

cb87a83960 ▾

...

py / py / _path / svnurl.py / <> Jump to ▾



scop Python 3.6 invalid escape sequence deprecation fixes ...

History

2 contributors



380 lines (340 sloc) | 14.4 KB

...

```

1  """
2  module defining a subversion path object based on the external
3  command 'svn'. This modules aims to work with svn 1.3 and higher
4  but might also interact well with earlier versions.
5  """
6
7  import os, sys, time, re
8  import py
9  from py import path, process
10 from py._path import common
11 from py._path import svnwc as svncommon
12 from py._path.cacheutil import BuildcostAccessCache, AgingCache
13
14 DEBUG=False
15
16 class SvnCommandPath(svncommon.SvnPathBase):
17     """ path implementation that offers access to (possibly remote) subversion
18     repositories. """
19
20     _lsrevcache = BuildcostAccessCache(maxentries=128)
21     _lsnorevcache = AgingCache(maxentries=1000, maxseconds=60.0)
22
23     def __new__(cls, path, rev=None, auth=None):
24         self = object.__new__(cls)
25         if isinstance(path, cls):
26             rev = path.rev
27             auth = path.auth
28             path = path.strpath
29         svncommon.checkbadchars(path)

```

```

30     path = path.rstrip('/')
31     self.strpath = path
32     self.rev = rev
33     self.auth = auth
34     return self
35
36 def __repr__(self):
37     if self.rev == -1:
38         return 'svnurl(%r)' % self.strpath
39     else:
40         return 'svnurl(%r, %r)' % (self.strpath, self.rev)
41
42 def _svnwithrev(self, cmd, *args):
43     """ execute an svn command, append our own url and revision """
44     if self.rev is None:
45         return self._svnwrite(cmd, *args)
46     else:
47         args = ['-r', self.rev] + list(args)
48         return self._svnwrite(cmd, *args)
49
50 def _svnwrite(self, cmd, *args):
51     """ execute an svn command, append our own url """
52     l = ['svn %s' % cmd]
53     args = ['"%s"' % self._escape(item) for item in args]
54     l.extend(args)
55     l.append('"%s"' % self._encodedurl())
56     # fixing the locale because we can't otherwise parse
57     string = " ".join(l)
58     if DEBUG:
59         print("execing %s" % string)
60     out = self._svncmdexecauth(string)
61     return out
62
63 def _svncmdexecauth(self, cmd):
64     """ execute an svn command 'as is' """
65     cmd = svncommon.fixlocale() + cmd
66     if self.auth is not None:
67         cmd += ' ' + self.auth.makecmdoptions()
68     return self._cmdexec(cmd)
69
70 def _cmdexec(self, cmd):
71     try:
72         out = process.cmdexec(cmd)
73     except py.process.cmdexec.Error:
74         e = sys.exc_info()[1]
75         if (e.err.find('File Exists') != -1 or
76             e.err.find('File already exists') != -1):
77             raise py.error.EEXIST(self)
78         raise

```

```

79         return out
80
81     def _svnopenauth(self, cmd):
82         """ execute an svn command, return a pipe for reading stdin """
83         cmd = svncommon.fixlocale() + cmd
84         if self.auth is not None:
85             cmd += ' ' + self.auth.makecmdoptions()
86         return self._popen(cmd)
87
88     def _popen(self, cmd):
89         return os.popen(cmd)
90
91     def _encodedurl(self):
92         return self._escape(self.strpath)
93
94     def _norev_delentry(self, path):
95         auth = self.auth and self.auth.makecmdoptions() or None
96         self._lsnorevcache.delentry((str(path), auth))
97
98     def open(self, mode='r'):
99         """ return an opened file with the given mode. """
100         if mode not in ("r", "rU",):
101             raise ValueError("mode %r not supported" % (mode,))
102         assert self.check(file=1) # svn cat returns an empty file otherwise
103         if self.rev is None:
104             return self._svnopenauth('svn cat "%s"' % (
105                                     self._escape(self.strpath), ))
106         else:
107             return self._svnopenauth('svn cat -r %s "%s"' % (
108                                     self.rev, self._escape(self.strpath)))
109
110     def dirpath(self, *args, **kwargs):
111         """ return the directory path of the current path joined
112             with any given path arguments.
113         """
114         l = self.strpath.split(self.sep)
115         if len(l) < 4:
116             raise py.error.EINVAL(self, "base is not valid")
117         elif len(l) == 4:
118             return self.join(*args, **kwargs)
119         else:
120             return self.new(basename='').join(*args, **kwargs)
121
122     # modifying methods (cache must be invalidated)
123     def mkdir(self, *args, **kwargs):
124         """ create & return the directory joined with args.
125             pass a 'msg' keyword argument to set the commit message.
126         """
127         commit_msg = kwargs.get('msg', "mkdir by py lib invocation")

```

```

128         createpath = self.join(*args)
129         createpath._svnwrite('mkdir', '-m', commit_msg)
130         self._norev_delentry(createpath.dirpath())
131         return createpath
132
133     def copy(self, target, msg='copied by py lib invocation'):
134         """ copy path to target with checkin message msg. """
135         if getattr(target, 'rev', None) is not None:
136             raise py.error.EINVAL(target, "revisions are immutable")
137         self._svncmdexcauth('svn copy -m "%s" "%s" "%s"' % (msg,
138             self._escape(self), self._escape(target)))
139         self._norev_delentry(target.dirpath())
140
141     def rename(self, target, msg="renamed by py lib invocation"):
142         """ rename this path to target with checkin message msg. """
143         if getattr(self, 'rev', None) is not None:
144             raise py.error.EINVAL(self, "revisions are immutable")
145         self._svncmdexcauth('svn move -m "%s" --force "%s" "%s"' % (
146             msg, self._escape(self), self._escape(target)))
147         self._norev_delentry(self.dirpath())
148         self._norev_delentry(self)
149
150     def remove(self, rec=1, msg='removed by py lib invocation'):
151         """ remove a file or directory (or a directory tree if rec=1) with
152         checkin message msg. """
153         if self.rev is not None:
154             raise py.error.EINVAL(self, "revisions are immutable")
155         self._svncmdexcauth('svn rm -m "%s" "%s"' % (msg, self._escape(self)))
156         self._norev_delentry(self.dirpath())
157
158     def export(self, topath):
159         """ export to a local path
160
161         topath should not exist prior to calling this, returns a
162         py.path.local instance
163         """
164         topath = py.path.local(topath)
165         args = ['"%s"' % (self._escape(self),),
166             '"%s"' % (self._escape(topath),)]
167         if self.rev is not None:
168             args = ['-r', str(self.rev)] + args
169         self._svncmdexcauth('svn export %s' % (' '.join(args),))
170         return topath
171
172     def ensure(self, *args, **kwargs):
173         """ ensure that an args-joined path exists (by default as
174         a file). If you specify a keyword argument 'dir=True'
175         then the path is forced to be a directory path.
176         """

```

```

177         if getattr(self, 'rev', None) is not None:
178             raise py.error.EINVAL(self, "revisions are immutable")
179         target = self.join(*args)
180         dir = kwargs.get('dir', 0)
181         for x in target.parts(reverse=True):
182             if x.check():
183                 break
184         else:
185             raise py.error.ENOENT(target, "has not any valid base!")
186         if x == target:
187             if not x.check(dir=dir):
188                 raise dir and py.error.ENOTDIR(x) or py.error.EISDIR(x)
189             return x
190         tocreate = target.relto(x)
191         basename = tocreate.split(self.sep, 1)[0]
192         tempdir = py.path.local.mkdtemp()
193         try:
194             tempdir.ensure(tocreate, dir=dir)
195             cmd = 'svn import -m "%s" "%s" "%s"' % (
196                 "ensure %s" % self._escape(tocreate),
197                 self._escape(tempdir.join(basename)),
198                 x.join(basename)._encodedurl())
199             self._svncmdexecauth(cmd)
200             self._norev_delentry(x)
201         finally:
202             tempdir.remove()
203         return target
204
205     # end of modifying methods
206     def _propget(self, name):
207         res = self._svnwithrev('propget', name)
208         return res[:-1] # strip trailing newline
209
210     def _proplist(self):
211         res = self._svnwithrev('proplist')
212         lines = res.split('\n')
213         lines = [x.strip() for x in lines[1:]]
214         return svncommon.PropListDict(self, lines)
215
216     def info(self):
217         """ return an Info structure with svn-provided information. """
218         parent = self.dirpath()
219         nameinfo_seq = parent._listdir_nameinfo()
220         bn = self.basename
221         for name, info in nameinfo_seq:
222             if name == bn:
223                 return info
224         raise py.error.ENOENT(self)
225

```

```

226
227 def _listdir_nameinfo(self):
228     """ return sequence of name-info directory entries of self """
229     def builder():
230         try:
231             res = self._svnwithrev('ls', '-v')
232         except process.cmdexec.Error:
233             e = sys.exc_info()[1]
234             if e.err.find('non-existent in that revision') != -1:
235                 raise py.error.ENOENT(self, e.err)
236             elif e.err.find("E200009:") != -1:
237                 raise py.error.ENOENT(self, e.err)
238             elif e.err.find('File not found') != -1:
239                 raise py.error.ENOENT(self, e.err)
240             elif e.err.find('not part of a repository') != -1:
241                 raise py.error.ENOENT(self, e.err)
242             elif e.err.find('Unable to open') != -1:
243                 raise py.error.ENOENT(self, e.err)
244             elif e.err.lower().find('method not allowed') != -1:
245                 raise py.error.EACCES(self, e.err)
246             raise py.error.Error(e.err)
247         lines = res.split('\n')
248         nameinfo_seq = []
249         for lsline in lines:
250             if lsline:
251                 info = InfoSvnCommand(lsline)
252                 if info._name != '.': # svn 1.5 produces '.' dirs,
253                     nameinfo_seq.append((info._name, info))
254         nameinfo_seq.sort()
255         return nameinfo_seq
256     auth = self.auth and self.auth.makecmdoptions() or None
257     if self.rev is not None:
258         return self._lsrevcache.getorbuild((self.strpath, self.rev, auth),
259                                             builder)
260     else:
261         return self._lsnorevcache.getorbuild((self.strpath, auth),
262                                             builder)
263
264 def listdir(self, fil=None, sort=None):
265     """ list directory contents, possibly filter by the given fil func
266         and possibly sorted.
267     """
268     if isinstance(fil, str):
269         fil = common.FNMatcher(fil)
270     nameinfo_seq = self._listdir_nameinfo()
271     if len(nameinfo_seq) == 1:
272         name, info = nameinfo_seq[0]
273         if name == self.basename and info.kind == 'file':
274             #if not self.check(dir=1):

```

```

275         raise py.error.ENOTDIR(self)
276     paths = [self.join(name) for (name, info) in nameinfo_seq]
277     if fil:
278         paths = [x for x in paths if fil(x)]
279     self._sortlist(paths, sort)
280     return paths
281
282
283     def log(self, rev_start=None, rev_end=1, verbose=False):
284         """ return a list of LogEntry instances for this path.
285         rev_start is the starting revision (defaulting to the first one).
286         rev_end is the last revision (defaulting to HEAD).
287         if verbose is True, then the LogEntry instances also know which files changed.
288         """
289         assert self.check() #make it simpler for the pipe
290         rev_start = rev_start is None and "HEAD" or rev_start
291         rev_end = rev_end is None and "HEAD" or rev_end
292
293         if rev_start == "HEAD" and rev_end == 1:
294             rev_opt = ""
295         else:
296             rev_opt = "-r %s:%s" % (rev_start, rev_end)
297         verbose_opt = verbose and "-v" or ""
298         xmlpipe = self._svnopenauth('svn log --xml %s %s "%s"' %
299                                     (rev_opt, verbose_opt, self.strpath))
300         from xml.dom import minidom
301         tree = minidom.parse(xmlpipe)
302         result = []
303         for logentry in filter(None, tree.firstChild.childNodes):
304             if logentry.nodeType == logentry.ELEMENT_NODE:
305                 result.append(svncommon.LogEntry(logentry))
306         return result
307
308 #012345678901234567890123456789012345678901234567
309 # 2256      hpk      165 Nov 24 17:55 __init__.py
310 # XXX spotted by Guido, SVN 1.3.0 has different aligning, breaks the code!!!
311 # 1312 johnny      1627 May 05 14:32 test_decorators.py
312 #
313 class InfoSvnCommand:
314     # the '0?' part in the middle is an indication of whether the resource is
315     # locked, see 'svn help ls'
316     lspattern = re.compile(
317         r'^ *(?P<rev>\d+) +(?P<author>.+?) +(0? *(?P<size>\d+))?' '
318         r'*(?P<date>\w+ +\d{2} +[\d:]+) +(?P<file>.*$)'
319     def __init__(self, line):
320         # this is a typical line from 'svn ls http://...'
321         #_ 1127      jum      0 Jul 13 15:28 branch/
322         match = self.lspattern.match(line)
323         data = match.groupdict()

```

```

324     self._name = data['file']
325     if self._name[-1] == '/':
326         self._name = self._name[:-1]
327         self.kind = 'dir'
328     else:
329         self.kind = 'file'
330     #self.has_props = l.pop(0) == 'P'
331     self.created_rev = int(data['rev'])
332     self.last_author = data['author']
333     self.size = data['size'] and int(data['size']) or 0
334     self.mtime = parse_time_with_missing_year(data['date'])
335     self.time = self.mtime * 1000000
336
337     def __eq__(self, other):
338         return self.__dict__ == other.__dict__
339
340
341     # _____
342     #
343     # helper functions
344     # _____
345     def parse_time_with_missing_year(timestr):
346         """ analyze the time part from a single line of "svn ls -v"
347         the svn output doesn't show the year makes the 'timestr'
348         ambiguous.
349         """
350         import calendar
351         t_now = time.gmtime()
352
353         tparts = timestr.split()
354         month = time.strptime(tparts.pop(0), '%b')[1]
355         day = time.strptime(tparts.pop(0), '%d')[2]
356         last = tparts.pop(0) # year or hour:minute
357         try:
358             if ":" in last:
359                 raise ValueError()
360             year = time.strptime(last, '%Y')[0]
361             hour = minute = 0
362         except ValueError:
363             hour, minute = time.strptime(last, '%H:%M')[3:5]
364             year = t_now[0]
365
366             t_result = (year, month, day, hour, minute, 0, 0, 0, 0)
367             if t_result > t_now:
368                 year -= 1
369             t_result = (year, month, day, hour, minute, 0, 0, 0, 0)
370         return calendar.timegm(t_result)
371
372     class PathEntry:

```



```
373     def __init__(self, ppart):
374         self.strpath = ppart.firstChild.nodeValue.encode('UTF-8')
375         self.action = ppart.getAttribute('action').encode('UTF-8')
376         if self.action == 'A':
377             self.copyfrom_path = ppart.getAttribute('copyfrom-path').encode('UTF-8')
378             if self.copyfrom_path:
379                 self.copyfrom_rev = int(ppart.getAttribute('copyfrom-rev'))
```