scop Python 3.6 invalid escape sequence deprecation fixes ... History

2 contributors  

380 lines (340 sloc) | 14.4 KB ...

```
"""
module defining a subversion path object based on the external
command 'svn'. This modules aims to work with svn 1.3 and higher
but might also interact well with earlier versions.
"""

import os, sys, time, re
import py
from py import path, process
from py._path import common
from py._path import svnwc as svncommon
from py._path.cacheutil import BuildcostAccessCache, AgingCache

DEBUG=False

class SvnCommandPath(svncommon.SvnPathBase):
    """ path implementation that offers access to (possibly remote) subversion
    repositories. """

    _lsrevcache = BuildcostAccessCache(maxentries=128)
    _lsnorevcache = AgingCache(maxentries=1000, maxseconds=60.0)

    def __new__(cls, path, rev=None, auth=None):
        self = object.__new__(cls)
        if isinstance(path, cls):
            rev = path.rev
            auth = path.auth
            path = path.strpath
        svncommon.checkbadchars(path)
```

```
        path = path.rstrip('/')
        self.strpath = path
        self.rev = rev
        self.auth = auth
        return self

    def __repr__(self):
        if self.rev == -1:
            return 'svnurl(%r)' % self.strpath
        else:
            return 'svnurl(%r, %r)' % (self.strpath, self.rev)

    def _svnwithrev(self, cmd, *args):
        """ execute an svn command, append our own url and revision """
        if self.rev is None:
            return self._svnwrite(cmd, *args)
        else:
            args = ['-r', self.rev] + list(args)
            return self._svnwrite(cmd, *args)

    def _svnwrite(self, cmd, *args):
        """ execute an svn command, append our own url """
        l = ['svn %s' % cmd]
        args = ['"%s"' % self._escape(item) for item in args]
        l.extend(args)
        l.append('"%s"' % self._encodedurl())
        # fixing the locale because we can't otherwise parse
        string = " ".join(l)
        if DEBUG:
            print("execing %s" % string)
        out = self._svncmdexecauth(string)
        return out

    def _svncmdexecauth(self, cmd):
        """ execute an svn command 'as is' """
        cmd = svncommon.fixlocale() + cmd
        if self.auth is not None:
            cmd += ' ' + self.auth.makecmdoptions()
        return self._cmdexec(cmd)

    def _cmdexec(self, cmd):
        try:
            out = process.cmdexec(cmd)
        except py.process.cmdexec.Error:
            e = sys.exc_info()[1]
            if (e.err.find('File Exists') != -1 or
                            e.err.find('File already exists') != -1):
                raise py.error.EEXIST(self)
            raise
```

```
        return out

    def _svnpopenauth(self, cmd):
        """ execute an svn command, return a pipe for reading stdin """
        cmd = svncommon.fixlocale() + cmd
        if self.auth is not None:
            cmd += ' ' + self.auth.makecmdoptions()
        return self._popen(cmd)

    def _popen(self, cmd):
        return os.popen(cmd)

    def _encodedurl(self):
        return self._escape(self.strpath)

    def _norev_delentry(self, path):
        auth = self.auth and self.auth.makecmdoptions() or None
        self._lsnorevcache.delentry((str(path), auth))

    def open(self, mode='r'):
        """ return an opened file with the given mode. """
        if mode not in ("r", "rU",):
            raise ValueError("mode %r not supported" % (mode,))
        assert self.check(file=1) # svn cat returns an empty file otherwise
        if self.rev is None:
            return self._svnpopenauth('svn cat "%s"' % (
                                      self._escape(self.strpath), ))
        else:
            return self._svnpopenauth('svn cat -r %s "%s"' % (
                                      self.rev, self._escape(self.strpath)))

    def dirpath(self, *args, **kwargs):
        """ return the directory path of the current path joined
            with any given path arguments.
        """
        l = self.strpath.split(self.sep)
        if len(l) < 4:
            raise py.error.EINVAL(self, "base is not valid")
        elif len(l) == 4:
            return self.join(*args, **kwargs)
        else:
            return self.new(basename='').join(*args, **kwargs)

    # modifying methods (cache must be invalidated)
    def mkdir(self, *args, **kwargs):
        """ create & return the directory joined with args.
        pass a 'msg' keyword argument to set the commit message.
        """
        commit_msg = kwargs.get('msg', "mkdir by py lib invocation")
```

```
        createpath = self.join(*args)
        createpath._svnwrite('mkdir', '-m', commit_msg)
        self._norev_delentry(createpath.dirpath())
        return createpath

    def copy(self, target, msg='copied by py lib invocation'):
        """ copy path to target with checkin message msg."""
        if getattr(target, 'rev', None) is not None:
            raise py.error.EINVAL(target, "revisions are immutable")
        self._svncmdexecauth('svn copy -m "%s" "%s" "%s"' %(msg,
                             self._escape(self), self._escape(target)))
        self._norev_delentry(target.dirpath())

    def rename(self, target, msg="renamed by py lib invocation"):
        """ rename this path to target with checkin message msg. """
        if getattr(self, 'rev', None) is not None:
            raise py.error.EINVAL(self, "revisions are immutable")
        self._svncmdexecauth('svn move -m "%s" --force "%s" "%s"' %(
                             msg, self._escape(self), self._escape(target)))
        self._norev_delentry(self.dirpath())
        self._norev_delentry(self)

    def remove(self, rec=1, msg='removed by py lib invocation'):
        """ remove a file or directory (or a directory tree if rec=1) with
checkin message msg."""
        if self.rev is not None:
            raise py.error.EINVAL(self, "revisions are immutable")
        self._svncmdexecauth('svn rm -m "%s" "%s"' %(msg, self._escape(self)))
        self._norev_delentry(self.dirpath())

    def export(self, topath):
        """ export to a local path

            topath should not exist prior to calling this, returns a
            py.path.local instance
        """
        topath = py.path.local(topath)
        args = ['"%s"' % (self._escape(self),),
                '"%s"' % (self._escape(topath),)]
        if self.rev is not None:
            args = ['-r', str(self.rev)] + args
        self._svncmdexecauth('svn export %s' % (' '.join(args),))
        return topath

    def ensure(self, *args, **kwargs):
        """ ensure that an args-joined path exists (by default as
            a file). If you specify a keyword argument 'dir=True'
            then the path is forced to be a directory path.
        """
```

```
        if getattr(self, 'rev', None) is not None:
            raise py.error.EINVAL(self, "revisions are immutable")
        target = self.join(*args)
        dir = kwargs.get('dir', 0)
        for x in target.parts(reverse=True):
            if x.check():
                break
        else:
            raise py.error.ENOENT(target, "has not any valid base!")
        if x == target:
            if not x.check(dir=dir):
                raise dir and py.error.ENOTDIR(x) or py.error.EISDIR(x)
            return x
        tocreate = target.relto(x)
        basename = tocreate.split(self.sep, 1)[0]
        tempdir = py.path.local.mkdtemp()
        try:
            tempdir.ensure(tocreate, dir=dir)
            cmd = 'svn import -m "%s" "%s" "%s"' % (
                    "ensure %s" % self._escape(tocreate),
                    self._escape(tempdir.join(basename)),
                    x.join(basename)._encodedurl())
            self._svncmdexecauth(cmd)
            self._norev_delentry(x)
        finally:
            tempdir.remove()
        return target

    # end of modifying methods
    def _propget(self, name):
        res = self._svnwithrev('propget', name)
        return res[:-1] # strip trailing newline

    def _proplist(self):
        res = self._svnwithrev('proplist')
        lines = res.split('\n')
        lines = [x.strip() for x in lines[1:]]
        return svncommon.PropListDict(self, lines)

    def info(self):
        """ return an Info structure with svn-provided information. """
        parent = self.dirpath()
        nameinfo_seq = parent._listdir_nameinfo()
        bn = self.basename
        for name, info in nameinfo_seq:
            if name == bn:
                return info
        raise py.error.ENOENT(self)

```

```
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):
```

```
                raise py.error.ENOTDIR(self)
        paths = [self.join(name) for (name, info) in nameinfo_seq]
        if fil:
            paths = [x for x in paths if fil(x)]
        self._sortlist(paths, sort)
        return paths


    def log(self, rev_start=None, rev_end=1, verbose=False):
        """ return a list of LogEntry instances for this path.
rev_start is the starting revision (defaulting to the first one).
rev_end is the last revision (defaulting to HEAD).
if verbose is True, then the LogEntry instances also know which files changed.
"""
        assert self.check() #make it simpler for the pipe
        rev_start = rev_start is None and "HEAD" or rev_start
        rev_end = rev_end is None and "HEAD" or rev_end

        if rev_start == "HEAD" and rev_end == 1:
            rev_opt = ""
        else:
            rev_opt = "-r %s:%s" % (rev_start, rev_end)
        verbose_opt = verbose and "-v" or ""
        xmlpipe =  self._svnpopenauth('svn log --xml %s %s "%s"' %
                                      (rev_opt, verbose_opt, self.strpath))
        from xml.dom import minidom
        tree = minidom.parse(xmlpipe)
        result = []
        for logentry in filter(None, tree.firstChild.childNodes):
            if logentry.nodeType == logentry.ELEMENT_NODE:
                result.append(svncommon.LogEntry(logentry))
        return result

#01234567890123456789012345678901234567890123467
#   2256      hpk        165 Nov 24 17:55 __init__.py
# XXX spotted by Guido, SVN 1.3.0 has different aligning, breaks the code!!!
#   1312 johnny           1627 May 05 14:32 test_decorators.py
#
class InfoSvnCommand:
    # the '0?' part in the middle is an indication of whether the resource is
    # locked, see 'svn help ls'
    lspattern = re.compile(
        r'^ *(?P<rev>\d+) +(?P<author>.+?) +(0? *(?P<size>\d+))? '
            r'*(?P<date>\w+ +\d{2} +[\d:]+) +(?P<file>.*)$')
    def __init__(self, line):
        # this is a typical line from 'svn ls http://...'
        #_    1127      jum        0 Jul 13 15:28 branch/
        match = self.lspattern.match(line)
        data = match.groupdict()
```

```
        self._name = data['file']
        if self._name[-1] == '/':
            self._name = self._name[:-1]
            self.kind = 'dir'
        else:
            self.kind = 'file'
        #self.has_props = l.pop(0) == 'P'
        self.created_rev = int(data['rev'])
        self.last_author = data['author']
        self.size = data['size'] and int(data['size']) or 0
        self.mtime = parse_time_with_missing_year(data['date'])
        self.time = self.mtime * 1000000

    def __eq__(self, other):
        return self.__dict__ == other.__dict__


#____________________________________________________
#
# helper functions
#____________________________________________________
def parse_time_with_missing_year(timestr):
    """ analyze the time part from a single line of "svn ls -v"
    the svn output doesn't show the year makes the 'timestr'
    ambigous.
    """
    import calendar
    t_now = time.gmtime()

    tparts = timestr.split()
    month = time.strptime(tparts.pop(0), '%b')[1]
    day = time.strptime(tparts.pop(0), '%d')[2]
    last = tparts.pop(0) # year or hour:minute
    try:
        if ":" in last:
            raise ValueError()
        year = time.strptime(last, '%Y')[0]
        hour = minute = 0
    except ValueError:
        hour, minute = time.strptime(last, '%H:%M')[3:5]
        year = t_now[0]

        t_result = (year, month, day, hour, minute, 0,0,0,0)
        if t_result > t_now:
            year -= 1
    t_result = (year, month, day, hour, minute, 0,0,0,0)
    return calendar.timegm(t_result)

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'))
```