Test of the hg evolve extension for easier upstreaming

Arne Babenhauserheide

January 12, 2013

Contents

1 Rationale

Currently I rework my code extensively before I push it into upstream SVN. Some of that is inconvenient and it would be nicer to have easy to use refactoring tools.

hg evolve might offer that.

This test uses the mutable-hg extension in revision c70a1091e0d8 (24 changesets after 2.1.0). It will likely be obsolete, soon, since mutable-hg is currently moved into Mercurial

core by Pierre-Yves David, its main developer. I hope it will be useful for you, to assess the future possibilities of Mercurial today.[1]

2 Tests

```
# Tests for refactoring history with the evolve extension
export LANG=C # to get rid of localized strings
export PS1="$ "
rm -r testmy testother testpublic
```

2.1 Init

Initialize the repos I need for the test.

We have one public repo and 2 nonpublishing repos.

```
# Initialize the test repo
hg init testpublic # a public repo
hg init testmy # my repo
hg init testother # other repo
# make the two private repos nonpublishing
for i in my other
  do echo "[ui]
username = $i
[phases]
publish = False" > test${i}/.hg/hgrc
done
```

note: it would be nice if we could just specify nonpublishing with the init command.

2.2 Prepare

Prepare the content of the repos.

```
cd testmy
echo "Hello World" > hello.txt
hg ci -Am "Hello World"
hg log -G
cd ..
```

```
 $ adding hello.txt
 @  changeset:   0:295012c497d2
```

[1]: This is not (only) a pun on "obsolete", the functionality at the core of evolve which allows safe, collaborative history rewriting </small>

```
    tag:         tip
    user:        my
    date:        Sat Jan 12 00:30:57 2013 +0100
    summary:     Hello World
```

2.3 Amend

Add a bad change and amend it.

```
cd testmy
sed -i s/World/Evoluton/ hello.txt
hg ci -m "Hello Evolution"
echo
hg log -G
cat hello.txt
# FIX this up
sed -i s/Evoluton/Evolution/ hello.txt
hg amend -m "Hello Evolution" # pass the message explicitely again to avoid having the edi
echo
hg log -G
cd ..


$ $
@  changeset:   1:f7e322874794
|  tag:         tip
|  user:        my
|  date:        Sat Jan 12 00:30:59 2013 +0100
|  summary:     Hello Evolution
|
o  changeset:   0:295012c497d2
   user:        my
   date:        Sat Jan 12 00:30:57 2013 +0100
   summary:     Hello World
Hello Evoluton
$ $ $
@  changeset:   3:3469b9a99206
|  tag:         tip
|  parent:      0:295012c497d2
|  user:        my
|  date:        Sat Jan 12 00:30:59 2013 +0100
|  summary:     Hello Evolution
|
o  changeset:   0:295012c497d2
```

```
user:        my
date:        Sat Jan 12 00:30:57 2013 +0100
summary:     Hello World
```

2.4 ... together

Add a bad change. Followed by a good change. Pull both into another repo and amend it. Do a good change in the other repo. Then amend the bad change in the original repo, pull it into the other and evolve.

2.4.1 Setup

Now we change the format to planning a roleplaying session to have a more complex task. We want to present this as coherent story on how to plan a story, so we want clean history.

First I do my own change.

```
cd testmy
# Now we add the bad change
echo "Wishes:
- The Solek wants Action
- The Judicator wants Action

" >> plan.txt
hg ci -Am "What the players want"
# show what we did
echo
hg log -G -r tip
# and the good change
echo "Places:
- The village
- The researchers cave
" >> plan.txt
hg ci -m "The places"
echo
hg log -G -r 1:
cd ..


$ > > > > $ adding plan.txt
$
@  changeset:   4:abd578b7cb03
|  tag:         tip
|  user:        my
|  date:        Sat Jan 12 00:31:01 2013 +0100
```

```
|  summary:     What the players want
|
$ > > > $ $
@  changeset:   5:60e12b82e24e
|  tag:         tip
|  user:        my
|  date:        Sat Jan 12 00:31:02 2013 +0100
|  summary:     The places
|
o  changeset:   4:abd578b7cb03
|  user:        my
|  date:        Sat Jan 12 00:31:01 2013 +0100
|  summary:     What the players want
|
o  changeset:   3:3469b9a99206
|  parent:      0:295012c497d2
|  user:        my
|  date:        Sat Jan 12 00:30:59 2013 +0100
|  summary:     Hello Evolution
|
```

Now my file contains the wishes of the players as well as the places.

We pull the changes into the repo of another gamemaster with whom we plan this game.

```
hg -R testother pull -u testmy
hg -R testother log -G -r 1:
```

```
pulling from testmy
requesting all changes
adding changesets
adding manifests
adding file changes
added 4 changesets with 4 changes to 2 files
2 files updated, 0 files merged, 0 files removed, 0 files unresolved
@  changeset:   3:60e12b82e24e
|  tag:         tip
|  user:        my
|  date:        Sat Jan 12 00:31:02 2013 +0100
|  summary:     The places
|
o  changeset:   2:abd578b7cb03
|  user:        my
|  date:        Sat Jan 12 00:31:01 2013 +0100
```

```
|  summary:     What the players want
|
o  changeset:   1:3469b9a99206
|  user:        my
|  date:        Sat Jan 12 00:30:59 2013 +0100
|  summary:     Hello Evolution
|
```

note: the revisions numbers are different because the other repo only gets those obsolete revisions which are ancestors to non-obsolete revisions. That way evolve slowly cleans out obsolete revisions from the history without breaking repositories which already have them (but giving them a clear and easy path for evolution).

He then adds the important people:

```
cd testother
echo "People:
- The Lost
- The Specter
" >> plan.txt
hg ci -m "The people"
echo
hg log -G -r 1:
cd ..
```

```
> > > $ $
@  changeset:   4:b3b9cc1d8d18
|  tag:         tip
|  user:        other
|  date:        Sat Jan 12 00:31:06 2013 +0100
|  summary:     The people
|
o  changeset:   3:60e12b82e24e
|  user:        my
|  date:        Sat Jan 12 00:31:02 2013 +0100
|  summary:     The places
|
o  changeset:   2:abd578b7cb03
|  user:        my
|  date:        Sat Jan 12 00:31:01 2013 +0100
|  summary:     What the players want
|
o  changeset:   1:3469b9a99206
|  user:        my
```

```
|  date:        Sat Jan 12 00:30:59 2013 +0100
|  summary:     Hello Evolution
|
```

2.4.2 Fix my side

And I realize too late, that my estimate of the wishes of the players was wrong. So I simply amend it.

```
cd testmy
hg up -r -2
sed -i "s/The Solek wants Action/The Solek wants emotionally intense situations/" plan.txt
hg amend -m "The wishes of the players"
hg log -G -r 1:
cd ..
```

```
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
$ 1 new unstable changesets
@  changeset:   7:14081e12386d
|  tag:         tip
|  parent:      3:3469b9a99206
|  user:        my
|  date:        Sat Jan 12 00:31:08 2013 +0100
|  summary:     The wishes of the players
|
| o  changeset:   5:60e12b82e24e
| |  user:        my
| |  date:        Sat Jan 12 00:31:02 2013 +0100
| |  summary:     The places
| |
| x  changeset:   4:abd578b7cb03
|/   user:        my
|    date:        Sat Jan 12 00:31:01 2013 +0100
|    summary:     What the players want
|
o  changeset:   3:3469b9a99206
|  parent:      0:295012c497d2
|  user:        my
|  date:        Sat Jan 12 00:30:59 2013 +0100
|  summary:     Hello Evolution
|
```

Now I amended my commit, but my history does not look good, yet. Actually it looks evil, since I have 2 heads, which is not so nice. The changeset under which we just pulled

away the bad change has become unstable, because its ancestor has been obsoleted, so it has no stable foothold anymore. In other DVCSs, this means that we as users have to find out what was changed and fix it ourselves.

Changeset evolution allows us to evolve our repository to get rid of dependencies on obsolete changes.

```
cd testmy
hg evolve
hg log -G -r 1:
cd ..
```

```
move:[5] The places
atop:[7] The wishes of the players
merging plan.txt
@  changeset:   8:ba84b8c0e6a2
|  tag:         tip
|  user:        my
|  date:        Sat Jan 12 00:31:02 2013 +0100
|  summary:     The places
|
o  changeset:   7:14081e12386d
|  parent:      3:3469b9a99206
|  user:        my
|  date:        Sat Jan 12 00:31:08 2013 +0100
|  summary:     The wishes of the players
|
o  changeset:   3:3469b9a99206
|  parent:      0:295012c497d2
|  user:        my
|  date:        Sat Jan 12 00:30:59 2013 +0100
|  summary:     Hello Evolution
|
```

Now I have nice looking history without any hassle - and without having to resort to low-level commands.

2.4.3 Be a nice neighbor

But I rewrote history. What happens if my collegue pulls this?

```
hg -R testother pull testmy
hg -R testother log -G
```

```
pulling from testmy
```

```
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files (+1 heads)
(run 'hg heads' to see heads, 'hg merge' to merge)
1 new unstable changesets
o  changeset:   6:ba84b8c0e6a2
|  tag:         tip
|  user:        my
|  date:        Sat Jan 12 00:31:02 2013 +0100
|  summary:     The places
|
o  changeset:   5:14081e12386d
|  parent:      1:3469b9a99206
|  user:        my
|  date:        Sat Jan 12 00:31:08 2013 +0100
|  summary:     The wishes of the players
|
| @  changeset:   4:b3b9cc1d8d18
| |  user:        other
| |  date:        Sat Jan 12 00:31:06 2013 +0100
| |  summary:     The people
| |
| x  changeset:   3:60e12b82e24e
| |  user:        my
| |  date:        Sat Jan 12 00:31:02 2013 +0100
| |  summary:     The places
| |
| x  changeset:   2:abd578b7cb03
|/   user:        my
|    date:        Sat Jan 12 00:31:01 2013 +0100
|    summary:     What the players want
|
o  changeset:   1:3469b9a99206
|  user:        my
|  date:        Sat Jan 12 00:30:59 2013 +0100
|  summary:     Hello Evolution
|
o  changeset:   0:295012c497d2
   user:        my
   date:        Sat Jan 12 00:30:57 2013 +0100
   summary:     Hello World
```

As you can see, he is told that his changes became unstable, since they depend on obsolete history. No need to panic: He can just evolve his repo to be state of the art again.

But the unstable change is the current working directory, so evolve does not change it. Instead it tells us, that we might want to call it with ‘–any‘. And as it is the case with most hints in hg, that is actually the case.

```
hg -R testother evolve

 nothing to evolve here
 (1 troubled changesets, do you want --any ?)
```

note: that message might be a candidate for cleanup.

```
hg -R testother evolve --any
hg -R testother log -G -r 1:

move:[4] The people
atop:[6] The places
merging plan.txt
@  changeset:   7:ca58acfa3af5
|  tag:         tip
|  user:        other
|  date:        Sat Jan 12 00:31:06 2013 +0100
|  summary:     The people
|
o  changeset:   6:ba84b8c0e6a2
|  user:        my
|  date:        Sat Jan 12 00:31:02 2013 +0100
|  summary:     The places
|
o  changeset:   5:14081e12386d
|  parent:      1:3469b9a99206
|  user:        my
|  date:        Sat Jan 12 00:31:08 2013 +0100
|  summary:     The wishes of the players
|
o  changeset:   1:3469b9a99206
|  user:        my
|  date:        Sat Jan 12 00:30:59 2013 +0100
|  summary:     Hello Evolution
|
```

And as you can see, everything looks nice again.

2.5 . . . safely

Publishing the changes into a public repo makes them immutable.

Now imagine, that my co-gamemaster publishes his work. Mercurial will then store that his changes were published and warn us, if we try to change them.

```
cd testother
hg up > /dev/null
echo "current phase"
hg phase .
hg push ../testpublic
echo "phase after publishing"
hg phase .
cd ..
```

```
$ current phase
7: draft
pushing to ../testpublic
searching for changes
adding changesets
adding manifests
adding file changes
added 5 changesets with 5 changes to 2 files
phase after publishing
7: public
```

Now trying to amend history will fail (except if we first change the phase to draft with ‘hg phase –force –draft .‘).

```
cd testother
hg amend -m "change published history"
# change to draft
hg phase -fd .
hg phase .
# now we could amend, but that would defeat the point of this section, so we go to public
hg phase -p .
cd ..
```

```
abort: can not rewrite immutable changeset ca58acfa3af5
$ $ 7: draft
```

Once I pull from that repo, the revisions which are in there will also switch phase to public in my repo.

So by pushing the changes into a publishing repo, you can get the Mercurial of all contributors to track which revisions are safe to change - and which are not. An alternative is using ‘hg phase -p REV‘.

2.6 Fold

Do multiple commits to create a patch, then fold them into one commit.

Now I go into a bit of a planning spree.

```
cd testmy
echo "Scenes:" >> plan.txt
hg ci -m "we need scenes"

echo "- Lost appears" >> plan.txt
hg ci -m "scene"
echo "- People vanish" >> plan.txt
hg ci -m "scene"
echo "- Portals during dreamtime" >> plan.txt
hg ci -m "scene"
echo
hg log -G -r 9:
cd ..
```

```
$ $ $ $ $ $ $ $ $
@  changeset:   12:2911e2fd2c82
|  tag:         tip
|  user:        my
|  date:        Sat Jan 12 00:31:22 2013 +0100
|  summary:     scene
|
o  changeset:   11:80ac442b013c
|  user:        my
|  date:        Sat Jan 12 00:31:22 2013 +0100
|  summary:     scene
|
o  changeset:   10:6f211bb0106e
|  user:        my
|  date:        Sat Jan 12 00:31:21 2013 +0100
|  summary:     scene
|
o  changeset:   9:e019913d9ef0
|  user:        my
|  date:        Sat Jan 12 00:31:21 2013 +0100
```

```
|  summary:     we need scenes
|
```

Yes, I tend to do that...

But we actually only need one change, so make it one by folding the last 4 changes changes into a single commit.

Since fold needs an interactive editor (it does not take -m, yet), we will leave that out. The commented commands allow you to fold the changesets.

```
cd testmy
# hg fold -r "-1:-4"
# hg log -G -r 9:
cd ..
```

2.7 Split

Do one big commit, then split it into two atomic commits.

Now I apply the scenes to wishes, places and people. Which is not useful: First I should apply them to the wishes and check if all wishes are fullfilled. But while writing I forgot that, and anxious to show my co-gamemaster, I just did one big commit.

```
cd testmy
sed -i "s/The Judicator wants Action/The Judicator wants Action - portals/" plan.txt
sed -i "s/The village/The village - lost, vanish, portals/" plan.txt
hg ci -m "Apply Scenes to people and places."
echo
hg log -G -r 12:
cd ..
```

```
$ $ $
@  changeset:   13:376374eec1ed
|  tag:         tip
|  user:        my
|  date:        Sat Jan 12 00:31:25 2013 +0100
|  summary:     Apply Scenes to people and places.
|
o  changeset:   12:2911e2fd2c82
|  user:        my
|  date:        Sat Jan 12 00:31:22 2013 +0100
|  summary:     scene
|
```

Let's fix that: uncommit it and commit it as separate changes. Normally I would just use ‘hg record‘ to interactively select changes to record. Since this is a non-interactive test, I manually undo and redo changes instead.

```
cd testmy
hg uncommit --all # to undo all changes, not just those for specified files
hg diff
sed -i "s/The village - lost, vanish, portals/The village/" plan.txt
hg amend -m "Apply scenes to wishes"
sed -i "s/The village/The village - lost, vanish, portals/" plan.txt
hg commit -m "Apply scenes to places"
echo
hg log -G -r 12:
cd ..


new changeset is empty
(use "hg kill ." to remove it)
diff --git a/plan.txt b/plan.txt
--- a/plan.txt
+++ b/plan.txt
@@ -1,10 +1,10 @@
 Wishes:
 - The Solek wants emotionally intense situations
-- The Judicator wants Action
+- The Judicator wants Action - portals


 Places:
-- The village
+- The village - lost, vanish, portals
 - The researchers cave

 Scenes:
$ $ $ $
@  changeset:   17:0cab0ae7e432
|  tag:         tip
|  user:        my
|  date:        Sat Jan 12 00:31:28 2013 +0100
|  summary:     Apply scenes to places
|
o  changeset:   16:6f6b6970efcd
|  parent:      12:2911e2fd2c82
|  user:        my
|  date:        Sat Jan 12 00:31:28 2013 +0100
|  summary:     Apply scenes to wishes
|
o  changeset:   12:2911e2fd2c82
```

```
|  user:        my
|  date:        Sat Jan 12 00:31:22 2013 +0100
|  summary:     scene
|
```

2.8 . . . as afterthought

Do one big commit, add an atomic commit. Then split the big commit.

Let's get the changes from our co-gamemaster and apply people to wishes, places and scenes. Then add a scene we need to fullfill the wishes and clean the commits afterwards.

First get the changes:

```
cd testmy
hg pull ../testother
hg merge  --tool internal:merge tip # the new head from our co-gamemaster
# fix the conflicts
sed -i "s/<<<.*local//" plan.txt
sed -i "s/====.*/\n/" plan.txt
sed -i "s/>>>.*other//" plan.txt
# mark them as solved.
hg resolve -m
hg commit -m "merge people"
echo
hg log -G -r 12:
cd ..
```

```
pulling from ../testother
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files (+1 heads)
(run 'hg heads .' to see heads, 'hg merge' to merge)
merging plan.txt
warning: conflicts during merge.
merging plan.txt incomplete! (edit conflicts, then use 'hg resolve --mark')
0 files updated, 0 files merged, 0 files removed, 1 files unresolved
use 'hg resolve' to retry unresolved file merges or 'hg update -C .' to abandon
$ $ $ $ $ $ $
@    changeset:   19:951f3586151e
|\   tag:         tip
| |  parent:      17:0cab0ae7e432
| |  parent:      18:ca58acfa3af5
```

```
| |  user:        my
| |  date:        Sat Jan 12 00:31:31 2013 +0100
| |  summary:     merge people
| |
| o  changeset:   18:ca58acfa3af5
| |  parent:      8:ba84b8c0e6a2
| |  user:        other
| |  date:        Sat Jan 12 00:31:06 2013 +0100
| |  summary:     The people
| |
o |  changeset:   17:0cab0ae7e432
| |  user:        my
| |  date:        Sat Jan 12 00:31:28 2013 +0100
| |  summary:     Apply scenes to places
| |
o |  changeset:   16:6f6b6970efcd
| |  parent:      12:2911e2fd2c82
| |  user:        my
| |  date:        Sat Jan 12 00:31:28 2013 +0100
| |  summary:     Apply scenes to wishes
| |
o |  changeset:   12:2911e2fd2c82
| |  user:        my
| |  date:        Sat Jan 12 00:31:22 2013 +0100
| |  summary:     scene
| |
```

Now we have all changes in our repo. We begin to apply people to wishes, places and scenes.

```
cd testmy
sed -i "s/The Solek wants emotionally intense situations/The Solek wants emotionally inten
sed -i "s/Lost appears/Lost appears | Lost/" plan.txt
sed -i "s/People vanish/People vanish | Specter/" plan.txt
hg commit -m "apply people to wishes, places and scenes"
echo
hg log -G -r 19:
cat plan.txt
cd ..
```

```
$ $ $ $
@  changeset:   20:ad476101f0c6
|  tag:         tip
```

```
|  user:        my
|  date:        Sat Jan 12 00:31:33 2013 +0100
|  summary:     apply people to wishes, places and scenes
|
o    changeset:   19:951f3586151e
|\   parent:      17:0cab0ae7e432
| |  parent:      18:ca58acfa3af5
| |  user:        my
| |  date:        Sat Jan 12 00:31:31 2013 +0100
| |  summary:     merge people
| |
Wishes:
- The Solek wants emotionally intense situations | specter, Lost
- The Judicator wants Action - portals


Places:
- The village - lost, vanish, portals
- The researchers cave


Scenes:
- Lost appears | Lost
- People vanish | Specter
- Portals during dreamtime


People:
- The Lost
- The Specter
```

As you can see, the specter only applies to the wishes, and we miss a person for the action.

Let's fix that.

```
cd testmy
sed -i "s/- The Specter/- The Specter\n- Wild Memories/" plan.txt
sed -i "s/- Portals during dreamtime/- Portals during dreamtime\n- Unconnected Memories/"
hg ci -m "Added wild memories to fullfill the wish for action"
echo
hg log -G -r 19:
cd ..
```

```
$ $ $
@  changeset:   21:93cb42ca555a
|  tag:         tip
|  user:        my
|  date:        Sat Jan 12 00:31:35 2013 +0100
|  summary:     Added wild memories to fullfill the wish for action
|
o  changeset:   20:ad476101f0c6
|  user:        my
|  date:        Sat Jan 12 00:31:33 2013 +0100
|  summary:     apply people to wishes, places and scenes
|
o    changeset:   19:951f3586151e
|\   parent:      17:0cab0ae7e432
| |  parent:      18:ca58acfa3af5
| |  user:        my
| |  date:        Sat Jan 12 00:31:31 2013 +0100
| |  summary:     merge people
| |
```

Now split the big change into applying people first to wishes, then to places and scenes.

```
cd testmy
# go back to the big change
hg up -r -2
# uncommit it
hg uncommit --all
# Now rework it into two commits
sed -i "s/- Lost appears | Lost/- Lost appears/" plan.txt
sed -i "s/- People vanish | Specter/- People vanish/" plan.txt
hg amend -m "Apply people to wishes"
sed -i "s/- Lost appears/- Lost appears | Lost/" plan.txt
sed -i "s/- People vanish/- People vanish | Specter/" plan.txt
hg commit -m "Apply people to scenes"
# let\T1\textquoteright s mark this for later use
hg book splitchanges
# and evolve to get rid of the obsoletes
echo
hg evolve --any
hg log -G -r 19:
cd ..
```

```
$ 1 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

```
$ new changeset is empty
(use "hg kill ." to remove it)
1 new unstable changesets
$ $ $ $ $ $ $ $ $ $
move:[21] Added wild memories to fullfill the wish for action
atop:[24] Apply people to wishes
merging plan.txt
@  changeset:   26:1d1c2efbc952
|  tag:         tip
|  parent:      24:3bc7d90f80a7
|  user:        my
|  date:        Sat Jan 12 00:31:35 2013 +0100
|  summary:     Added wild memories to fullfill the wish for action
|
| o  changeset:   25:e5908fa8f249
|/   bookmark:    splitchanges
|    user:        my
|    date:        Sat Jan 12 00:31:38 2013 +0100
|    summary:     Apply people to scenes
|
o  changeset:   24:3bc7d90f80a7
|  parent:      19:951f3586151e
|  user:        my
|  date:        Sat Jan 12 00:31:38 2013 +0100
|  summary:     Apply people to wishes
|
o    changeset:   19:951f3586151e
|\   parent:      17:0cab0ae7e432
| |  parent:      18:ca58acfa3af5
| |  user:        my
| |  date:        Sat Jan 12 00:31:31 2013 +0100
| |  summary:     merge people
| |
```

You can see the additional commit sticking out. We want to get the history easy to follow, so we just graft the last last change atop the split changes.

note: We seem to have the workdir on the new changeset instead of on the one we did before the evolve. I assume that's a bug to fix.

```
cd testmy
hg up splitchanges
hg graft -O tip
hg log -G -r 19:
cd ..
```

```
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
grafting revision 26
merging plan.txt
@  changeset:   27:27a74090f473
|  bookmark:    splitchanges
|  tag:         tip
|  parent:      25:e5908fa8f249
|  user:        my
|  date:        Sat Jan 12 00:31:35 2013 +0100
|  summary:     Added wild memories to fullfill the wish for action
|
o  changeset:   25:e5908fa8f249
|  user:        my
|  date:        Sat Jan 12 00:31:38 2013 +0100
|  summary:     Apply people to scenes
|
o  changeset:   24:3bc7d90f80a7
|  parent:      19:951f3586151e
|  user:        my
|  date:        Sat Jan 12 00:31:38 2013 +0100
|  summary:     Apply people to wishes
|
o    changeset:   19:951f3586151e
|\   parent:      17:0cab0ae7e432
| |  parent:      18:ca58acfa3af5
| |  user:        my
| |  date:        Sat Jan 12 00:31:31 2013 +0100
| |  summary:     merge people
| |
```

note: We use graft here, because using a second amend would just change the changeset in between but not add another change. If there had been more changes after the single followup commit, we would simply have called evolve to fix them, because graft -O left an obsolete marker on the grafted changeset, so evolve would have seen how to change all its children.

That's it. All that's left is finishing plan.txt, but I'll rather do that outside this guide :)

3 Conclusion

Evolve does a pretty good job at making it convenient and safe to rework history. If you're an early adopter, I can advise testing it yourself. Otherwise, it might be better to wait until more early adopters tested it and polished its rough edges.

note: hg amend was subsumed into hg commit –amend, so the dedicated command will likely disappear.

note: This guide was created by Arne Babenhauserheide with emacs org-mode.