theskumar / **python-dotenv** Public

Code | Issues 52 | Pull requests 37 | Actions | Wiki | Security and quality

Commit `790c5c0`

bbc2 authored on Mar 1 · 9 / 9 · Verified

```
Merge commit from fork

Changes for users:

- (BREAKING) `dotenv.set_key` and `dotenv.unset_key` used to follow
  symlinks in some situations. This is no longer the case.  For that
  behavior to be restored in all cases, `follow_symlinks=True` should be
  used.
- (BREAKING) In the CLI, `set` and `unset` used to follow symlinks in
  some situations. This is no longer the case.
- (BREAKING) `dotenv.set_key`, `dotenv.unset_key` and the CLI commands
  `set` and `unset` used to reset the file mode of the modified .env
  file to `0o600` in some situations. This is no longer the case: The
  original mode of the file is now preserved. Is the file needed to be
  created or wasn't a regular file, mode `0o600` is used.
```

`main` · v1.2.2

1 parent 43340da commit 790c5c0

3 files changed +199 -15

Top

Filter files…

- src/dotenv
  - cli.py
  - main.py
- tests
  - test_main.py

Search within code

src/dotenv/cli.py

```
@@ -114,7 +114,13 @@ def list_values(ctx: click.Context, output_format: str)
 -> None:
114 114 @click.argument("key", required=True)
115 115 @click.argument("value", required=True)
116 116 def set_value(ctx: click.Context, key: Any, value: Any) -> None:
117     -     """Store the given key/value."""
    117 +     """
    118 +     Store the given key/value.
    119 +
    120 +     This doesn't follow symlinks, to avoid accidentally modifying a file at a
    121 +     potentially untrusted path.
    122 +     """
    123 +
118 124     file = ctx.obj["FILE"]
119 125     quote = ctx.obj["QUOTE"]
120 126     export = ctx.obj["EXPORT"]
@@ -146,7 +152,12 @@ def get(ctx: click.Context, key: Any) -> None:
146 152 @click.pass_context
147 153 @click.argument("key", required=True)
148 154 def unset(ctx: click.Context, key: Any) -> None:
149     -     """Removes the given key."""
    155 +     """
    156 +     Removes the given key.
    157 +
    158 +     This doesn't follow symlinks, to avoid accidentally modifying a file at a
    159 +     potentially untrusted path.
    160 +     """
150 161     file = ctx.obj["FILE"]
151 162     quote = ctx.obj["QUOTE"]
152 163     success, key = unset_key(file, key, quote)
```

src/dotenv/main.py

```
@@ -2,7 +2,6 @@
2 2 import logging
3 3 import os
4 4 import pathlib
```

```
5       - import shutil
6   5     import stat
7   6     import sys
8   7     import tempfile
          @@ -14,9 +13,7 @@
14  13    from .variables import parse_variables
15  14
16  15    # A type alias for a string path to be used for the paths in this file.
17      - # These paths may flow to `open()` and `shutil.move()`; `shutil.move()`
18      - # only accepts string paths, not byte paths or file descriptors. See
19      - # https://github.com/python/typeshed/pull/6832.
        + # These paths may flow to `open()` and `os.replace()`.
20  17    StrPath = Union[str, "os.PathLike[str]"]
21  18
22  19    logger = logging.getLogger(__name__)
          @@ -142,21 +139,54 @@ def get_key(
142 139   def rewrite(
143 140       path: StrPath,
144 141       encoding: Optional[str],
        +     follow_symlinks: bool = False,
145 143   ) -> Iterator[Tuple[IO[str], IO[str]]]:
146     -     pathlib.Path(path).touch()
        +     if follow_symlinks:
        +         path = os.path.realpath(path)
147 146
148     -     with tempfile.NamedTemporaryFile(mode="w", encoding=encoding, delete=False)
      as dest:
        +     try:
        +         source: IO[str] = open(path, encoding=encoding)
        +         try:
        +             path_stat = os.lstat(path)
        +             original_mode: Optional[int] = (
        +                 stat.S_IMODE(path_stat.st_mode)
        +                 if stat.S_ISREG(path_stat.st_mode)
        +                 else None
        +             )
        +         except BaseException:
        +             source.close()
        +             raise
```

```
                +     except FileNotFoundError:
                +         source = io.StringIO("")
                +         original_mode = None
                +
                +     with tempfile.NamedTemporaryFile(
                +         mode="w",
                +         encoding=encoding,
                +         delete=False,
                +         prefix=".tmp_",
                +         dir=os.path.dirname(os.path.abspath(path)),
                +     ) as dest:
                +         dest_path = pathlib.Path(dest.name)
149  171            error = None
                +
150  173            try:
151       -              with open(path, encoding=encoding) as source:
                +              with source:
152  175                    yield (source, dest)
153  176            except BaseException as err:
154  177                error = err
155  178
156  179        if error is None:
157       -          shutil.move(dest.name, path)
                +          try:
                +              if original_mode is not None:
                +                  os.chmod(dest_path, original_mode)
                +
                +              os.replace(dest_path, path)
                +          except BaseException:
                +              dest_path.unlink(missing_ok=True)
                +              raise
158  188        else:
159       -          os.unlink(dest.name)
                +          dest_path.unlink(missing_ok=True)
160  190            raise error from None
161  191
162  192
                @@ -167,12 +197,16 @@ def set_key(
167  197        quote_mode: str = "always",
168  198        export: bool = False,
```

```
169  199         encoding: Optional[str] = "utf-8",
     200  +      follow_symlinks: bool = False,
170  201     ) -> Tuple[Optional[bool], str, str]:
171  202         """
172  203         Adds or Updates a key/value to the given .env
173  204
174       -      If the .env path given doesn't exist, fails instead of risking creating
175       -      an orphan .env somewhere in the filesystem
     205  +      The target .env file is created if it doesn't exist.
     206  +
     207  +      This function doesn't follow symlinks by default, to avoid accidentally
     208  +      modifying a file at a potentially untrusted path. If you don't need this
     209  +      protection and need symlinks to be followed, use `follow_symlinks`.
176  210         """
177  211         if quote_mode not in ("always", "auto", "never"):
178  212             raise ValueError(f"Unknown quote_mode: {quote_mode}")
          @@ -190,7 +224,10 @@ def set_key(
190  224         else:
191  225             line_out = f"{key_to_set}={value_out}\n"
192  226
193       -      with rewrite(dotenv_path, encoding=encoding) as (source, dest):
     227  +      with rewrite(dotenv_path, encoding=encoding,
           follow_symlinks=follow_symlinks) as (
     228  +          source,
     229  +          dest,
     230  +      ):
194  231             replaced = False
195  232             missing_newline = False
196  233             for mapping in with_warn_for_invalid_lines(parse_stream(source)):
          @@ -213,19 +250,27 @@ def unset_key(
213  250         key_to_unset: str,
214  251         quote_mode: str = "always",
215  252         encoding: Optional[str] = "utf-8",
     253  +      follow_symlinks: bool = False,
216  254     ) -> Tuple[Optional[bool], str]:
217  255         """
218  256         Removes a given key from the given `.env` file.
219  257
220  258         If the .env path given doesn't exist, fails.
221  259         If the given key doesn't exist in the .env, fails.
```

```
      260 +
      261 +     This function doesn't follow symlinks by default, to avoid accidentally
      262 +     modifying a file at a potentially untrusted path. If you don't need this
      263 +     protection and need symlinks to be followed, use `follow_symlinks`.
222   264         """
223   265         if not os.path.exists(dotenv_path):
224   266             logger.warning("Can't delete from %s - it doesn't exist.", dotenv_path)
225   267             return None, key_to_unset
226   268
227   269         removed = False
228       -     with rewrite(dotenv_path, encoding=encoding) as (source, dest):
      270 +     with rewrite(dotenv_path, encoding=encoding, follow_symlinks=follow_symlinks) as (
      271 +         source,
      272 +         dest,
      273 +     ):
229   274             for mapping in with_warn_for_invalid_lines(parse_stream(source)):
230   275                 if mapping.key == key_to_unset:
231   276                     removed = True
```

tests/test_main.py

```
@@ -62,6 +62,86 @@ def test_set_key_encoding(dotenv_path):
62    62         assert dotenv_path.read_text(encoding=encoding) == "a='é'\n"
63    63
64    64
      65 + @pytest.mark.skipif(
      66 +     sys.platform == "win32", reason="file mode bits behave differently on Windows"
      67 + )
      68 + def test_set_key_preserves_file_mode(dotenv_path):
      69 +     dotenv_path.write_text("a=x\n")
      70 +     dotenv_path.chmod(0o640)
      71 +     mode_before = stat.S_IMODE(dotenv_path.stat().st_mode)
      72 +
      73 +     dotenv.set_key(dotenv_path, "a", "y")
      74 +
      75 +     mode_after = stat.S_IMODE(dotenv_path.stat().st_mode)
      76 +     assert mode_before == mode_after
      77 +
```

```
78  +
79  + def test_rewrite_closes_file_handle_on_lstat_failure(tmp_path):
80  +     dotenv_path = tmp_path / ".env"
81  +     dotenv_path.write_text("a=x\n")
82  +     real_open = open
83  +     opened_handles = []
84  +
85  +     def tracking_open(*args, **kwargs):
86  +         handle = real_open(*args, **kwargs)
87  +         opened_handles.append(handle)
88  +         return handle
89  +
90  +     with mock.patch("dotenv.main.os.lstat", side_effect=FileNotFoundError):
91  +         with mock.patch("dotenv.main.open", side_effect=tracking_open):
92  +             dotenv.set_key(dotenv_path, "a", "x")
93  +
94  +     assert opened_handles, "expected at least one file to be opened"
95  +     assert all(handle.closed for handle in opened_handles)
96  +
97  +
98  + @pytest.mark.skipif(
99  +     sys.platform == "win32", reason="symlinks require elevated privileges on
    Windows"
100 + )
101 + def test_set_key_symlink_to_existing_file(tmp_path):
102 +     target = tmp_path / "target.env"
103 +     target.write_text("a=x\n")
104 +     symlink = tmp_path / ".env"
105 +     symlink.symlink_to(target)
106 +
107 +     dotenv.set_key(symlink, "a", "y")
108 +
109 +     assert target.read_text() == "a=x\n"
110 +     assert not symlink.is_symlink()
111 +     assert "a='y'" in symlink.read_text()
112 +     assert stat.S_IMODE(symlink.stat().st_mode) == 0o600
113 +
114 +
115 + @pytest.mark.skipif(
```

```
116 +     sys.platform == "win32", reason="symlinks require elevated privileges on
    Windows"
117 + )
118 + def test_set_key_symlink_to_missing_file(tmp_path):
119 +     target = tmp_path / "nx"
120 +     symlink = tmp_path / ".env"
121 +     symlink.symlink_to(target)
122 +
123 +     dotenv.set_key(symlink, "a", "x")
124 +
125 +     assert not target.exists()
126 +     assert not symlink.is_symlink()
127 +     assert symlink.read_text() == "a='x'\n"
128 +
129 +
130 + @pytest.mark.skipif(
131 +     sys.platform == "win32", reason="symlinks require elevated privileges on
    Windows"
132 + )
133 + def test_set_key_follow_symlinks(tmp_path):
134 +     target = tmp_path / "target.env"
135 +     target.write_text("a=x\n")
136 +     symlink = tmp_path / ".env"
137 +     symlink.symlink_to(target)
138 +
139 +     dotenv.set_key(symlink, "a", "y", follow_symlinks=True)
140 +
141 +     assert target.read_text() == "a='y'\n"
142 +     assert symlink.is_symlink()
143 +
144 +
65  145   @pytest.mark.skipif(
66  146       sys.platform != "win32" and os.geteuid() == 0,
67  147       reason="Root user can access files even with 000 permissions.",
    @@ -195,6 +275,54 @@ def test_unset_non_existent_file(tmp_path)::
195 275       )
196 276
197 277
    278 + @pytest.mark.skipif(
```

```
279 +     sys.platform == "win32", reason="symlinks require elevated privileges on
    Windows"
280 + )
281 + def test_unset_key_symlink_to_existing_file(tmp_path):
282 +     target = tmp_path / "target.env"
283 +     target.write_text("a=x\n")
284 +     symlink = tmp_path / ".env"
285 +     symlink.symlink_to(target)
286 +
287 +     dotenv.unset_key(symlink, "a")
288 +
289 +     assert target.read_text() == "a=x\n"
290 +     assert not symlink.is_symlink()
291 +     assert symlink.read_text() == ""
292 +
293 +
294 + @pytest.mark.skipif(
295 +     sys.platform == "win32", reason="symlinks require elevated privileges on
    Windows"
296 + )
297 + def test_unset_key_symlink_to_missing_file(tmp_path):
298 +     target = tmp_path / "nx"
299 +     symlink = tmp_path / ".env"
300 +     symlink.symlink_to(target)
301 +     logger = logging.getLogger("dotenv.main")
302 +
303 +     with mock.patch.object(logger, "warning") as mock_warning:
304 +         result = dotenv.unset_key(symlink, "a")
305 +
306 +     assert result == (None, "a")
307 +     assert symlink.is_symlink()
308 +     mock_warning.assert_called_once()
309 +
310 +
311 + @pytest.mark.skipif(
312 +     sys.platform == "win32", reason="symlinks require elevated privileges on
    Windows"
313 + )
314 + def test_unset_key_follow_symlinks(tmp_path):
315 +     target = tmp_path / "target.env"
```

```
+     target.write_text("a=b\n")
+     symlink = tmp_path / ".env"
+     symlink.symlink_to(target)
+
+     dotenv.unset_key(symlink, "a", follow_symlinks=True)
+
+     assert target.read_text() == ""
+     assert symlink.is_symlink()
+
+
  def prepare_file_hierarchy(path):
      """
      Create a temporary folder structure like the following:
```

Comments 0

Please sign in to comment.