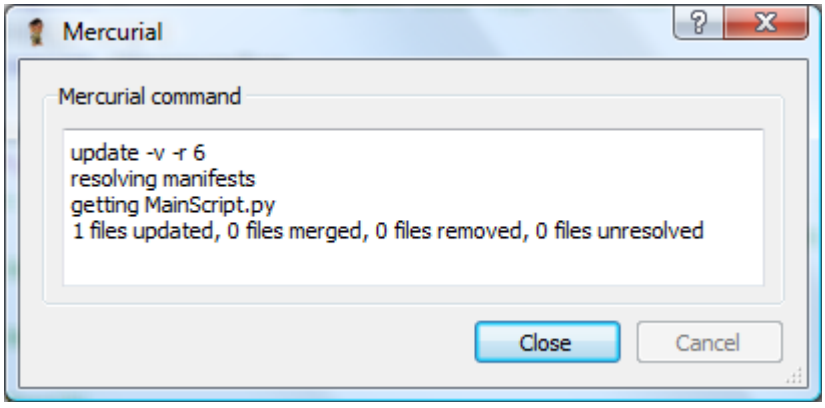
Defaulted path

Case Log



Remark

<!> Note that this is not an Eric message but, obviously, a Mercurial's, consequent the entering of a Mercurial command [*that is:* `version`]. But note that also when entering E-Hg commands equivalent to some Mercurial ones, consequent messages are Mercurial generated, not E-Hg.

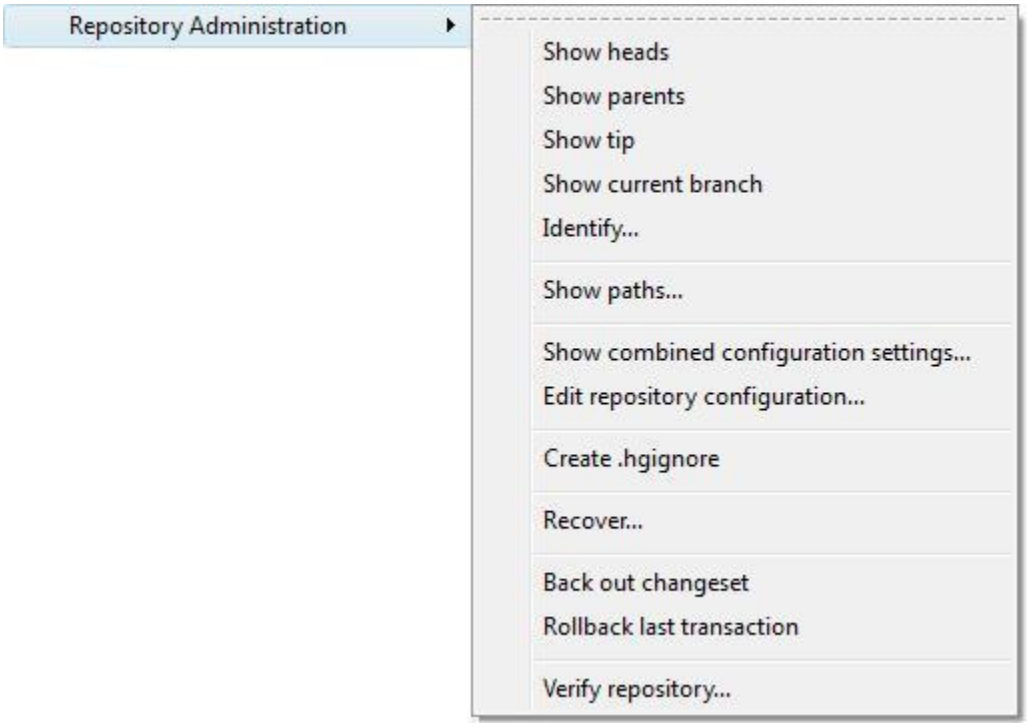


This is to be remembered when using E-Hg, so to interpret correctly the messages—otherwise possibly a bit obscure as this "`resolving manifests`"—according to their source.

– –

>> Repository Administration

The right place where to inspect and manage "all" about the current Mercurial metedata Repository.



Command List

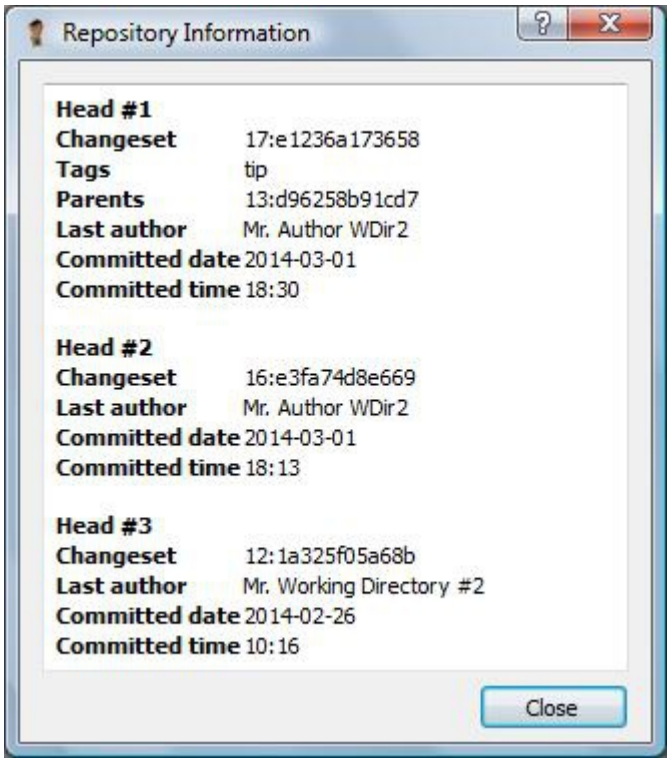
```
Show Heads            Show Parents       Show Tip             Show Current Branch
Identify...[57]       Show Paths...      Show Combined Configuration Settings...
Edit Repository Configuration...         Create .hgignore     Recover...
Back Out Changeset    Rollback Last Transaction               Verify Repository...
```

– –

57 Ellipsis "…" unnecessary, soon removed.

>> Repository Administration > **Show Heads**

Designed to show all Repository's Changeset Heads, that is Changesets with no Children (i.e.: Descendants) [*cf.:* `> Show Parents`].



Where:

`Head` Item number, as more than one Head can be found in a Repository

`Changeset` Identifiers, with format: <local `No`>:<global `Hex`>

`Tags` String identifier, with the most recent Changeset automatically labeled: "`tip`". N.B.: "`Tags`" plural, as more than one may be associated to the same Changeset, such as the automatic "`tip`", plus possibly another one assigned with command `>> Tag in Repository...` [*see*].

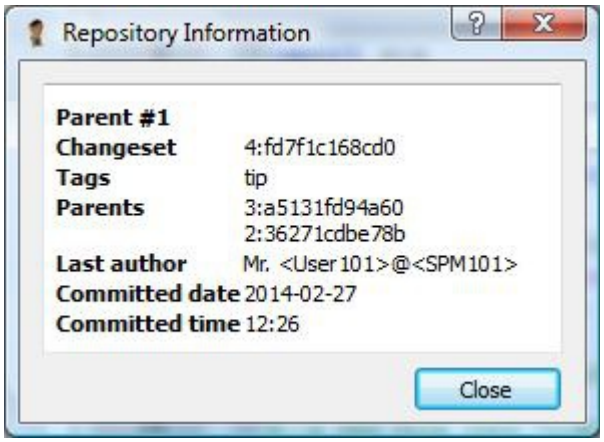
`Last author` Commit author name [*ref.: parameter* `username`, *at* Configuration Files, *in* Appendix].

`Committed` date and time

– –

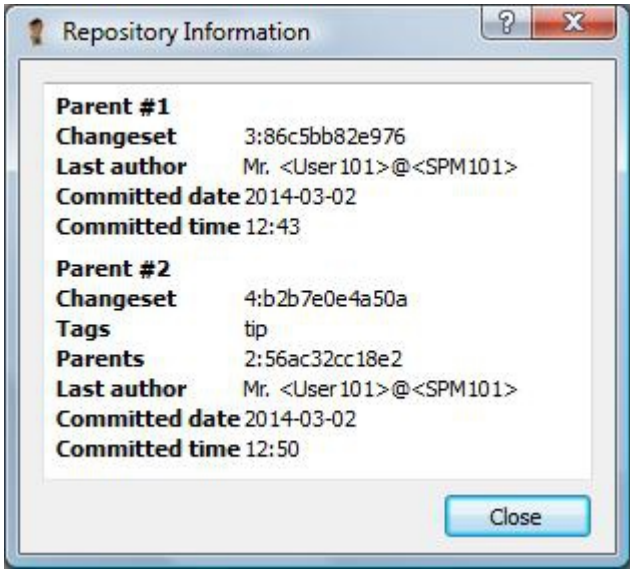
>> Repository Administration > **Show Parents**

Designed to show the Parents of the current Active Changeset. Typically just one, but also possibly two, if the Changeset is the result of a merge of two Branches [*see:* `>> Merge Changes...`].



Where:

`Parent` Item number. Two at most, in case of a merge [*as in the example hereafter*]



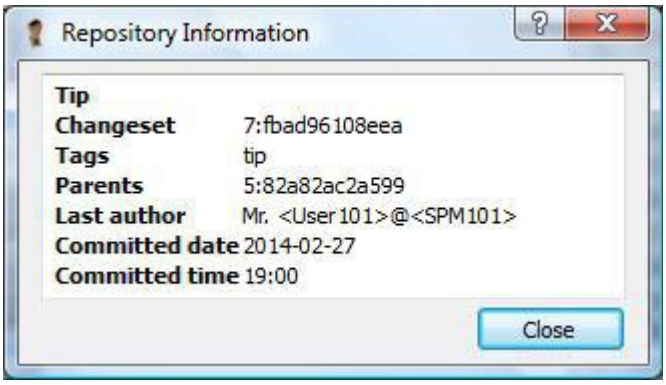
`Changeset` Identifiers, with format: <local `No`>:<global `Hex`>

... Others, as with `> Show Heads` [*see*]

– –

>> Repository Administration > **Show Tip**

Designed to show which is the most recent Changeset of the Repository, automatically tagged "`tip`".



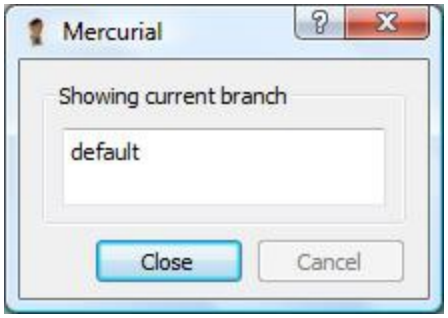
Where:

`Tip`	Info header (singular, as there can be only one Tip in a Repository).
`Changeset`	Identifiers, with format: <local `No`>:<global `Hex`>
...	Others, as with `> Show Heads` [*see*]

- -

>> Repository Administration > **Show Current Branch**

Designed to show which is the name of the currently active Branch on the Working Directory. Information of great practical relevance, as the ramification of a Project into distinct Branches is commonplace.



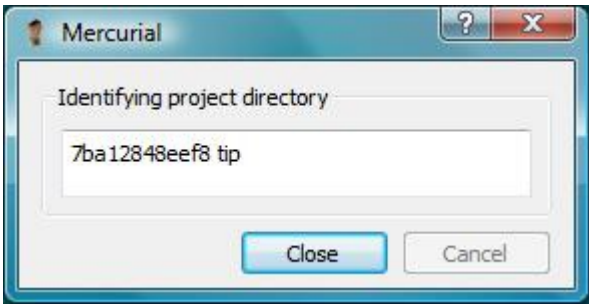
Remark

Note that one Branch is always defined. Initially it is the so called "`default`", automatically created at the very Repository inception. Others can be then custom defined and enabled [*see:* `>> Create Branch...`].

– –

>> Repository Administration > **Identify**

Designed to identify the current Active Chageset, that is to show its hexadecimal global identifier, possibly followed by Tag and Bookmark, if any.



Viewpoint

<~> The "`project directory`" heading is here intended to signify the contents of the directory, not the very directory in itself which, of course, is identified by a path and not by a hexadecimal code.

That said, it's nevertheless our opinion that such kind of Mercurial terminology could easily lead to confusion, and therefore this is precisely the reason why, in this Report, we are consistently trying to use the terminology adopted with the initial section "Terms and Concepts" [*see, just after* Essentials].

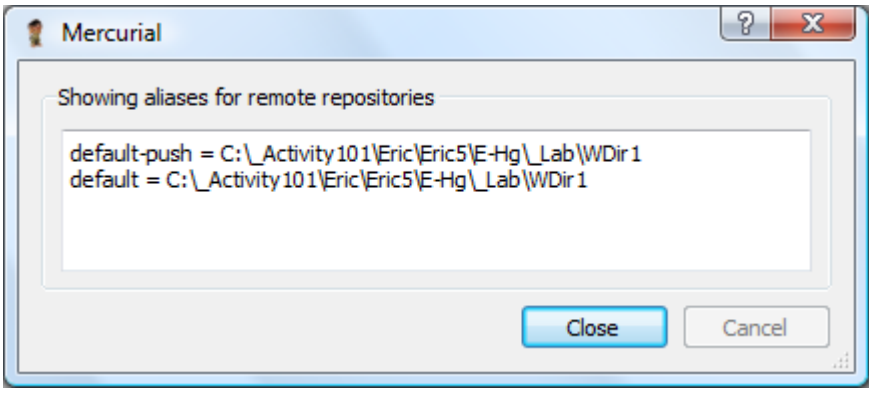
For instance, saying here Active Revision or, equivalently, Active Changeset, is by far clearer than "`project directory`". Isn't it?

– –

>> Repository Administration > **Show Paths...**

Designed to show the paths, or URLs, as defined in `Mercurial.ini` and `.hg\hgrc` configuration files [*see:* Configuration Files*, in* Appendix] and enabled for such E-Hg commands as: `>> Push / Pull`

`Changes` [*see*].



Viewpoint

<~> "`aliases for remote repositories`" doesn't seem to us an appropriate heading for this box.

– –

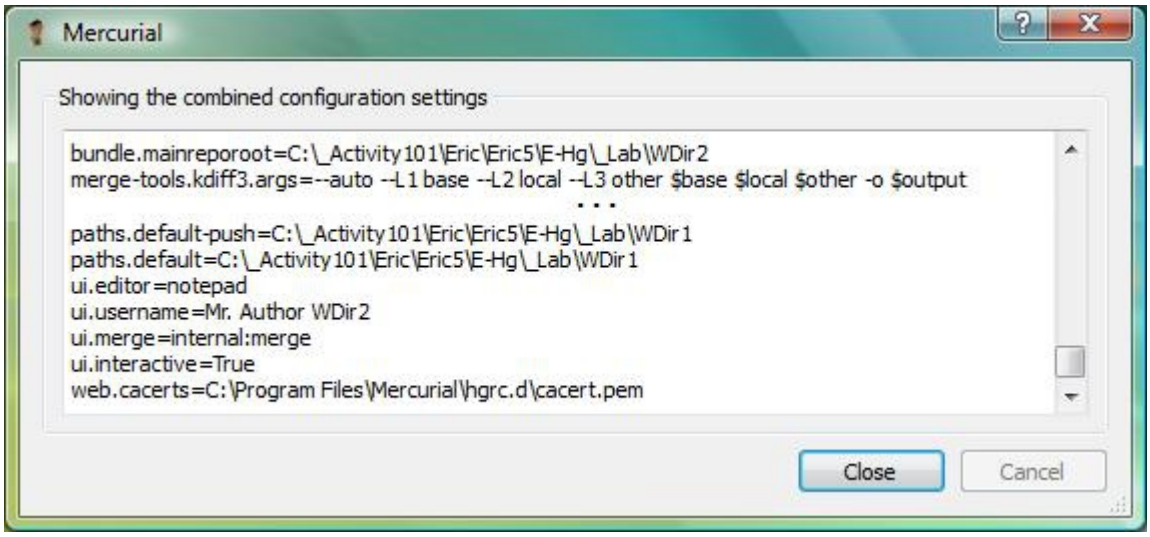
>> Repository Administration >

Show Combined Configuration Settings...

All configuration parameters relevant for E-Hg operations are here described at the Configuration Files section [*see: in* Appendix], and can be inspected, and edited, via commands: `>> Edit User Configuration...` and `>> Repository Administration > Edit Repository Configuration...` [*see*].

Besides that, Mercurial can have many other configuration settings—a subject off the scope of this Report—, as described in the standard "Mercurial Configuration Files" documentation [*ref.: at* Mercurial Package*, in* Appendix], recorded in different configuration files and with a different priority level, in this order: Global, User, Repository.

This command here is provider to merely show which is their current resulting list and configuration.



Note that:

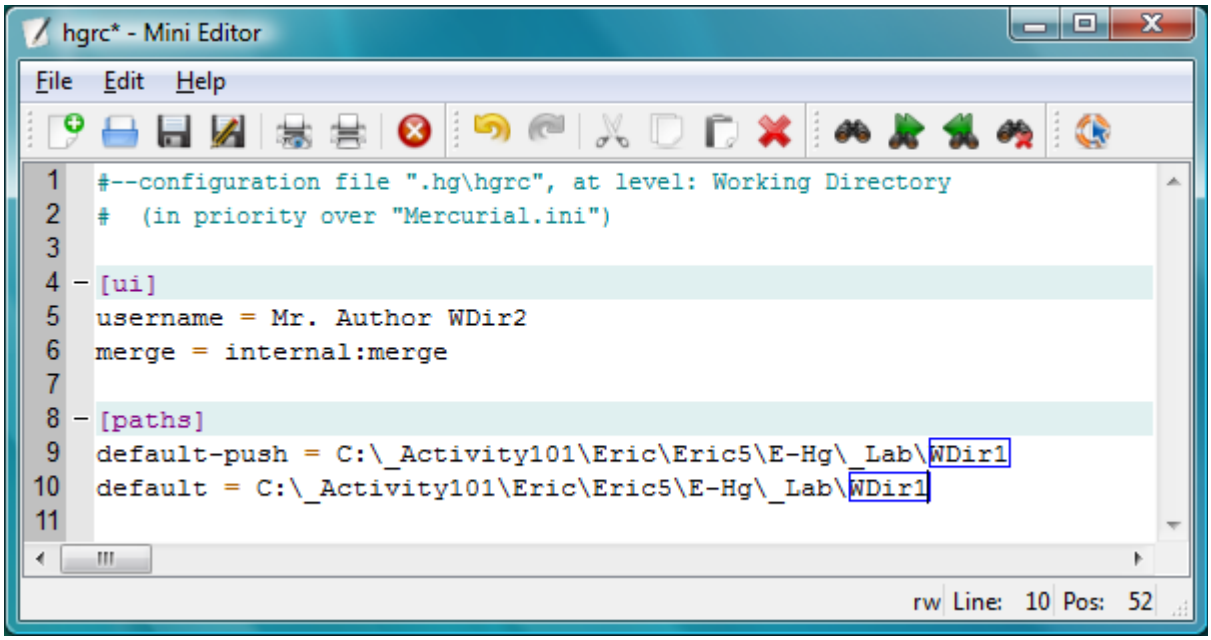
- It's easy to single out the specific E-Hg parameter sections `[ui]` and `[paths]`, even though here listed with a different format; e.g.: "`ui.`", instead of "`[ui]`".
- Beside the usual E-Hg parameters sections, there are such others as: `bundle`, `merge-tools`, and `web`. Anyhow, as said, a subject beyond the scope of this Report.

– –

>> Repository Administration >

Edit Repository Configuration...

Designed to run an editing session on the E-Hg "`hgrc`" configuration file, as stored into the current "`\.hg`" Repository directory.



Rather convenient a tool where to inspect and manage this file. For detailed description see section Configuration Files, in Appendix [*see*].

– –

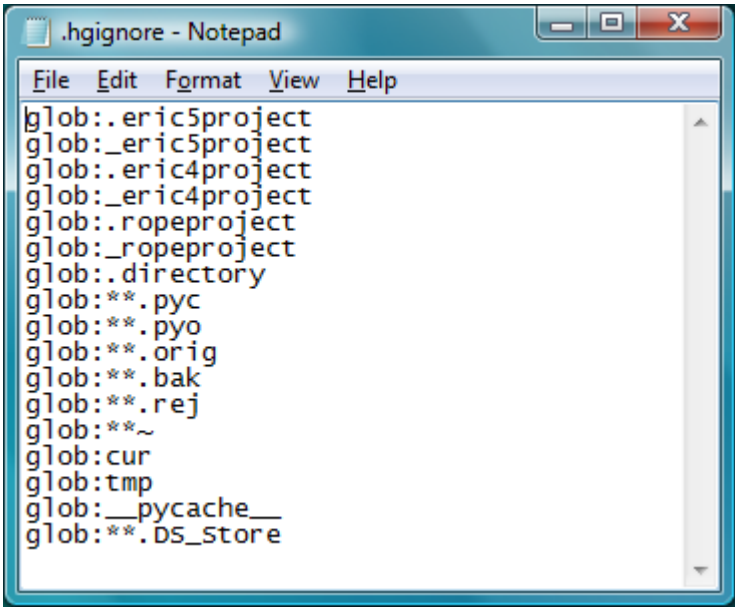
>> Repository Administration > Create `.hgignore`

Not all files belonging to an Eric Project—as with any projects in general—need to be version controlled, for instance there is no need to keep track of what happens to temporary files. That's why such a "`.hgignore`" file is provided, to tell which are the files to be anyhow ignored by E-Hg. This command is here provided to create, or re-create, such a "`.hgignore`" file[58].

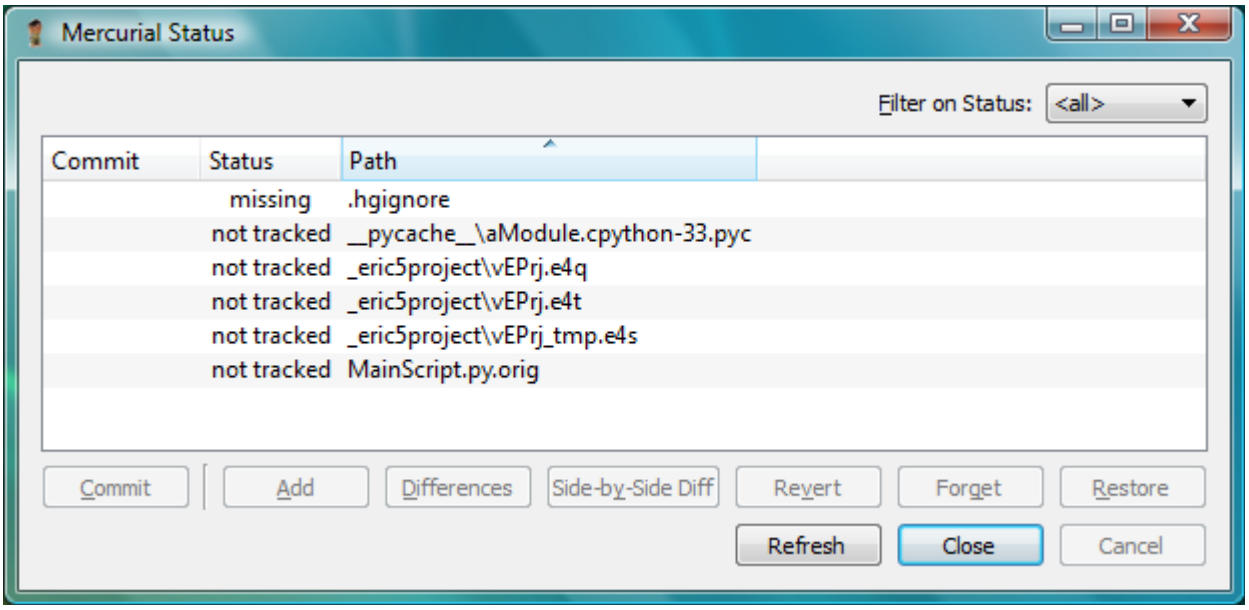
Use any text editor to possibly modify it, with Mercurial Ignore Files as the standard reference [*ref.: at* Mercurial Package*, in* Appendix][59]. Hereafter an example.

58 To some extent a mechanism similar to the `>> Cleanup` [*cf.*].

59 <~> Note that, if you create it from scratch, Windows file manager will reject this name, as starting with "`.`" (dot), a character reserved as extension separator. A difficulty easy to overcome when saving the file with the standard Notepad text editor, where the desired "`.hgignore`" name is well accepted if you save it as an "`All Files (*.*)`" type.



Files here listed will be ignored by E-Hg, that is, also when calling `>> Show Status...` [*see*], where, otherwise, they would always aimlessly appear as "`not tracked`"; as shown in the following example.

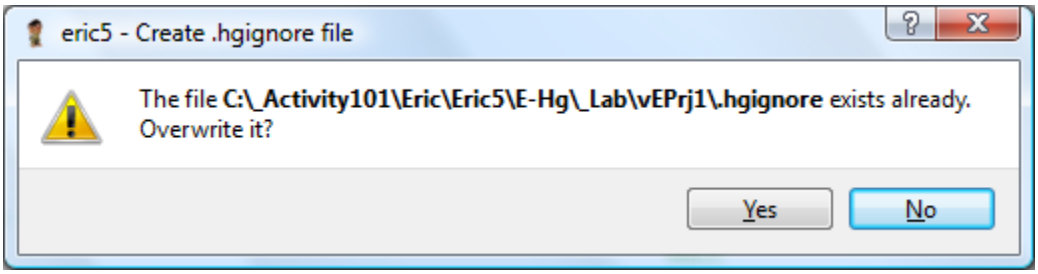


– –

This "`.hgignore`" file has, in some way, the inverse role of the context command `Project-Viewer(^)> Add to Repository`, which is precisely provided to instruct E-Hg to take care of a new

file possibly added to the Project [*see*]. With some peculiarities though, as hereafter described.

- Automatically created at start by E-Hg, already suitably configured for a standard Eric Project [*see it with any text editor*].
- Can be anyhow re-created with this very command, if the case. Indeed, this is precisely its purpose. And that's why you're warned this way:



  so not to inadvertently lose valid changes, in case it has been deliberately customized.
- This file is conveniently located inside the Project directory as it is meant to be versioned and, therefore, suitably shared if customized[60], to the benefit of all related Kin Projects.
- In other words, this is a file that, although with a specific Mercurial role, is not located into the "`\.hg`" metadata Repository, but into the very Project directory. And, therefore, there kept also in case of a `>> Export from Repository...` action [*see*]; even though then useless, in that case.

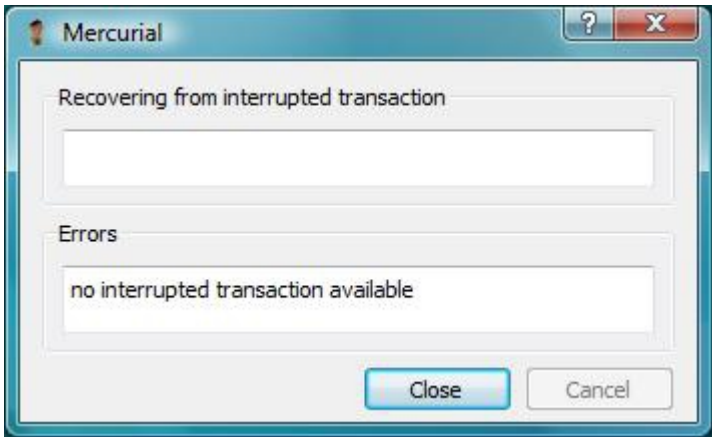
– –

>> Repository Administration > **Recover...**

This is a special command to be entered only upon an explicit Mercurial suggestion[61], so to possibly recover the Repository status after an interrupted Commit or Pull action [*see:* `>> Commit Changes to Repository... / Pull Changes`].

60 An occurrence that you'd rather avoid, thanks to the quality of the original E-Hg "`.hgignore`" file as here contributed. Otherwise you'd have to learn another [*another!*] coding dialect, the so called "`glob`", spoken by some Unix tribes.

61 <~> That is, a message like: `[abort: abandoned transaction found - run hg recover!]`



A no-action command, if entered when not required.

– –

>> Repository Administration > **Back Out Changeset**

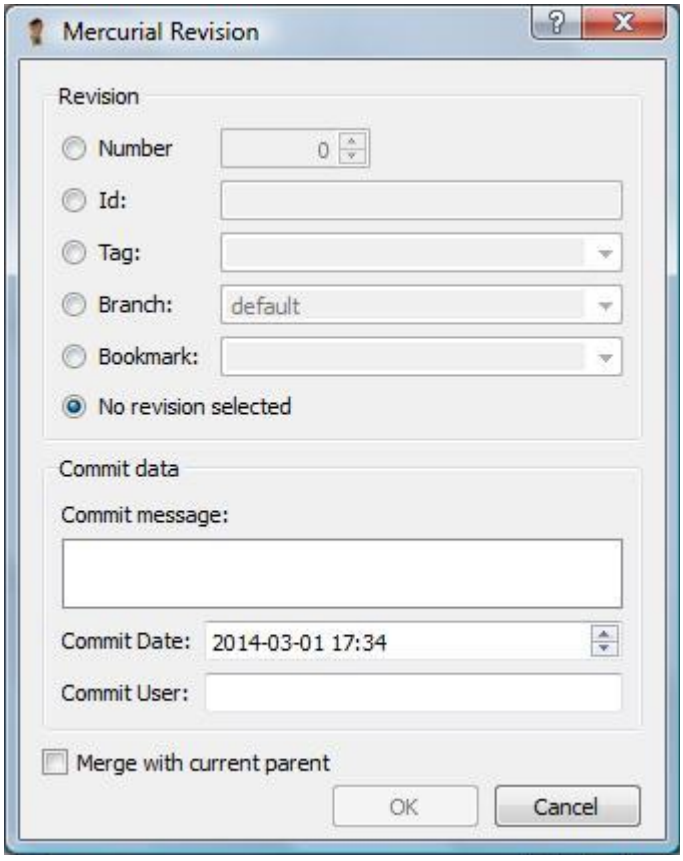
Designed to create a so called "Backout" Changeset, that is the "negative" version of a given Changeset, aimed ad "undoing" it. It's therefore a way for canceling a undesired Changeset already Committed which, as such, could not be erased anymore from a versioned Development story [*cf.: section* Undo and Redo of Commit Changes*, after command:* `>> Commit Changes to Repository...`].

Backout Action, Operative Aspects

- Cannot execute in case of un-Committed changes.
- Cannot execute on different Branches.
- The simplest execution is when you "Back out" the Head of a Branch. The result is a next Committed Head Changeset, whose action is simply that of undoing the preceding Changeset, after which you will proceed as if the "Back outed" Revision never existed.
- A bit more complex is the case when the selected Revision to "Back out" is not a Head, but a preceding Ancestor. In that case the execution implies a Topological Branch stemming out from the selected Changeset with, as new Head, the desired Backout Changeset (that is the Changeset capable of undoing the selected one). And, as the second Head, the "old" one of the initial Topological Branch.
- None of the two Heads becomes the Active Changeset, as these two Heads are expected to be manually merged into one new Head of one reduced Branch.

- Then the user is expected to explicitly Commit this new merged Head Changeset, after a look at the resulting source Project, after the manual solution of possible merge conflicts, and after a final declaration of `>> Conflicts Resolved` [*see*].

– –



Control Box

`Revision` <~> This is just an area header, but rather interesting, as it implies the equivalence between the terms Changeset—as in this very command name—and Revision—as in this very command box—, as assumed also throughout this whole Report [*see:* Terms and Concepts, *just after* Essentials].

`Number ... Bookmark`
Usual Revision selectors, as with command `>> Switch...` [*see, <~> and be informed that "*`TIP`*" might not point to what you'd expect; see also:* Dealing with Mercurial Tip, Heads and Active Revision, *in* Terms and Concepts].

`No revision selected`
: Not a valid option: `OK` button remains disabled.

`Commit message`
: Usual Commit message, here defaulted as: `Backed out changeset <No>.`

`Commit Date` Default: current

`Commit User` Optional, otherwise usual default.

`Merge with current parent`
: Ignored / dummy [<~> *corresponding Mercurial "*`--merge`*" option seen anyhow always enabled; and correctly so, we deem*]

`OK` Execute button, enabled only after a valid "backout" selection.

– –

Hg Execution

```
backout -v --merge --message Backed out changeset <No>. <No>
```

Case Log

`abort: uncommitted changes`
: No Backout action admitted with un-Committed changes

`abort: cannot backout change on a different branch`
: No Backout action admitted on different Branches

```
committed changeset 4:427665ec05f1
changeset 4:427665ec05f1 backs out changeset 3:115b7dbc5a5a
```
: Simple case: "Back out" and automatic Commit of a Head Changeset

```
warning: conflicts during merge.
merging MainScript.py incomplete! (edit conflicts, then use 'hg resolve -mark')
```
: Not-so-simple case: "Back out" of an Ancestor Changeset, then requiring user intervention to be completed (see above detailed description).

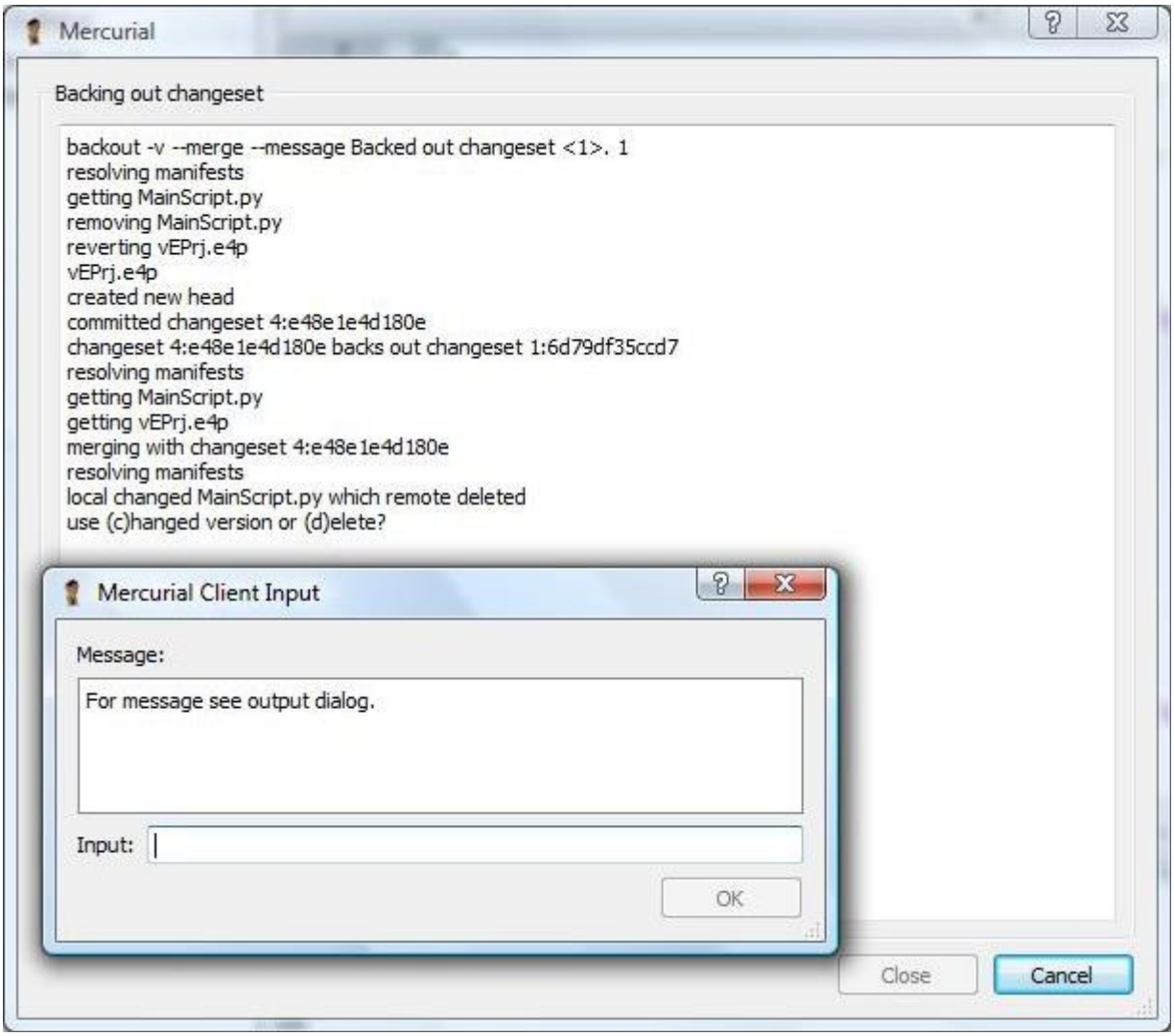
`(branch merge, don't forget to commit)`
: A merge case with no conflict, just follow the instructions.

– –

Remark

<~> Real life cases of Ancestor's Back-Out can be too intricate to be here all considered and described. Just as a notable example, consider the case when you are trying to undo a Changeset involving the addition of a new versioned source file, possibly than required by subsequent Revisions.

Is such a case you may have Mercurial "in doubt", and therefore asking you what to do. Here is an example:



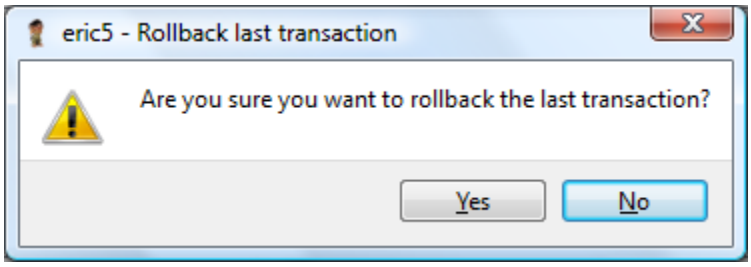
Here the question is: "`use (c)hanged version or (d)elete?`", with the "`Mercurial Client Input`" E-Hg box where to enter the expected answer and with, as default, a prudential: "`(c)hanged`"; that is: not "`(d)elete`".

In such cases it's perhaps better not to rely too heavily onto this "Back Out" technique, and simply intervene modifying things as a normal editing activity.

– –

>> Repository Administration > **Rollback Last Transaction**

Designed to undo the last command—only—implying the creation of a new Changeset, being it a transnational action. That is, such commands as: `>> Commit Changes to Repository...` / `Pull Changes` and `>> Patch Management > Import Patch...`

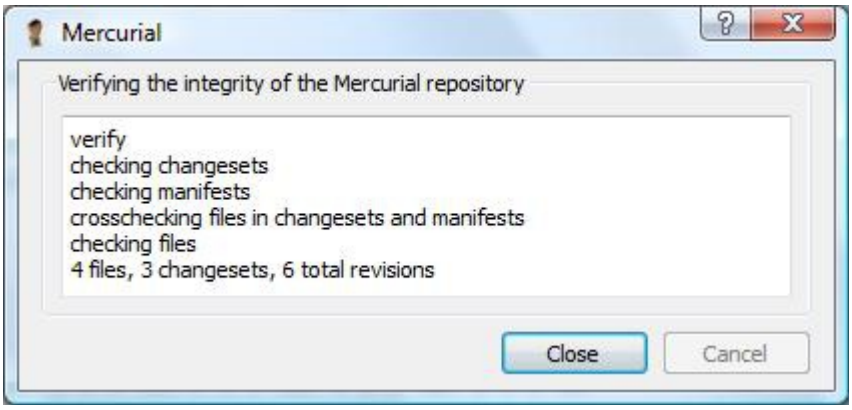


Before proceeding you'll be asked to confirm, as there is no redo for this action [*cf.: section* Undo and Redo of Commit Changes, *after command:* `>> Commit Changes to Repository...`].

– –

>> Repository Administration > **Verify Repository...**

Designed to verify the integrity of the current Repository as a first aid action in case of suspected corruption. Typically: with hardware failure, or a user error affecting "`\.hg`" metadata files.

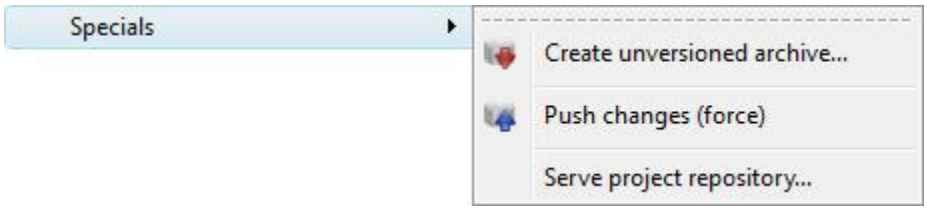


– –

>> Specials

A sub-menu designed to put up a "special" version of two E-Hg commands as available elsewhere, in their more usual form, and a third one with no correspondence in any other E-Hg menus. That is:

- "hg `archive`" a special version of: E-Hg `>> Export from Repository...`
- "hg `push -f`" a special version of: E-Hg plain `>> Push Changes`
- E-Hg server activation



Command List

`Create Unversioned Archive...` `Push Changes (force)` `Serve Project Repository...`

– –

>> Specials > Create Unversioned Archive...

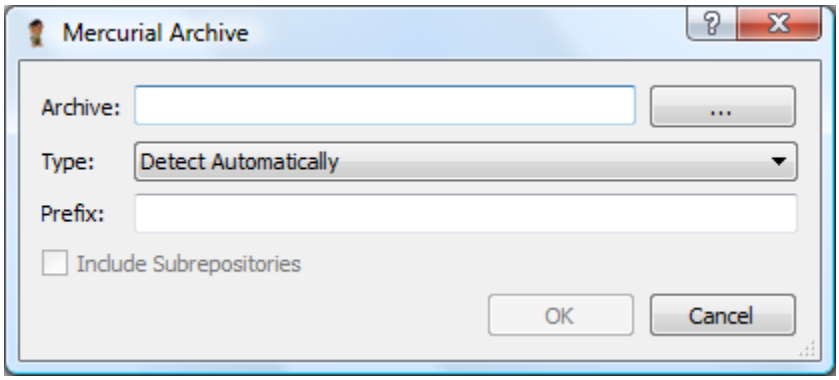
Designed to wrap the original Mercurial command "`archive`"[62], basically aimed at extracting out of the Working Directory the sheer "unversioned" Project, as in the currently Active Revision [*see command:* `>> Switch...`], and saving it into an "Archive". That is, only the Project, without the Mercurial metadata Repository.

Remark

It's a command similar to the `>> Export from Repository...` [*see*], presumably made here available to the benefit of users preferring this original—and also more versatile—Mercurial version, than the cited E-Hg's.

Note that E-Hg permits also the execution of the inverse action, that is the versioning of an existing Project, with command: `>> Add to Repository...` [*see*].

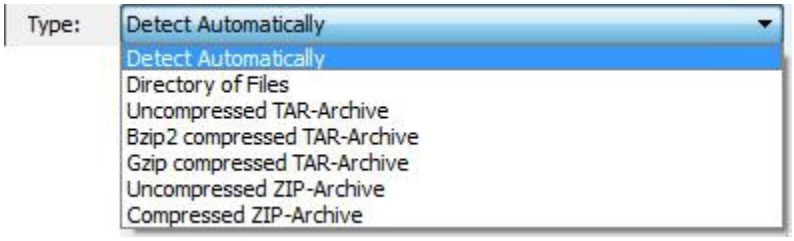
62 As [*very poorly*] described in the official "Mercurial Command Reference" [*ref.: at* Mercurial Package*, in* Appendix].



Control Box

`Archive` Name of the Archive to be generated, can be: a plain directory, or an equivalent standard compressed file, created if not existing, <!> *overwritten if existing*.
No default value. Default location: current Working Directory

`Type` Type of Archive—that is: directory or compressed file—to be generated, chosen from a drop-down list.



Where:

`Detect Automatically`
Type of archive derived automatically from the "`*.xxx`" extension suffix possibly assigned to the `Archive` name [*see cases hereafter*]. Otherwise a plain directory, for no extension.

Remark

<~> Note that, in case of an explicit selection of one of the six types as hereafter listed, no file type extension will be automatically added to the resulting Archive, which therefore might well retain a discordant extension, or no extension at all. No such danger when using the "`Detect Automatically`" option, as such preferable.

– –

Explicit `Type` list:

`Directory of Files`	`[--type files]`	Ordinary directory of files
`Uncompressed TAR-Archive`	`[--type tar]`	"`*.tar`" file
`Bzip2 compressed TAR-Archive`	`[--type tbz2]`	"`*.tbz2`" file (using `bzip2`)
`Gzip compressed TAR-Archive`	`[--type tgz]`	"`*.tgz`" file (using `gzip`)
`Uncompressed ZIP-Archive`	`[--type uzip]`	"`*.uzip`" file if explicit, but <~> an ordinary directory of files, if automatic.
`Compressed ZIP-Archive`	`[--type zip]`	"`*.zip`" file

– –

`Prefix` Optional root-directory name where to expanded a compressed file type archive, such as "`*.zip`". It doesn't apply when archiving directly into plain directories of files [*see: next* ***Case Log***].

`Include Subrepositories`
Check-box to include possible "Subrepositories" [*see:* `>> Sub-Repository`].

Hg Execution

```
archive --prefix RootDir myPrj.zip
```

<~> In this example you have the creation of a "`*.zip`" type archive which will be extracted into a "`\RootDir`".

Case Log

`abort: cannot give prefix when archiving to [`*directory of*`] files`
<~> This is in case of a `Prefix` erroneously assigned when archiving into an ordinary directory of files, and not into such structured files as "`*.zip`" [*see:* ***Hg Execution*** *example above*].

– –

Remark

Some notable aspects about the execution of this command, worth knowing.

- What is archived is not, and cannot possibly be, an absolutely "pure" Project. Indeed such kind of specific Mercurial "`.hg`"-files are archived too: `.hgignore`, `.htags`. …

- Also this other file is found automatically added to the lot: `.hg_archival.txt`, comprising such kind of Revision identifying parameters:

```
repo: c2313465bdf866f3c6c842666081b18be1b97a07
node: ac8f7479a41ec2cd5c428c88c2dd346b2740683a
branch: default
latesttag: null
latesttagdistance: 13
```

- -

>> Specials > **Push Changes (force)**

Same as `>> Push Changes` [*see*], just with added the "`-f`" option, so to force anyway the Push action, at your own risk. We remember that with the not-forced Push execution:

- If any change has been independently introduced into the Receiver Working Directory, the Push action should necessarily create a new remote Head on a Topological Branch, which is not advisable, as it would there make unclear which is the head in use. In such a "`new remote head`" case you'll be informed, with an "`abort`" message, that the Push action has not been executed.

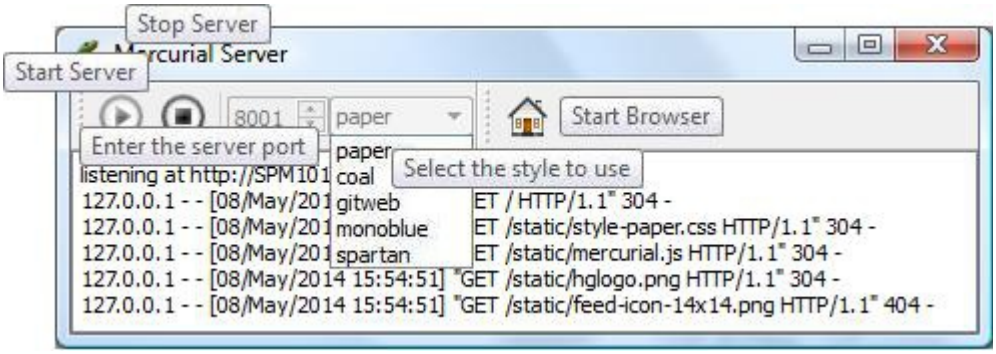
  The suggested remedy is first a plain Pull and merge action executed locally, with possible conflicts resolved; then followed by the desired Push.

  In other words:
  <!> In case of a contemporary not-empty Incoming *and* Outgoing Log, it is advisable first to execute a Pull, and only then a Push.

- -

>> Specials > **Serve Project Repository...**

<。?。> We heard that this command is aimed at running a Mercurial server enabling your local versioned Project to become HTTP-visible in the Internet. Great, wonderful. But then we realized it as a nice example of an interesting and promising command, completely useless because of a missing crucial information.



Control Box

`Start / Stop Server`  / 

Self explanatory controls.

`Enter the server port`

Presumably a numeric parameter playing the role of the Port number, required to identify a specific "extension" (i.e.: sub-address) within a single computer identified by a unique IP address in a TCP/IP protocol network.

Port numbers are used to convey data to the "right" application among the possibly many ones concurrently running in the same computer. It's a number in the range of 0÷65535, conventionally divided into the three sub-ranges[63] called: *well-known ports* 0÷1023, *registered ports* 1024÷49151, and *dynamic* or *private ports* 49152÷65535.

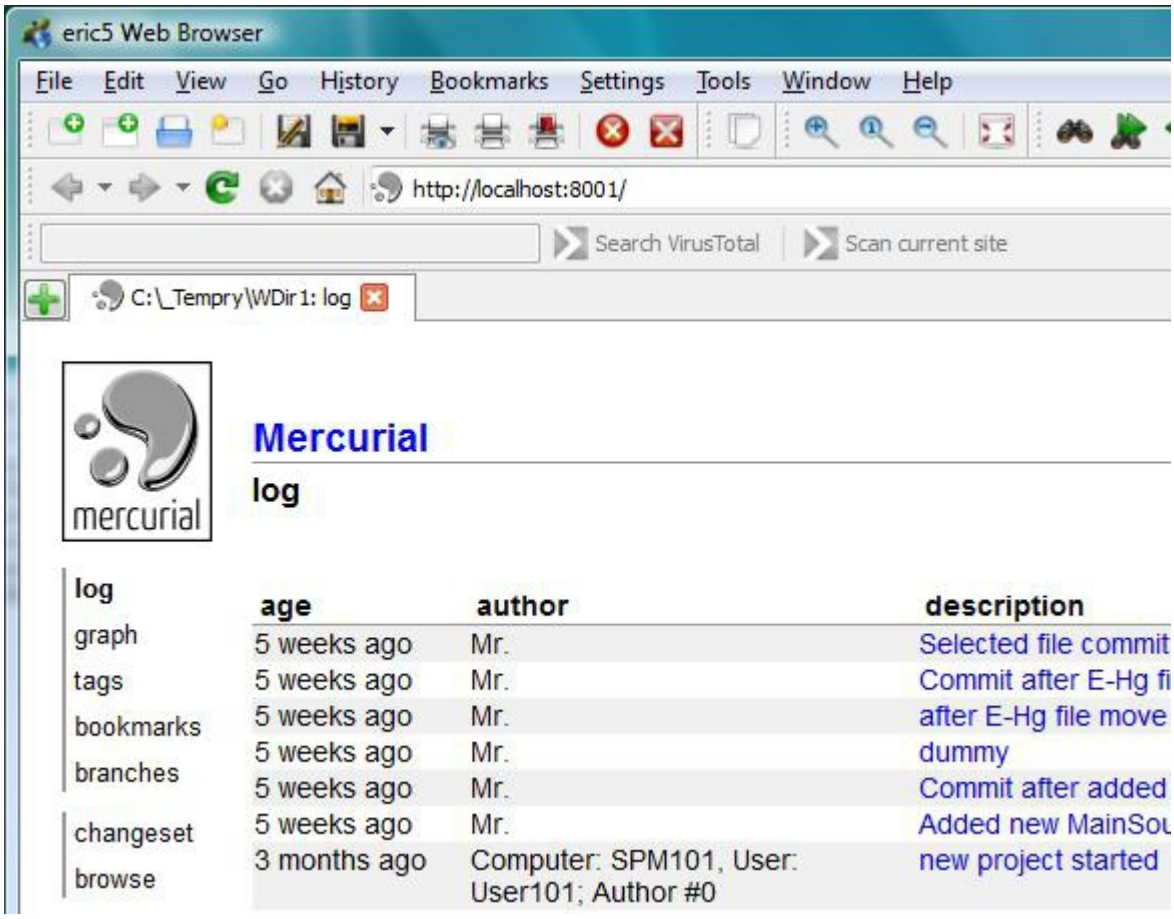
`Select the style to use`

"`style`" of what unknown, as a parameter not really tested.

`Start Browser` 

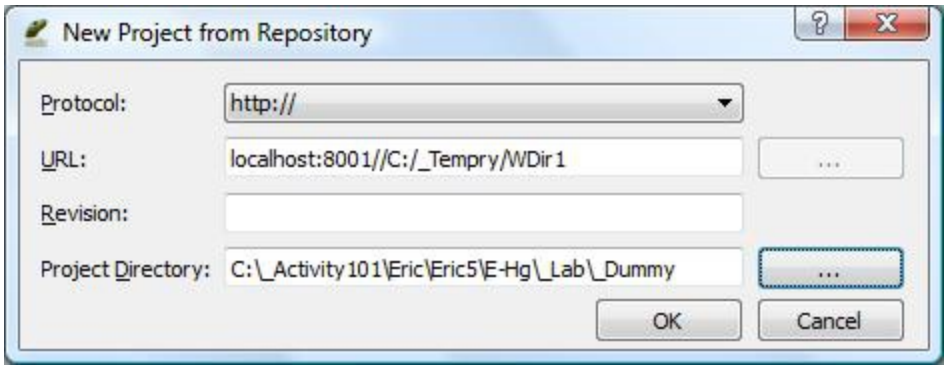
Apparently a self-explanatory feature, in reality incomprehensible as we have no precise idea how to interpret such resulting display of its:

63 A convention overseen by the Internet Assigned Numbers Authority (IANA).

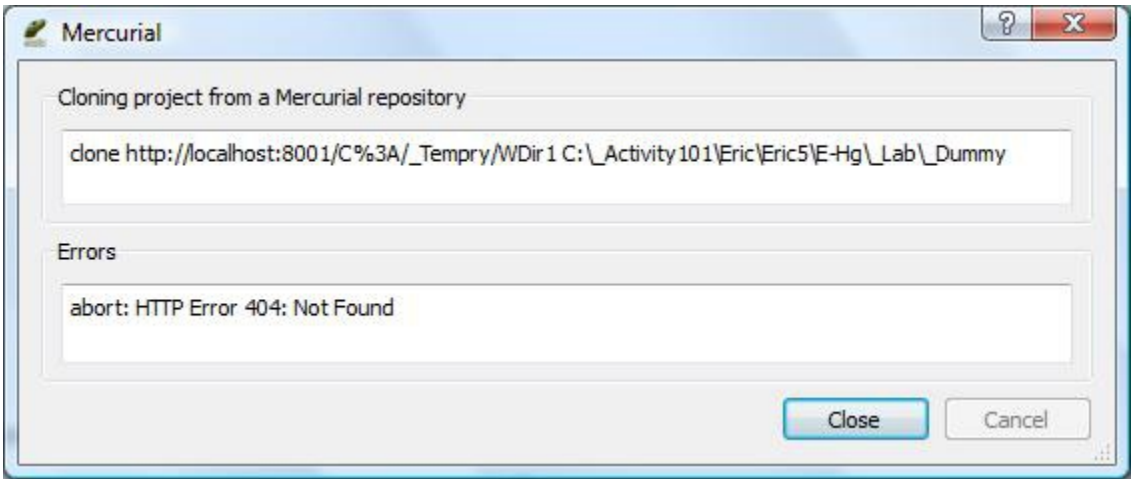


– –

<~> Indeed, trying to use this server with the E-Hg Clone command >> `New from Repository...`, we discovered that we had no idea how to actually enter the required URL, so to turn a promise into a fact.



Therefore, after a few such unsuccessful attempts with two Eric instances on the same computer:



<。?。> simply we gave it up.

– –

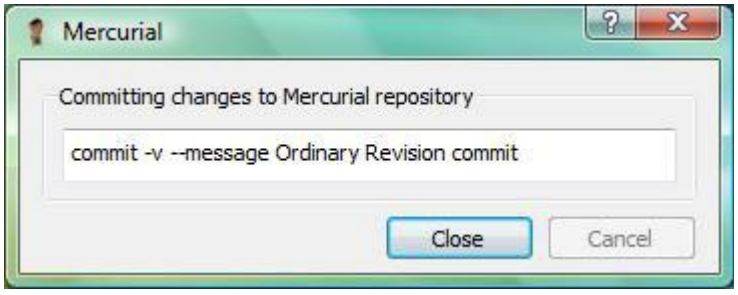
>> Command Options...

Designed for advanced users, wanting to assign extra options to the original Mercurial commands as called trough the E-Hg menu commands.

Remark

 Command under revision, as flawed and somehow unpractical. Could be possibly eliminated altogether.

Note that what is the exact text of the original Mercurial commands, in which E-Hg menu commands are automatically translated, is shown in conclusion of the E-Hg command execution, on such a box:



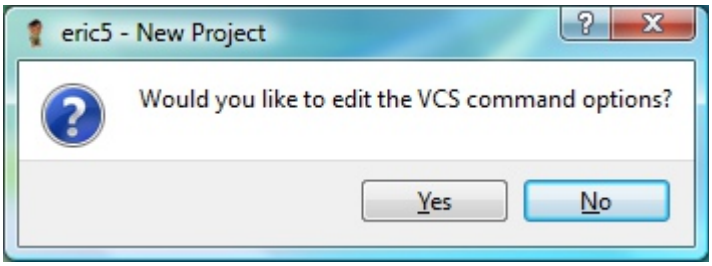
Where, by the way, the very Mercurial command text can be selected and copied, to be then possibly pasted as any text string. Besides, in this Report, such command text will be listed as part of the E-Hg command's description; typically this way:

Hg Execution

```
commit -v --amend --message ***
```

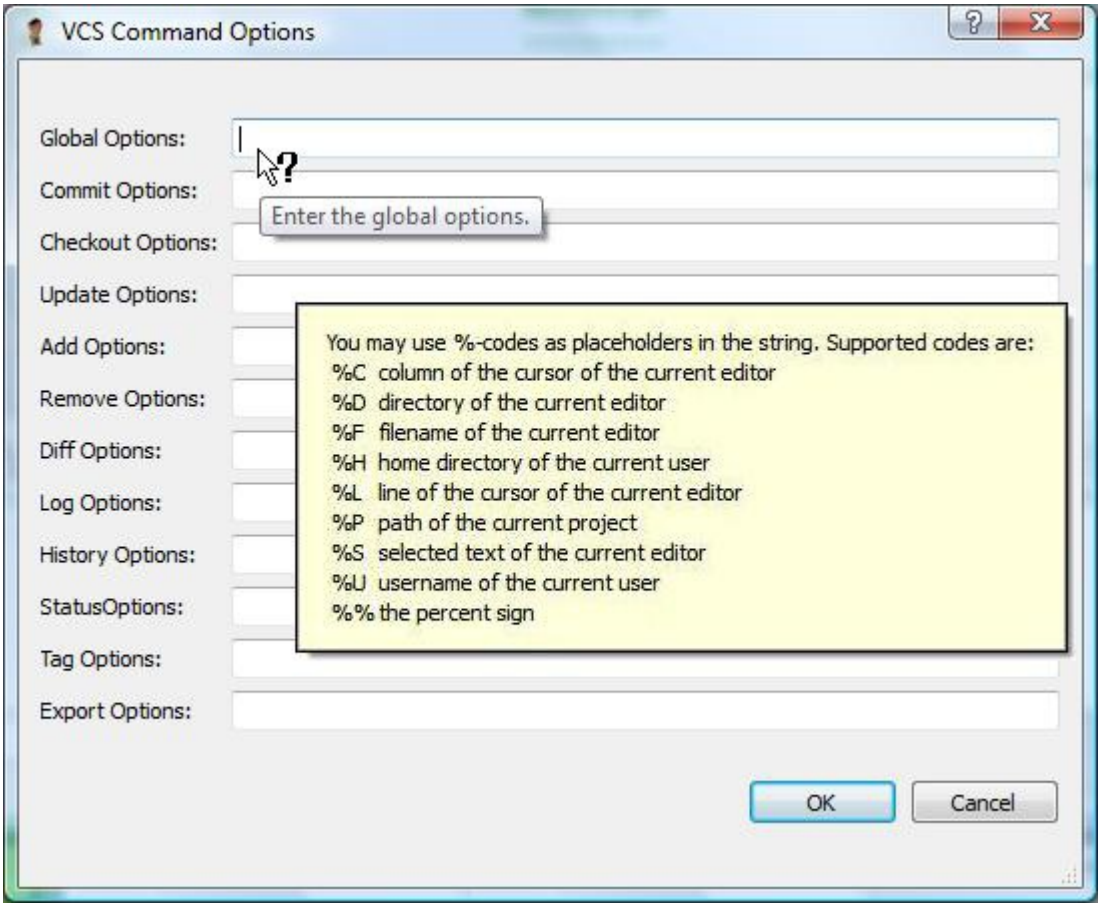
Remark

E-Hg GUI interface is incomparably friendlier than the original Mercurial's CLI, and fairly complete too. But circumstances may arise requiring a direct control over Mercurial actions, attainable entering specific options, as can be done with this `>> Command Options...` command, or making use of this possibility:



as offered by some E-Hg commands [*see:* `Project > New...` *versioned, or* `>> New from Repository...`]; or even entering direct Mercurial commands, via `>> Execute Command...` [*see*]. In all these cases an adequate knowledge of the Mercurial tool is, of course, required. A knowledge which is, of course, off the scope of this Report [*see section:* Scope of this Report].

– –



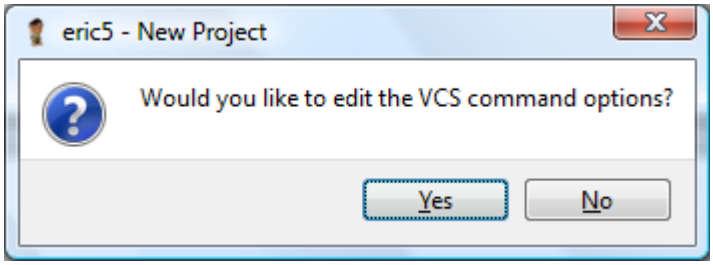
Control Box[64]

`Global` Such as: `--repository` for the repository root directory, or: `--help` to display help and exit; as in the official "Mercurial Command Reference", section Options [*ref.: at* Mercurial Package*, in* Appendix].

`Commit` For: `>> Commit Changes to Repository...` [*see*]

`Checkout` For: `>> New from Repository...` [*see*].
<~> It doesn't work with this very command, but only if entered after assenting to this proposal:

64 <~> In the following captions we've omitted the term "`Options`", as redundant.



`Update` For: `>> Update from Repository` [*see*]

`Add` For: `>> Add to Repository...` and
CM `Project-Viewer(^)> Add to Repository` [*see*]

`Remove` For: *CM* `Project-Viewer(^)> from Repository (and disk)` [*see*]
(<~> N.B.: Not for *CM* `Project-Viewer(^)> from Repository`).

`Diff` For: `>> Show Difference` [*see*]

`Log` For: `>> Show Log` and `>> Show Log Browser` [*see*].
<?> Flawed. If you test such an option: `--help`, all you get is this Error:
`No log available for '<WDir>'`

`History` <~> Not used.

`Status` For: `>> Show Status...` [*see*].
<?> Flawed. If you test such an option: `--help`, all you get is a command crash.

`Tag` For: `>> Tag in Repository...` [*see*]
<?> Flawed. If you test such an option: `--help`, simply it doesn't work.

`Export` For: `>> Patch Management > Export Patches...` [*see*].
<?> Flawed. If you test such an option: `--help`, simply it doesn't work.

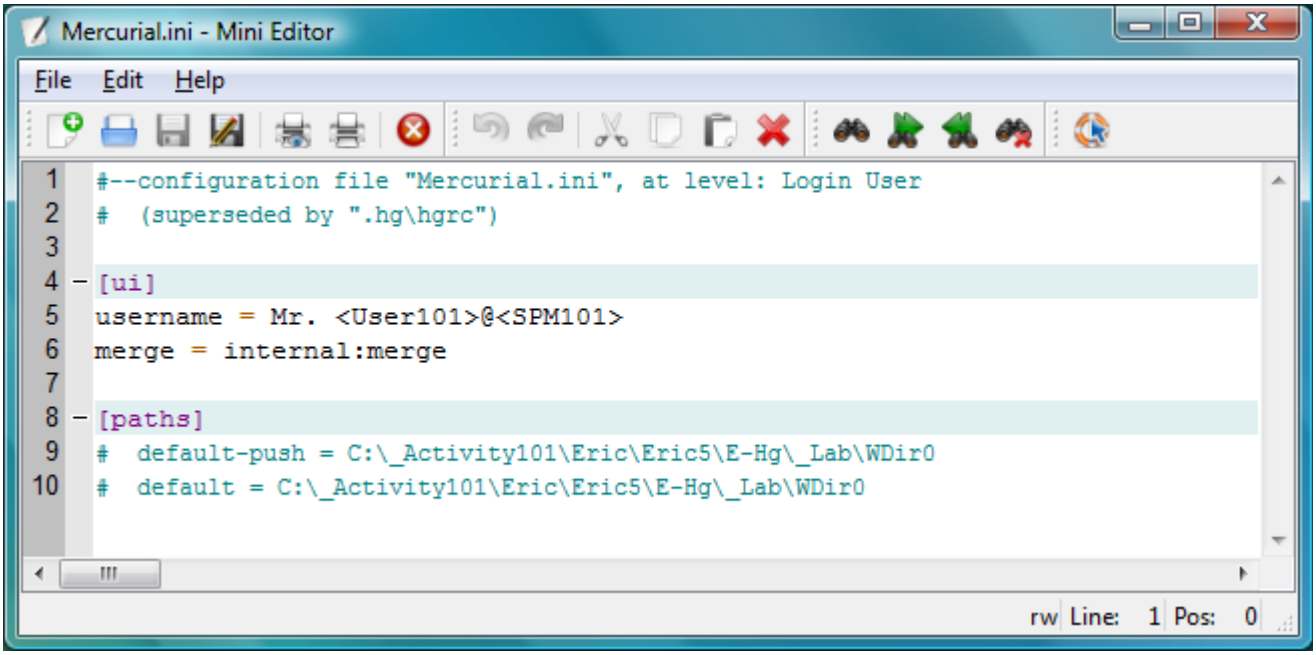
Remark

<~> Be aware that it is the user's responsibility to avoid any possible conflict between custom options and the automatic ones, lest such a result: `abort: Command: The system cannot find the file specified`

– –

>> Edit User Configuration...

Designed to inspect and edit the "`Mercurial.ini`" configuration file, valid at logon User level. Analogous to "`.hg\hgrc`", the other E-Hg configuration file, valid at each Repository level, in priority [*see: command* >> `Repository Administration > Edit Repository Configuration...`].



This file is located into such Windows Personal Folder as: `C:\Users\<User Name>`, where it is typically created automatically by Eric at the first versioned Project inception [*see:* `Project > New...`, *as described in this* Report], and initialized with just a Mercurial "`username`" parameter pattern[65].

<~> It should be then conveniently completed as hereafter described in the "Configuration Files" section [*see in:* Appendix].

Remark

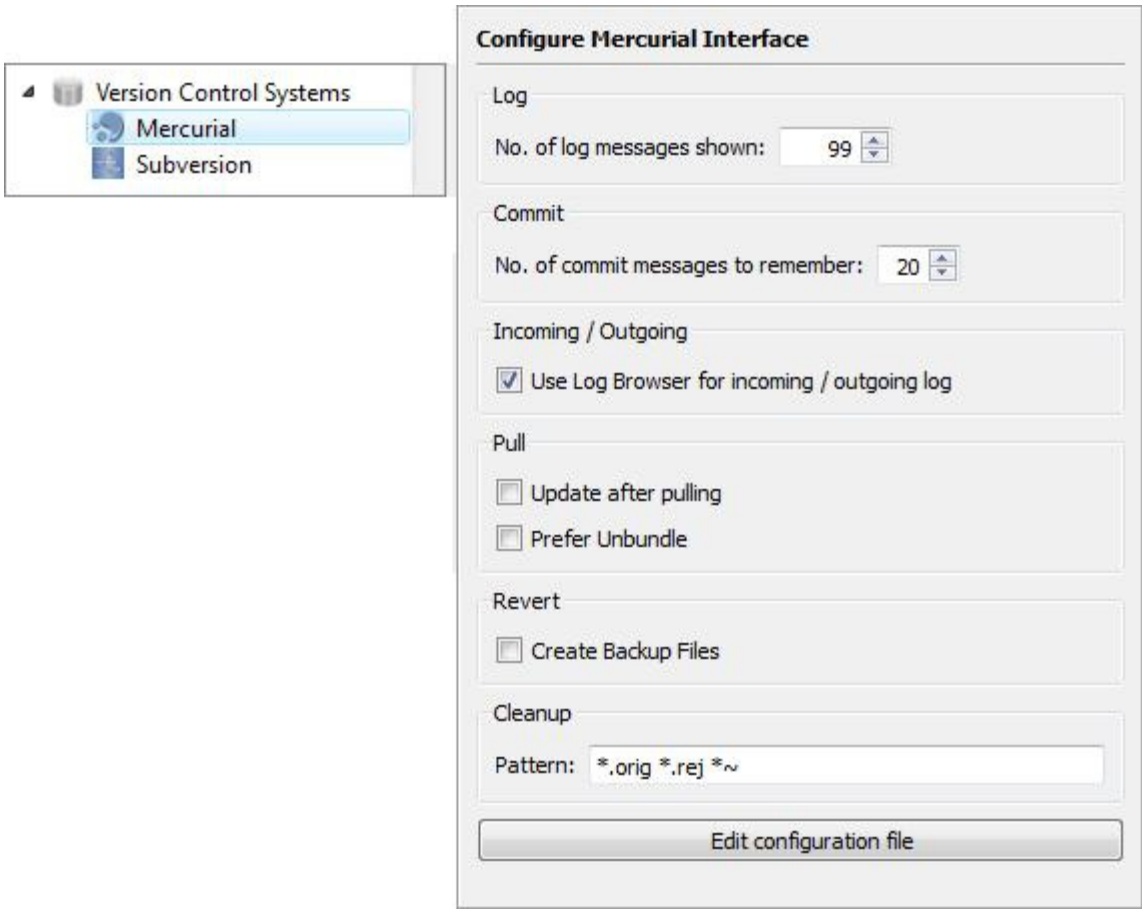
For a complete technical reference see standard "Mercurial Configuration Files" documentation [*ref.: at* Mercurial Package, *in* Appendix].

– –

65 <~> A pattern of the kind: `<User Name>@<Computer Name>`

>> Configure...

Designed to show the main Eric `Settings > Preferences...` window [*see*], just conveniently opened on the "`Configure Mercurial Interface`" control box.



Control Box

`No of log messages shown`
: Default limit for the number of messages to display with command: `>> Show Log` [*see*].

`No of commit messages to remember`
: Limit for the drop-down history list with command: `>> Commit Changes to Repository...` [*see*].

`Use Log Browser for incoming / outgoing log`
: Check-box to activate the same display tool as with `>> Show Log Browser` [*see*] for commands: `>> Show Incoming` / `Outgoing Log`, instead of the plain listing tool as with `>> Show Log` [*see*]. Default: checked

`Update after pulling`
: Check-box to force automatic update of the local Repository—that is: `>> Update from Repository` [*see*]—after a `>> Pull Changes` action [*see*]. Default: unchecked

`Prefere Unbundle`
: Check-box to activate a "`Unbundle`" execution mode for the `>> Pull Changes` command. Bundle execution mode is to avoid repeated downloading in case a "`bundle`" file [*cf. sub-menu:* `>> Changegroup Management`] has been already downloaded as the usual automatic consequence of a `>> Show Incoming Log` command execution [*see*]. Default: unchecked

`Create Backup Files`
: Check-box to enable creation of backup files in case of `>> Revert Changes` [*see*] <。?。> Is it really so? Is there then a way to re-do the changes? How? Further inquiry required. Default: unchecked

`Pattern`
: Space-separated list of "garbage" file types, for command: `>> Cleanup` [*see*]. Default: `*.orig *.rej *~`

`Edit configuration file`
: To run the same `Mercurial.ini` editor as with the command: `>> Edit User Configuration...` [*see*].

– –

Main Menu: **Settings**

Eric main menu command comprising just this one E-Hg command:

Settings > **Preferences...** – Version Control Systems > Mercurial

Equivalent to the `>> Configure...` command, here above [*see*].

– = –

[*CM*] E-Hg Context Menus

[*CM*] Context Menus vs. [*MM*] Main Menus

Scope, operative environment, of the E-Hg Context Menus is the single item—file or directory—of an Eric Project. Whereas scope, operative environment, of the E-Hg Main Menus [*see: MM*] is the entire Eric Project, considered as a whole, with no distinction between the single items belonging to the same Project. Eric Users must be well aware of such distinction, and purpose of this Report is to offer a valid technical reference for both these classes of menus.

List of E-Hg Context Menus

`Project-Viewer(^)`
: On the Left Pane, Tabs: `Sources`, ..., `Others`

`Mercurial Status(^)`
: On the form of the *MM* command `>> Show Status...`

`VCS Status Monitor "LED"(^)`
: At right of the bottom Information Bar  [66].

– –

Viewpoint

<~> With E-Hg the distinction between Main and Context Menus is even more evident than with Eric in general, as these two classes of Menus belong to different and, to some extent, not communicating scopes; sometimes requiring to be synchronized by User's explicit intervention. A fact that adds up to the relative independence existing also between the Eric and Mercurial scopes.

A relevant example is the addition of a new source file to a versioned Project, which should be told both to Eric and E-Hg, independently and acting at Context level; even though you have already declared "globally", at Main level, a Project as versioned and contained into a given working directory.

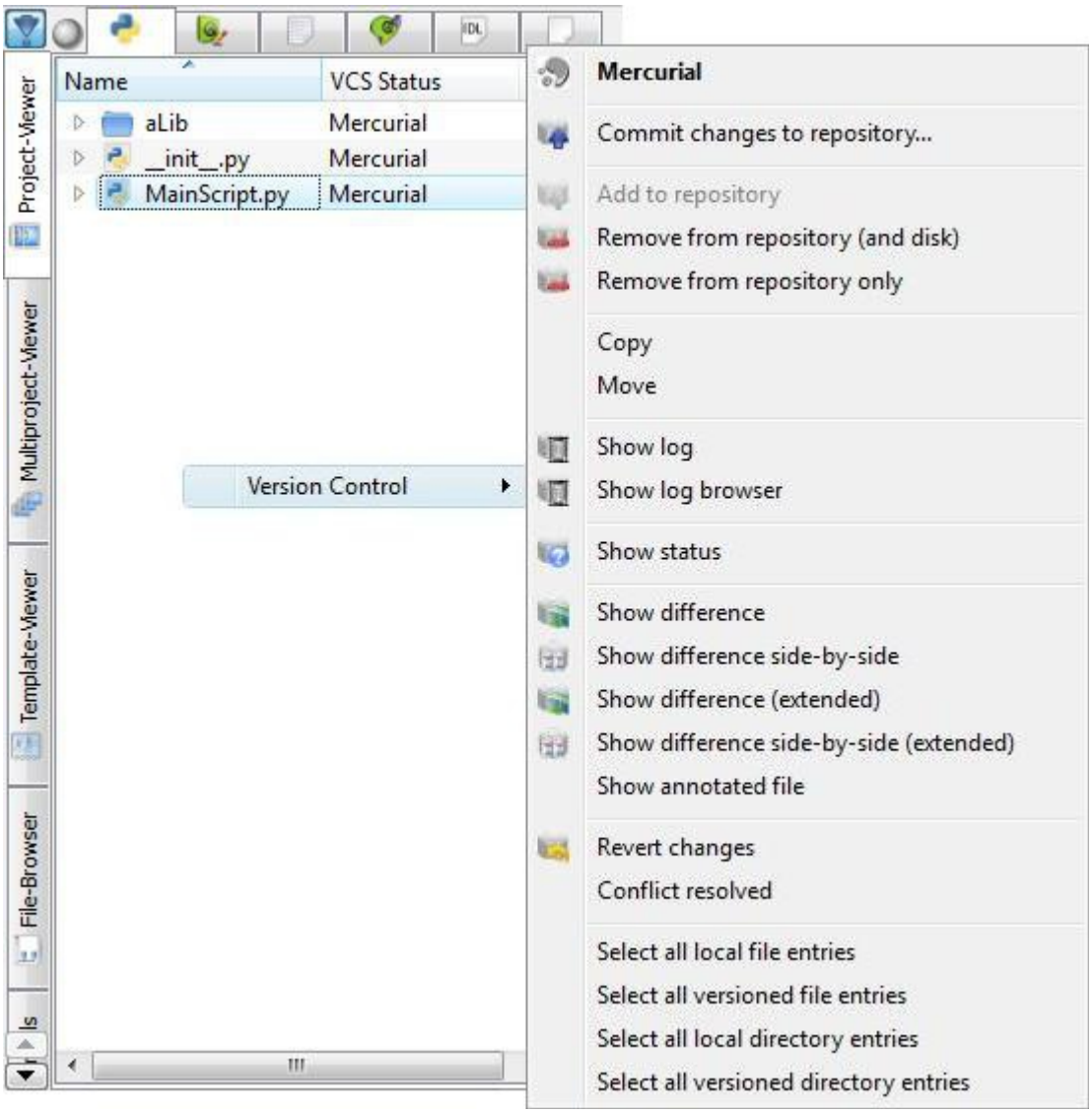
What we mean here is that a friendly versioning system should be totally "transparent" to the User, in the sense that once a projectual action had been correctly executed for Eric, it should result automatically known to the versioning system too, without any further need for the poor Mr. User to tell also to Mercurial what he has already told to Eric. Now, not being so, Mercurial can be perceived as an added burden, instead of as a relief. That's why it is our opinion that current Version Control Systems are still at a not-so-advanced stage.

-<>-

66 Color coded as in `Shift+F1` Help Hint [*see*], that is: Green=Ok, Red=Err, Yellow=Checking; Gray=Off.

Context Menu: **Project-Viewer(^) Version Control**

On the Left Pane, Tabs: `Sources`, ..., `Others`



Command List

Mercurial	`Commit Changes to Repository...`		[*see: MM*]
`Add to Repository`	`Remove from Repository (and Disk)`	`Remove from Repository Only`	
`Copy`	`Move`		
`Show Log`	`Show Log Browser`	`Show Status`	[*see: MM*]

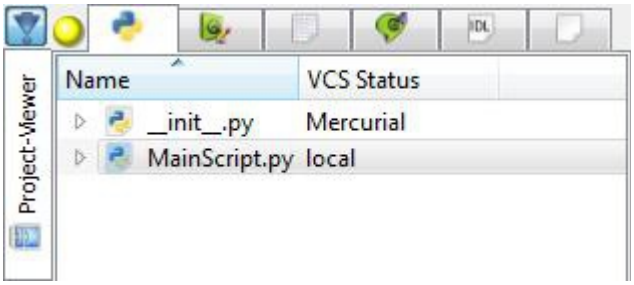
```
Show Difference                 Show Difference Side-By-Side[67]          [see: MM]
Show Difference (Extended)      Show Difference Side-By-Side (Extended)   [see: MM]
Show Annotated File
Revert Changes                  Conflicts Resolved[68]                    [see: MM]
Select All Local File Entries   Select All Versioned File Entries
Select All Local Directory Entries      Select All Versioned Directory Entries
```

- -

(^)> Add to Repository

Designed to inform E-Hg of any new file or directory possibly added to a versioned Project, so to have it versioned too. Commit then required, to complete.

<!> Action required because whenever a new file, of any kind, source or resource, is added to a versioned Project, it is not automatically assumed as version controlled too[69]. To that purpose you have got to explicitly say so by means of this context E-Hg command.



As an example, let's assume that, after having created, or cloned, an Eric Project declared as Mercurial versioned, you add, say, a "`MainScript.py`" module. On the `Project-Viewer` form this new module will be shown under the `VCS Status` column as "`local`", that is not "`Mercurial`"; as instead are the other preceding modules so automatically declared since the very beginning.

---

67 With "`Side-By-Side`" as for `Side-by-Side Diff` button of *MM* command >> `Show Status` [*see*].

68 Plural "`s`" here deliberately added, for uniformity.

69 <~> To our surprise, we should say. Indeed, it is our opinion that a fair Version Control System, once a Project has been declared versioned, should be as "transparent" as possible, that is operate silent and forgotten, without further bothering the IDE user, usually well busy enough doing his job.
In this specific case it is our opinion that only negative statements should be here considered, that is to say that a user should be given the possibility to declare when a given file, though added to a versioned Project, should NOT be version controlled. Whereas positive and deliberate additions shouldn't be so declared twice, both to Eric and also to E-Hg.
Other story altogether is that of files automatically created by the IDE, that is not deliberately and explicitly added by the user, such as the "`*.e4p`" file. Such files, being out of user's control, should be *automatically* declared versioned or not as a responsible IDE decision. Without discharging onto the user's shoulders also this burden.
<~> Of course this is only our opinion, clearly not shared by this VCS designer.

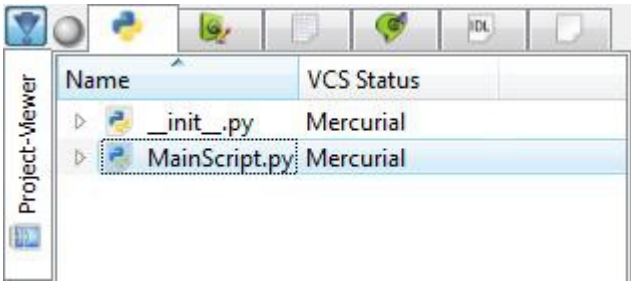
To execute, select the "`local`" file, right-click this context command, and you'll turn the status to "`Mercurial`", well ready for the next Commit action.

Hg Execution

```
add -v C:\_Activity101\Eric\Eric5\E-Hg\_Lab\WDir1\MainScript.py
adding MainScript.py
```

Then:

```
commit -v --message Added new MainScript.py file.
MainScript.py
vEPrj.e4p
committed changeset 1:2c3375363183
```

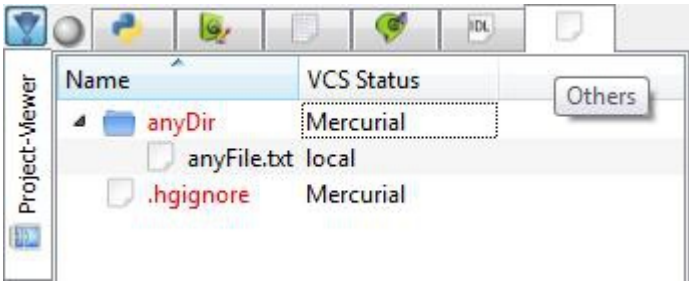


– –

(^)> Remove from Repository (and Disk)

(^)> Remove from Repository Only

Designed to perform precisely the inverse action of command `(^)> Add to Repository` [*see*], with action limited to the metadata Repository or also to the Working Directory. Commit then required, to complete.



All these commands equally applicable to sources as to any other dir & file.

– –

Hg Execution

```
forget -v C:\_Activity101\Eric\Eric5\E-Hg\_Lab\WDir1\MainScript.py
removing MainScript.py
```

– –

(^)> Copy

(^)> Move

Special E-Hg commands required to Copy / Move versioned files and directories, so to preserve their condition of versioned items. Commit then required, to complete.

Hg Execution

```
rename -v ***\xFile.txt ***\anyDir\xFileMoved.txt
moving xFile.txt to anyDir\xFileMoved.txt
```

– –

Viewpoint

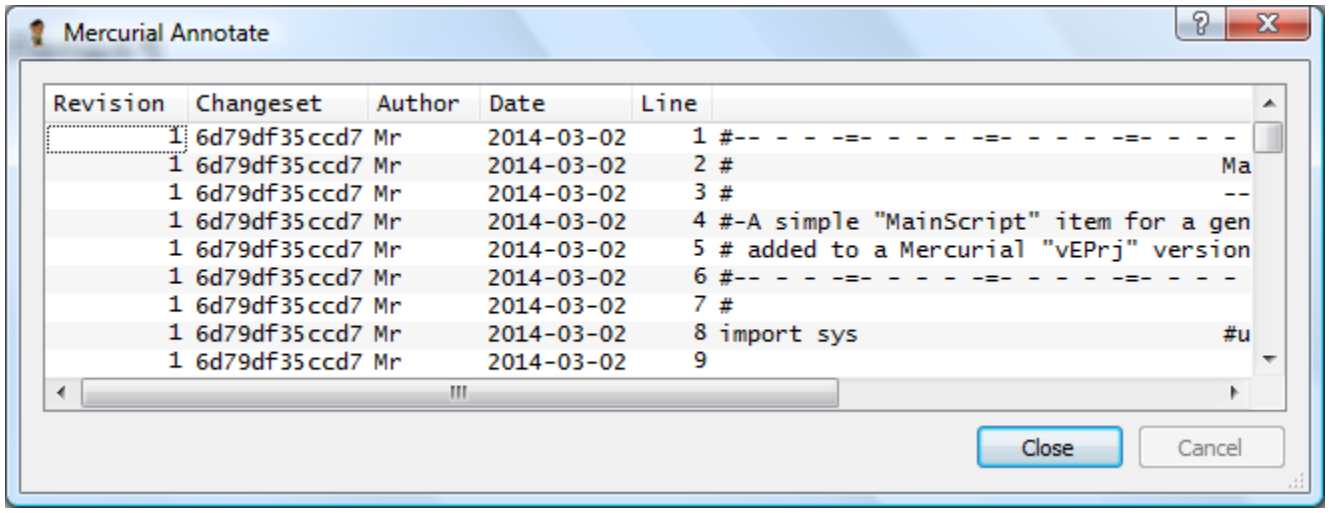
<~> Here too, as with `(^)> Add` and `(^)> Remove from Repository` [*see*], we see a curious special E-Hg duplication of such ordinary file management functions. Of course there is a reason for that, and the reason is that Eric and Mercurial are still distinct, almost non-communicating, not automatically synchronized entities, as we think they *should* be.

In other words, it is not enough to declare, once and for all, a given Eric Project as Version Controlled, the user is assumed anyhow responsible of *repeating this same declaration* at each new file management operation, but the sheer text editing. Of course this is a typical sign of immature technology, whatever clever and scandalized justifications you might be told.

– –

(^)> Show Annotated File

Designed to display a list of the selected file along with the usual annotations as with Mercurial "`annotate`" command [*cf.:* Mercurial Command Reference*, at* Mercurial Package*, in* Appendix].



<?> "`Author`" field here appears as incomplete.

- -

(^)> Select All Local File Entries

(^)> Select All Versioned File Entries

(^)> Select All Local Directory Entries

(^)> Select All Versioned Directory Entries

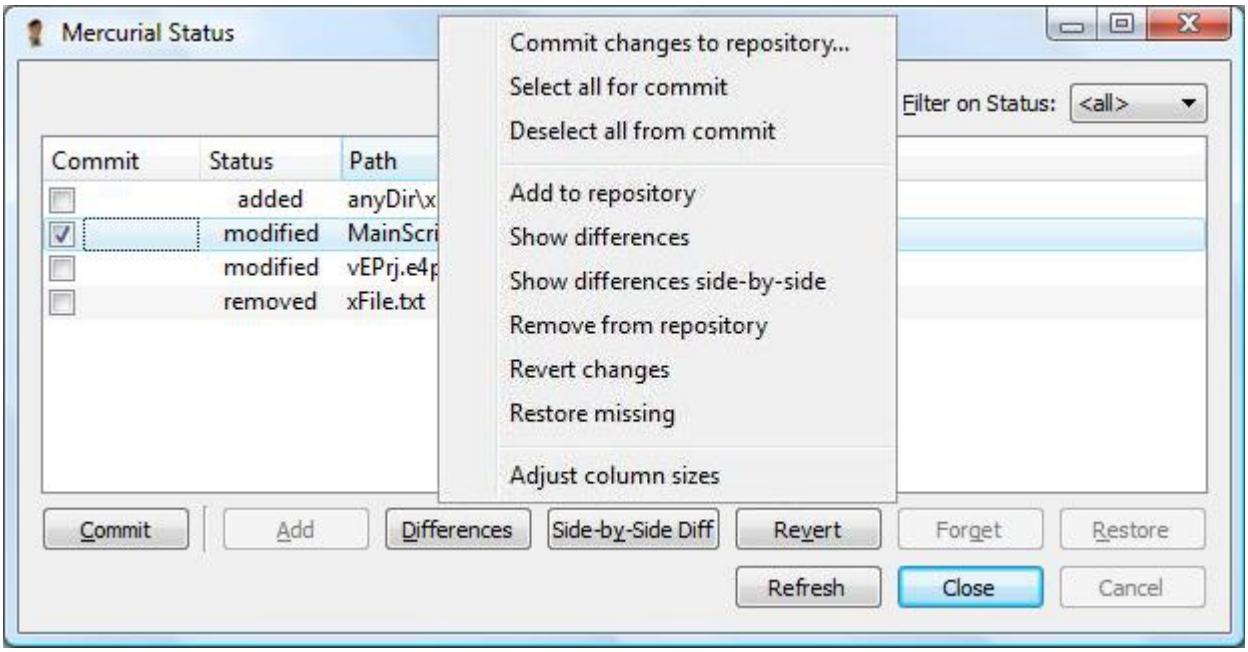
Designed to select Project files and directories according to their VCS Status as "`Mercurial`" (that is: versioned), or "`local`" (that is: not versioned).

A useful distinction for such commands as `(^)> Add to Repository` [*see*].

-<>-

Context Menu: **Mercurial Status(^)**

As available on the form of the *MM* command `>> Show Status...`



– –

Command List

`Commit Changes to Repository...` [*see: MM*]
`Select All for Commit` `Deselect All from Commit`
`Add to Repository` [*see: CM*]
`Show Difference`[70] `Show Difference Side-By-Side`[71] [*see: MM*]
`Remove from Repository` [*see: CM*]
`Revert Changes` `Restore Missing`[72] [*see: MM*]
`Adjust Column Sizes`

– –

70 Plural "s" here deliberately dropped, for uniformity.

71 With "`Side-By-Side`" as for `Side-by-Side Diff` button of *MM* command `>> Show Status` [*see*].

72 As for `Restore` button of *MM* command `>> Show Status` [*see*].

(^) Select All for Commit

(^) Deselect All from Commit

Designed to select / deselect all listed files to be then here possibly Committed.

By the way, note that this *CM* Commit action can be applied to distinct selected files, whereas the corresponding *MM* Commit action [*see:* *MM* `>> Commit Changes to Repository...`] necessarily operates on the entire Working Directory.

- -

(^) Adjust Column Sizes

Designed to reset the initial default column sizes, as with `Refresh` button.

-<>-

Context Menu: **VCS Status Monitor "LED"(^)**

As available at right of the bottom Information Bar.

Command List

`Check Status` `Set Interval...` `Switch On` `Switch Off`

- -

(^) Check Status

Designed to check the operating status of the VCS monitor thread, with "LED" color code: Green=Ok, Red=Err, Yellow=Checking; Gray=Off [*see:* `Shift+F1` *Help Hint, and tooltip*].

- -

(^) Set Interval...

(^) Switch On

(^) Switch Off

Designed to switch On / Off the status monitor, and set its automatic refresh time [*in sec, 0=Off*].

- = -

Appendix

In this section two completely different classes of topics, equally relevant.

About E-Hg: Setup and General Management
Working in the Internet

About this Report: Rules of Style

– –

Setup and General Management

Mercurial SCM may be seen and used as an optional tool embedded into the development environment Eric. This means that an Eric standard setup can be fully operative also without this tool, and also that such a tool can be used independently from Eric, and also in the same computer system.

That said, in this section we'll consider the setup and management actions required to get this Mercurial SCM tool fully integrated and usable within Eric [*update ref. as listed in:* Essentials].

– –

Mercurial Prerequisites

1/2] Eric

Base setup as described in the "Eric Python 3 (& 2) Integrated Development Environment Technical Report" [*see in: Eric Web Site*].

2/2] Python 2

Plus the ver. 2 of the interpreter Python, required by Mercurial, and accepted as an option by Eric too.

Where

Available for downloading from URL: `http://www.python.org/download/releases/<.?.>/`

What

Download file: `•Windows x86 MSI Installer (<.?.>) (sig)` [*or more recent*]

That is, among the various available items, the latest Python 2.*x.y*, Windows x86, MSI Installer — Windows binary only, no source included[73].

73 Then if you wander what the cryptic "`(sig)`" stands for, it's short for "*PGP signature*", as available at the bottom of the

How

Setup-run, with `Administrator` permission. All defaults accepted, and you'll get such a: "`\Python2x`"[74] directory installed [*see*].

All items on the resulting `Start > Python` button are of immediate comprehension, but perhaps the "`Module Docs`" item which is aimed at running "`pydoc`", a tool to manage Python modules documentation.

Then

Then a test-`Start` run of the "`IDLE (Python GUI)`" program, to verify that it's working all right.

Uninstall

Under the resulting system `Start > Python` button there is also a reassuring "`Uninstall Python`" command. We've prudentially executed it from here before each successive upgrading.

– –

Mercurial Package

Where

Available for downloading from URL: `http://mercurial.selenic.com/downloads/`

What

Download file: `Mercurial 2.8 MSI installer - x86 Windows`, that is[75]:
`mercurial-2.8.1-x86.msi` [*or more recent*]

How

Setup-run, with `Administrator` permission.

All defaults accepted, and you'll get a `C:\Program Files\Mercurial` setup directory. And a:
`Mercurial 2.8.0` folder into the system `Start` button, comprising the following doc set [*see*]:

Mercurial Command Reference
Mercurial Configuration Files
Mercurial Ignore Files
`Mercurial Web Site`, a link to: http://mercurial.selenic.com/wiki/

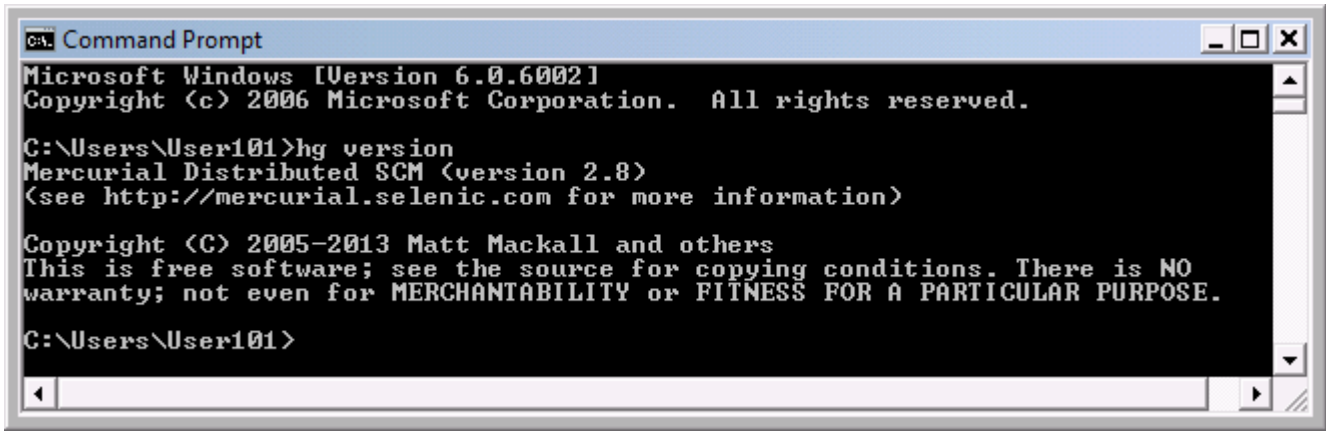
same web page, so to possibly verify the integrity of the downloaded file.

74 Actually a "`\Python27`", in our current case.

75 <~> Well preferable to the "`Inno`" setup edition, not to mention the source edition.

Then

Open a Command Prompt shell, and enter the Mercurial command: `hg version`



Just to verify that it's all up and running[76].

Uninstall

In the `Programs` section of the `Control Panel` you'll then see a reassuring `Mercurial 2.8.0 (x86)` item, there available to be uninstalled. As we've prudentially done before each successive upgrading.

– –

76 And also, by the way, to verify that you could well use it directly here, independently from Eric.

Configuration Files

Of all the various configuration systems and parameters available for Mercurial this section is to treat exclusively those which are relevant for E-Hg. For this subject as a whole see standard documentation "Mercurial Configuration Files" and "Configuring" section of "Mercurial Command Reference" [*ref.: at* Mercurial Package*, here above*].

E-Hg relies upon two distinct configuration files, which the user is supposed to conveniently set up as a normal house-keeping care before the actual versioning activity. They are:

`<User>\Mercurial.ini`
: Located into such Windows personal folder as: `C:\Users\<User Name>`, where it is typically created automatically by Eric at the first versioned Project inception, valid at system user's level; and:

`.hg\hgrc`
: Located into the "`\.hg`" Repository's sub-directory of each versioned Eric Project, valid at each Eric Project level, in priority.

Both files have the typical "`ini`" structure, being organized in: `[section]` and `key = value`. Both can contain the same parameters, but have different scopes: "`Mercurial.ini`" operates at user level, whereas each "`.hg\hgrc`" file operates at its own Project level; taking precedence over the former for possibly corresponding parameters. Therefore the choice where to put each parameter depends upon the scope assigned to it, that is: user-global or Project-local.

– –

Sections and Keys

`#`
: Comment prefix character

`[ui]`

`username = <Name String>`

`merge = internal:merge`
: To activate an internal simple algorithm for merging files. If any conflicts, it will leave markers in the partially merged file for subsequent user's manual intervention [*see:* `>> Merge Changes...`].

`[paths]`

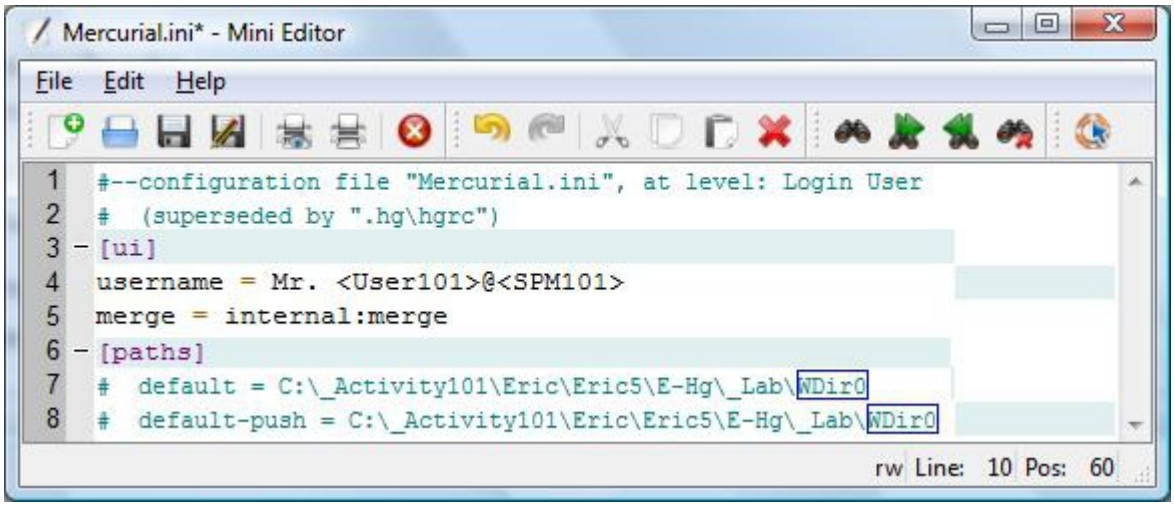
`default = <URL>`
: Sender of Pull actions [*see:* `>> Pull Changes`] and receiver of Push actions too, when "`default-push`" parameter not defined.

`default-push = <URL>`
: Receiver of Push actions [*see:* `>> Push Changes`], supersedes "`default`".

– –

File Editing

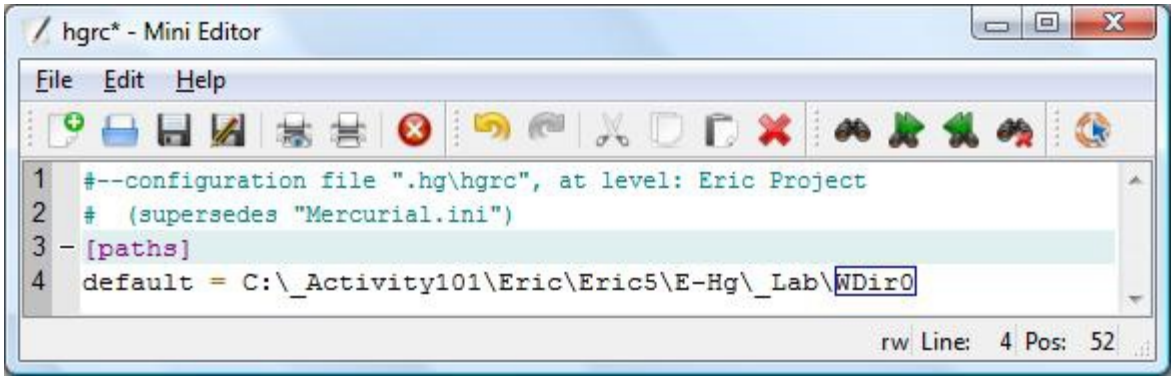
`<User>\Mercurial.ini`



It can be edited directly, or through command: `>> Configure...` or `Settings > Preferences...`, then: `- Version Control Systems > Mercurial, Configure Mercurial Interface, Edit configuration file` [*see*].
Or: `>> Edit User Configuration...` [*see*].

- -

`.hg\hgrc`

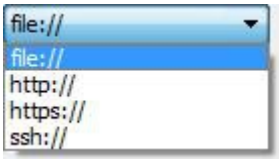


It can be edited directly, or through the command: `>> Repository Administration > Edit Repository Configuration...` [*see*].

-<>-

Working in the Internet

This the drop-down list of E-Hg connection protocols available for operating with different Working Directories, either local or remote.



It is in the very nature of a distributed VC System to operate locally on a Working Directory possibly copy-cloned from a remote master Working Directory [*see:* `>> New from Repository...`], with which to be then synchronized, from time to time, either in uploading or in downloading [*see:* `>> Push / Pull Changes`].

For most of this Report, all Working Directories are considered located in the same file system, with "`file://`" as the consequent connection protocol in use. In this section we'll consider a more general case, where the master and cloned Directories are not located in the same file system, but possibly also somewhere in the Internet. It is in this case that such protocol as "`https://`" comes handy; as shown in the example hereafter.

– –

Google Project Hosting

As a practical example, amongst the many Mercurial-compatible Internet host providers[77], we chose "Project Hosting on Google Code", as available at: `https://code.google.com/hosting/` [*see*].

Then, after a simple "Create a new project" process, we got a ready-to-use Mercurial Working Directory, located at such a URL as: `https://code.google.com/p/<myPrjName>`
in our case with such a "`<myPrjName>`" as: `hgveprj` (N.B.: lowercase only[78]).

That is, a brand-new and empty Mercurial Working Directory, ready available to be cloned locally with the E-Hg command: `Project > Version Control > New from Repository...`,

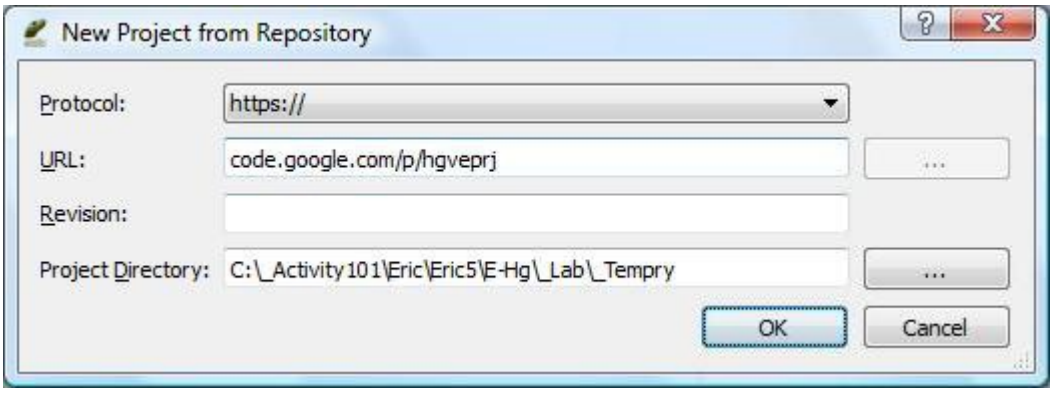
operating:

- from such remote URL: `https://code.google.com/p/hgveprj`
- to a local Working Directory

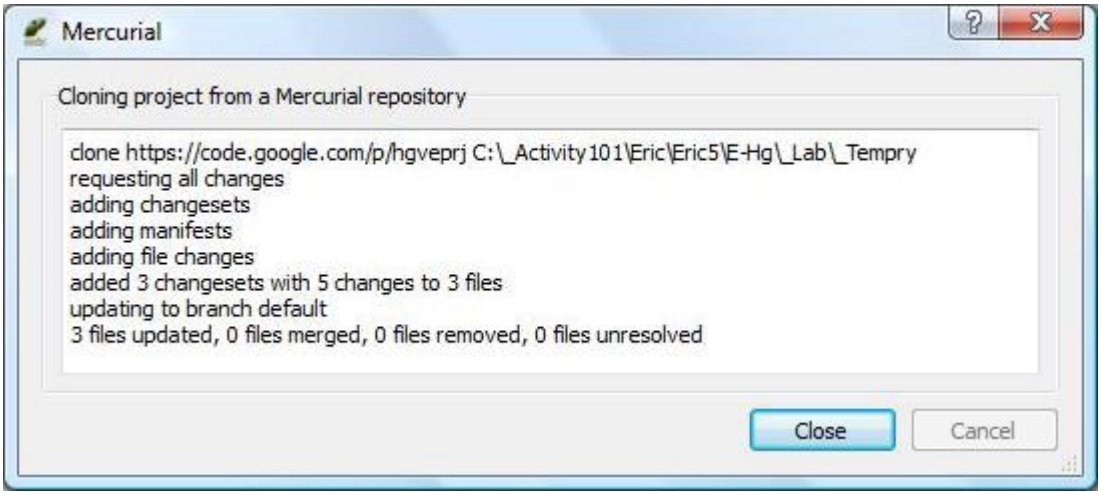
this way:

77 For some reference, see: Mercurial Hosting – Services that provide hosting of Mercurial repositories, at URL: `http://mercurial.selenic.com/wiki/MercurialHosting`

78 <~> As we realized trying first a: `HgvEPrj` (for: "Mercurial versioned Eric Project"), then not recognized at cloning.



With consequent *Execution Log*:



Remark

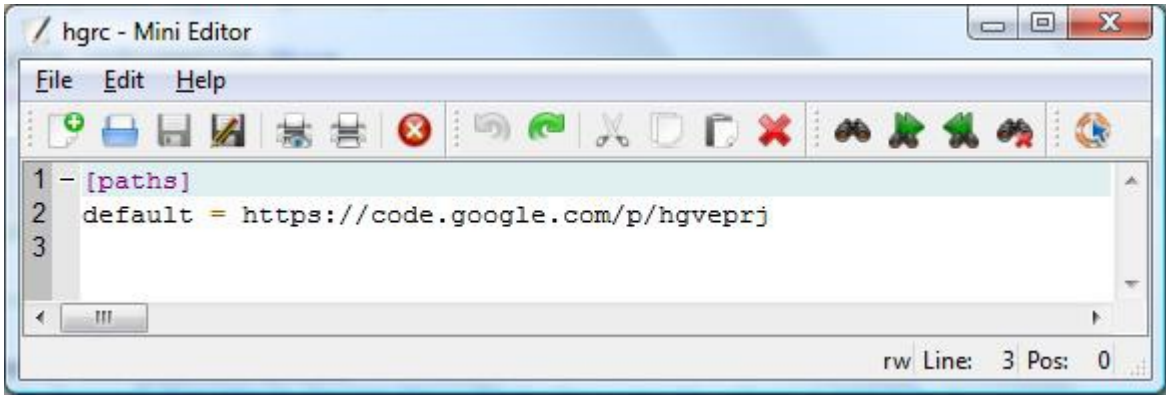
Here, though, there is a difference with the creation of a new versioned Eric Project [*cf.:* `Project > New...`]. Here you'll possibly have to add manually to the Repository such files as: `vEPrj.e4p`, `__init__.py` and `.hgignore` [*see:* `Project-Viewer(^)> Add to Repository`]; because, in this case, they are not added automatically.

– –

Synchronizing Local and Remote Working Directories

Once cloned locally, this Mercurial versioned Eric Project is available to be developed as usual, and then also, say, "Pushed" to the master location; provided you've got the required permission for it.

A look at the "`hgrc`" Repository configuration file [*see:* `>> Repository Administration > Edit Repository Configuration...`], and you'll see that the "`default`" path has been automatically set all right to the original master location URL. Fine.



Therefore, after some editing on your local Working Directory, you may feel encouraged to Push it almost as usual. But, unfortunately, things are not that simple, as will be hereafter explained.

– –

GoogleCode Password

First thing to be considered is that, to push your changes, even as a Project Owner, you'll need another password, different from that one you use with your Google Account [*see:* Settings, *at the* `GoogleCode.com` Password *section*]. And then, understandably enough, you'll have to authenticate with your Google Account and such newly generated GoogleCode Password. But then here there is a problem that we're glad to spare you.

Don't waist your time trying entering these two arguments when interactively asked to do so, because it doesn't work. At present[79] the only known way to authenticate when using E-Hg is to explicitly write them into the "`hgrc`" file, as part of the "`default`" path, this way:

---

79  We've been assured that Eric and Mercurial people are currently working on this issue, as it is a combined issue, sort of. Indeed—to make a long and tormented story short—as we've realized, if you go directly with Mercurial shell commands, the resulting behavior is still different.

`default = https://<username>@code.google.com/p/<project>`
so inserting your "`<username>`", to read-only;

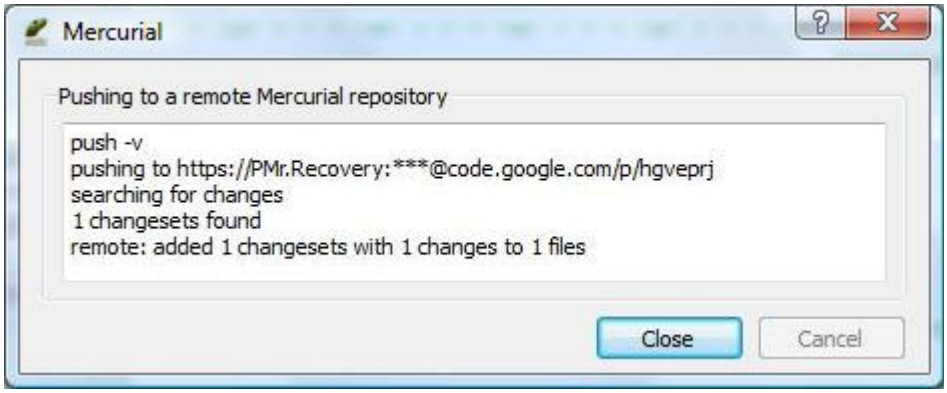
`default-push = https://<username>:<password>@code.google.com/p/<project>`
so inserting your "`<username>:<password>`", to read and write.

- -

As a result, your E-Hg commands aimed at your Google Code storage will be smoothly accepted, with such other notable execution

as: `>> Show Outgoing Log`

and: `>> Pull` / `Push Changes`



Remark

For what seen so far, a central E-Hg Working Directory located in the Internet can be Cloned locally, Pushed-up, Pulled-down, but not operated directly. In fact we don't know of any Eric way for opening it, as it is usual with local Working Directories.

-<>-

Rules of Style

Better to be aware that some expositive aspects of this Report are not at all casual, but deliberately adopted as Rules of Style.

– –

Sections and Revision

This whole Report is composed of distinct "*Sections*", delimited by such Begin-Of-Section (BOS):

S-PM 130500

and such End-Of-Section (EOS) lines:

– –

<!> With the "*yymmdd" date-revision code* on the BOS to be intended this way:

Each Section can be edited independently, with the most recent date that implicitly supersedes older ones. Sections possibly still unprovided with such date-revision code are supposed to inherit that one of the first BOS put on the cover page of the Report.

This explicit revision mechanism[80] is assumed to play a relevant role on the future life of this Report.

– –

Report's Glossary

Glossary of some particular terms as used throughout this Report.

Box	Generic term for such GUI structures as: window, form, area, pane, …
Control	Generic name for such elements as: button, selection check-box, option round-box, drop-down list, …
Control Box	Generic term for the GUI interaction structure of Eric commands, intended in the most broad sense.

– –

80 Besides the usual disclaimer telling that: "*The information in this document is subject to change without notice*" [*see:* Copyright Page].

Typographical Conventions

Prevalent typographical conventions here adopted to increase readability.

Arial	Font reserved for titles.
Capitalized	Usual text font—that is: Times New Roman, size 12pt—, with the initial capitalized not only for proper names, such as: Eric or Python, but also for terms to be intended in a specific technical sense, such as: Working Copy, Tag, Branch.
`Courier New`	For standard technical elements and references, such as: URL `http://sourceforge.net/projects/` File path `C:\Program Files\` Menu command `Settings > Preferences...`
`Italics`	Segment within a `Courier New` string, referring to a replaceable / variable item, such as a generic *`filename.ext`* into a standard path: `C:\Dir\Subdir\`*`fileneme.ext`* Sometimes inserted between angular braces to increase readability, as in: "<*`username`*>:<*`password`*>"
`>`	Separator for menu command path, e.g.: `Extras > Tools > Select Tool Group`
`(^)`	"Click" action symbol, for context pop-up menu, e.g.: `Project-Viewer(^) Version Control > ...`
<。>	Mark for a point deserving special attention — As: <!> Remarkable point <?> Questionable point <~> Perplexity, doubt, suggestion <。。。> Ellipsis, for topics still to be completed <。?。> Questionable point, further investigation required
	Icon for a "*Feature Under Revision*", and whose Tech. Report is currently left AsIs.

– –

Abbreviations

List of common abbreviations used in this Report, with particular reference to the Table Of Contents.

MM	Main Menu (intended as a Report section)
CM	Context Menu (intended as a Report section)
>>	`Project > Version Control >` (as E-Hg Main Menu command prefix)
(^)>	`(^) Version Control >` (as E-Hg Context Menu command prefix)
cf.	as "compare to" ("confer", Latin)
No	Number, integer; as in: Rev. No (without a trailing "." dot, in that comforted by the Oxford Advanced Dictionary)
Hex	Hexadecimal number; as in: Hex Id, Global Changeset Identifier

– –

– = –

Table of Contents

- = -

Off Report

S-PM 140524

A section to be considered as a trailer, "off" this Report. Just a few uncommitted brainstorming thoughts and considerations about the subject of versioning in general, and Mercurial in particular, trespassing a bit the initially declared "Scope of this Report" [see]; and not to be taken too seriously.

"In dubio, veritas" [81]

Well, "*veritas*" (truth) is certainly too ambitious, but a bit more understanding may be a reasonable goal. Here the "*dubio*" (doubt), the question, is: *When is it really worth using a Version Control System, and why?*

No ideological answer admitted, only sound and proved evidence [Thank you].

– –

Versioning Chaos

The availability of a versioning tool may act sometime as an encouragement to indulge into an ordered versioning chaos[82]. Designing implies to continuously take decisions, not to endlessly keep working on too different options; only because, relying upon a version control tool, you can always find your way back and forth into an intricate network of undecided developing paths.

That said, we dare this advice:

At start put your project under version control ok, then forget it until the release of your first proper version. Then again so, until the next version. In other words, not all single bit and twist occurring during a development process are worth to be permanently recorded ("Committed") with a Version Control System.

– –

Mission: (almost) Impossible

The effective versioning of any generic project assuming the form of generic disc files—that is: not exclusively s/w projects—is a task that we deem as almost impossible, impractical and, probably, useless too.

Indeed, the only successful case-histories of versioning we know about are exclusively related to software projects, intrinsically very simple to be version controlled, and also carried on by people

81 Another version: «*Dubitando ad veritatem pervenimus*» (By way of doubting we arrive at the truth), Cicero, Tusculanæ.

82 An example of "oxymoron", rhetoric figure of speech by which a locution produces an incongruous, seemingly self-contradictory effect.

particularly inclined to use such kind of tools.

<~> By the way, speaking of usability, we realized that the amount of commands and options comprising Mercurial reaches almost maniacal levels, requiring a severe and drastic selection—as rather appropriately and sensibly done by E-Hg—so to possibly turning it into a manageable tool.

Anyway, no other cases of success, outside the production of software, are known to us. And there too, effective versioning applies exclusively to the source text files, not certainly to any other resources that may legitimately be considered as part of a s/w project. A very delimited application scope indeed, and for rather good reasons. Hereafter some of them.

- The very concept of "difference", fundamental for solving conflicts when merging files, is elusive, almost impossible to bring under control in a practically manageable way when dealing with anything more than a plain, un-formatted, source text file. Try to use Mercurial for handling differences with any text formatted file, not to mention, say, graphic files, and you'll get such kind of result:

  ```
  diff -r 6346de817b7a -r 04b820e20150 DummyRep.odt
  Binary file DummyRep.odt has changed
  ```

  To say "`Binary … changed`" is equivalent to a declaration of impotence. Ok, it is conceivable to find out a proper tool for comparing—and managing, and fixing—differences between files of all different types. Conceivable yes[83], feasible in general, we doubt.

- But, even with a plain software project, versioning cannot go beyond the mere text source files. Just a change, say, on the properties of an Eric Project will possibly get the poor designer to deal with the internal structure of a XML "`*.e4p`" file. Indeed, for instance, in case you change such a property as the name of the Main Module of your Eric Project, you'll indirectly induce a change on the related Project's "`*.e4p`" file, out of your direct control; but that you may be anyhow called to adjust in case of a likely merge conflict with other Kin Projects connected in network. Rather tricky a situation.

- A clear cut between the very Project and its versioning metadata is almost impossible. Indeed, for instance, in case of an E-Hg versioned Project, you'll find, mixed up with your Project files also such files as: `.hgignore` and `.hgsub`; outside the dedicated "`\.hg`" metadata subdirectory. Inevitable, and for very good reasons. That is to say that the perfect transparency of a versioning system is a myth.

- Speaking of "transparency", here another reason of perplexity.
  Once a designer has declared a project as versioned, he should be through with that, having the right of forgetting all about versioning, at least between one Commit and the other. Having the right of going on with his business as a designer, without any further commitment towards the VC System. Indeed, it is the VC System assumed to be at the designer's service, not the other way round. It is in this sense that we'd expect a VC System to be really "transparent".

83 Examples: MS Word "diff" tool available in TortoiseHg, and a JSON (JavaScript Object Notation)-based custom tool to "diff" MS Access database.

But now things are certainly not so as, for instance, if in a versioned project of yours you happen to rename, or add, or erase a file or a directory, then it is up to you to inform of such action the VC System too.

<~> Well, whatever clever justification adduced for such a commitment, we deem it simply as a sign of immature technology.

- We heard of a versioning experiment dealing with a citizen legislative project[84]. Working great, according to the organizer (a s/w expert); but refused by the actual operators (all but s/w nerds) as impractical, unusable.

As a conclusion: We'd love to hear of some experiment or some project denying what here above.

– –

E-Hg Versioning in the Internet

We have happily tested E-Hg also in the Internet, making use of a Google public storage, more precisely a "GoogleCode Project Hosting", Mercurial compatible. Fine. But still a serious perplexity have we matured about the sheer fact that a special Mercurial-compatible storage should be used, instead of, simply and plainly, "any" public storage. What here we mean with "*any*", deserves an explanation.

<~> It's our opinion that any storage—also in the Internet—capable of hosting a given project, should also be capable of hosting the same project if possibly versioned; exactly as it can be done with any usual file system. Indeed, why not? Whatever subtle reasons[85] against such a plain & simple request, there is clearly a reason to assume that versioning technology is currently still very specialized, at best; or still very immature, at worst.

– –

"Off-Line" Distributed Versioning

With a distributed VC System, such as Mercurial, an "Off-Line" distributed versioning strategy could be used for overcoming the storage compatibility barrier mentioned in the above section. Of course it is just a work-around, not an organic solution, anyway worth knowing.

In a network of autonomous file systems, loosely connected in the Internet, if you renounce such a versioning functions as the on-line Cloning and the Pull / Push of changes, you may still synchronize Kin Working Directories via, say, e-mail, or so. This way "any" storage capable of hosting a project of yours, along with the related versioning metadata, will do; provided an alternative way of data

84 Contact info available upon request.

85 As the presence of a Mercurial ("`hgweb`") server implementing such Clone and Pull / Push services.

interchange is available, such as e-mail attachments or, even simpler, a remote virtual disk[86] mapped locally. Not so uncommon way of overcoming the mentioned compatibility barrier; still a work-around though, but it works fine.

– –

Collaboration Models

There is not a single best way for using a version control tool, as it's also matter of circumstance and preference. Reasons:

- Versioning is there to permit concurrent activity of different contributors, possibly working independently, in different geographical locations and with different timings, for developing and maintaining complex software projects.
- Nevertheless versioning techniques can be profitably used also in case of a single user, that is in an operative condition without the concurrent activity of distinct and independent contributors.
- Mercurial is a true "distributed" system, permitting each contributor to work autonomously and in parallel with other contributors. That's true, but that's not permanent; that's until you are to reach a conclusion, which obviously cannot be but a centralized action.
- In conclusion:
  How different distributed Projects can be blended into a single Master Project is matter of choice and circumstances, not necessarily Mercurial-dependent. Different Collaboration Models can be adopted, from separated e-mail shipment of each distributed Project to a Central Master Designer, who will manage the merging process; to the direct uploading executed by each contributor enjoying free access up to a Master Storage structurally build compatible with the specific version control system in use. With all possibly imaginable intermediate conditions.

– –

Viewpoint

<~> A deep perplexity though, about some of the cited interchanging methods, particularly those conceived for distributing distinct slices of a s/w project. A project, to be efficiently developed in a collaborative environment, should be structurally organized into modular sections, easy to develop and interchange, independently from the adoption of a Version Control System, which is anyhow there for other reasons.

Many experienced project managers we met matured and expressed the conviction that a Version Control System cannot be a substitute for rational and efficient work organization, and should not be

86 E.g. see: `www.jungledisk.com`

used to harness the chaos. Chaos should be preventively avoided, not encouraged. Therefore they use such Systems in general, and Mercurial in particular, for what they are meant to be used, that is for the Version Control of Projects, not for distributing source fragments, or single modules. Action, this one, to be by far more conveniently performed as a normal source editing activity, as usually carried on between distinct Commits of distinct Revisions, without unnatural and unnecessary involvement of special versioning tools, in this case more a burden than a relief.

- = -