pyrpo Documentation

Release 0.2.1

Wes Turner

May 24, 2015

Contents

Contents:

CHAPTER 1

pyrpo

GitHub | PyPi | Warehouse | Documentation | Travis-CI pyrpo: a shell command wrapper for hg, git, bzr, svn

1.1 Features

- Wrap and parse shell commands (largely as a reference)
- Walk for repository directories
- Generate reports for one or more repositories
- Call `hg status`, `git status`, etc.
- Generate mercurial `.hgsubs`
- Generate git `.gitsubmodule`
- Generate pip `requirements.txt`
- Generate shell script (to rebuild environment)
  - TODO: replicate branches/tags/revisions
- Functional namedtuples, iterators `yield` -ing generators
- optparse argument parsing (`-h`, `--help`)
- cookiecutter-pypackage project templating

1.2 Installing

Install from PyPi:

```
pip install pyrpo
```

Install from GitHub as editable (add a `pyrpo.pth` in `site-packages`):

```
pip install -e git+https://github.com/westurner/pyrpo#egg=pyrpo
```

1.3 Usage

Print help:

```
pyrpo --help
```

Scan for files:

```
# Scan and print a shell report
pyrpo -s . -r sh
pyrpo
```

Generate a TortoiseHG `thg-reporegistry.xml` file:

```
pyrpo -s . --thg
```

Generate a pip report:

```
pyrpo -r pip
```

Generate a status report:

```
pyrpo -r status
```

Generate an *.hgsubs* file:

```
pyrpo -r hgsub
```

Generate a `.gitmodules` file:

```
pyrpo -r gitmodule
```

Generate an origin report:

```
pyrpo -r origin
```

Generate a string report:

```
pyrpo -r str
```

1.4 License

BSD Software License

CHAPTER 2

Installation

At the command line:

```
$ easy_install pyrpo
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv pyrpo
$ pip install pyrpo
```

CHAPTER 3

Usage

To use pyrpo in a project:

```
import pyrpo
```

To use pyrpo as a shell command:

```
Usage: pyrpo [-h] [-v] [-q] [-s .] [-r <pip||full|status|hgsub|thg>] [--thg]

Options:
  -h, --help            show this help message and exit
  -s SCAN, --scan=SCAN  Path(s) to scan for repositories
  -r REPORTS, --report=REPORTS
                        pip || full || status || hgsub || thg
  --thg                 Write a thg-reporegistry.xml file to stdout
  --template=REPORT_TEMPLATE
                        Report template
  -v, --verbose
  -q, --quiet
```

CHAPTER 4

pyrpo

4.1 pyrpo package

4.1.1 Submodules

4.1.2 pyrpo.pyrpo module

class pyrpo.pyrpo.**BzrRepository**(*fpath*)

Bases: *pyrpo.pyrpo.Repository*

Bzr Repository subclass

label

str

vcs name (e.g. "hg")

prefix

str

vcs folder name (e.g. ".hg")

preparse

callable

pre-processing function for vcs log output

fsep

str

field separator / record delimiter

lsep

str

log output record separator / delimiter

fields

list of tuples

(colname, vcs formatter, postprocess_callable)

clone_cmd

str

name of commandline clone command (e.g. "clone")

field_trans
dict

mapping between bzr field outputs and tuple fields

logrgx
rgx

compiled regex for parsing log message fields

branch
Determine the branch name of the working directory of this Repository

Returns branch nick (`bzr nick`)

Return type str

clone_cmd = 'branch'

current_id
Determine the current revision identifier for the working directory of this Repository

Returns bazaar revision identifier (`bzr version-info --custom --template='{revision_id}'`)

Return type str

diff()
Run the repository diff command to compare working directory with 'tip'

Returns stdout output of `bzr diff`

Return type str

field_trans = {'timestamp': 'datestr', 'message': 'desc', 'revno': 'noderev', 'branch nick': 'branchnick', 'committer'

fields = (('datestr', None, <function parse at 0x7ff29e8eb230>), ('noderev', None, None), ('author', None, None), ('tags'

fsep = '\n'

label = 'bzr'

log(*n=None*, *template=None*)
Run the repository log command

Returns output of log command (`bzr log -l <n>`)

Return type str

logrgx = <_sre.SRE_Pattern object at 0x140b4e0>

lsep = '——'

prefix = '.bzr'

remote_url
Determine the primary remote url for this Repository

Returns primary remote url for this Repository (`bzr info | egrep '^ parent branch:' | awk '{ print $3 }'`)

Return type str

status
Run the repository status command and return stdout

Returns stdout output of `bzr status` command

Return type str

template = None

unique_id
Determine a "unique id" for this repository

Returns fpath of this repository

Return type str

class pyrpo.pyrpo.**GitRepository**(*fpath*)
Bases: *pyrpo.pyrpo.Repository*

Git Repository subclass

label
str

vcs name (e.g. "hg")

prefix
str

vcs folder name (e.g. ".hg")

preparse
callable

pre-processing function for vcs log output

fsep
str

field separator / record delimiter

lsep
str

log output record separator / delimiter

fields
list of tuples

(colname, vcs formatter, postprocess_callable)

clone_cmd
str

name of commandline clone command (e.g. "clone")

template
str

concatenated log output template

branch
Determine the branch name of the working directory of this Repository

Returns branch name (`git branch`)

Return type str

cfg_file

checkout_branch_cmd = 'checkout'

checkout_branch_hard_cmd = 'checkout -C'

checkout_rev_cmd = 'checkout -r'

clone_cmd = 'clone'

current_id

diff()
Run the repository diff command to compare working directory with 'tip'

Returns stdout output of `git diff`

Return type str

fields = (('datestr', '%ai', None, <function parse at 0x7ff29e8eb230>), ('noderev', '%h', None), ('author', '%an', None

incoming_cmd = 'incoming'

label = 'git'

last_commit
Get and parse the most recent Repository revision

Returns Repository log tuple

Return type tuple

log(*n=None*, ***kwargs*)
Run the repository log command

Returns output of log command (`git log -n <n> <--kwarg=value>`)

Return type str

loggraph()
Show the log annotated an with ASCII revlog graph

Returns stdout output from `git log --graph`

Return type str

new_branch_cmd = 'checkout -b'

outgoing_cmd = 'outgoing'

prefix = '.git'

pull_cmd = 'pull'

push_cmd = 'push'

remote_url
Determine the primary remote url for this Repository

Returns primary remote url for this Repository (`git config remote.origin.url`)

Return type str

remote_urls
Get all configured remote urls for this Repository

Returns primary remote url for this Repository (`git config -l | grep "url"`)

Return type str

repo_abspath_cmd = '-C'

serve()
Run the `git serve` command

status
Run the repository status command and return stdout

Returns stdout output of `hg status` command

Return type str

template = '%ai ||| %h ||| %an ||| %d ||| %s |..|'

unique_id
Determine a "unique id" for this repository

Returns fpath of this repository

Return type str

unpushed()

Returns stdout output from `git log master --not --remotes='*/master'.`

Return type str

class pyrpo.pyrpo.**MercurialRepository**(*fpath*)
Bases: *pyrpo.pyrpo.Repository*

Mercurial Repository subclass

label
str

vcs name (e.g. "hg")

prefix
str

vcs folder name (e.g. ".hg")

preparse
callable

pre-processing function for vcs log output

fsep
str

field separator / record delimiter

lsep
str

log output record separator / delimiter

fields
list of tuples

(colname, vcs formatter, postprocess_callable)

clone_cmd
str

name of commandline clone command (e.g. "clone")

template
str

concatenated log output template

branch
Determine the branch name of the working directory of this Repository

Returns branch name (`hg branch`)

Return type str

cfg_file

checkout_branch_cmd = 'checkout'

checkout_branch_hard_cmd = 'checkout -C'

checkout_rev_cmd = 'checkout -r'

clone_cmd = 'clone'

current_id
Determine the current revision identifier for the working directory of this Repository

Returns revision identifier (`hg id -i`)

Return type str

diff
Run the repository diff command to compare working directory with 'tip'

Returns stdout output of `hg diff -g`

Return type str

fields = (('datestr', '{date|isodatesec}', <function parse at 0x7ff29e8eb230>), ('noderev', '{node|short}', None), ('author

incoming_cmd = 'incoming'

label = 'hg'

log(*n=None*, ***kwargs*)
Run the repository log command

Returns output of log command (`hg log -l <n> <--kwarg=value>`)

Return type str

loggraph()
Show the log annotated an with ASCII revlog graph

Returns stdout output from `hg log --graph`

Return type str

new_branch_cmd = 'branch'

outgoing_cmd = 'outgoing'

overwrite_hg_paths(*text*)

prefix = '.hg'

pull_cmd = 'pull'

push_cmd = 'push'

remote_url
Determine the primary remote url for this Repository

Returns primary remote url for this Repository (`hg showconfig paths.default`)

Return type str

remote_urls
Get all configured remote urls for this Repository

Returns primary remote url for this Repository (`hg showconfig paths.default`)

Return type str

repo_abspath_cmd = '-R'

serve()
Run the `hg serve` command

sh(**args*, ***kwargs*)

status
Run the repository status command and return stdout

Returns stdout output of `hg status` command

Return type str

template = '{date|isodatesec} ||| {node|short} ||| {author|firstline} ||| {tags} ||| {desc} |..|'

classmethod to_hg_scheme_url(*url*)
Convert a URL to local mercurial URL schemes

Parameters **url** (*str*) – URL to map to local mercurial URL schemes

example

schemes.gh = git://github.com/ >> remote_url = git://github.com/westurner/dotfiles' >> to_hg_scheme_url(remote_url) << gh://westurner/dotfiles

classmethod to_normal_url(*url*)
convert a URL from local mercurial URL schemes to "normal" URLS

example

schemes.gh = git://github.com/ # remote_url = "gh://westurner/dotfiles" >> to_normal_url(remote_url) << 'git://github.com/westurner/dotfiles'

unique_id
Determine a "unique id" for this repository

Returns fpath of this repository

Return type str

unpushed()
Show outgoing changesets

Raises `NotImplementedError` – always

class pyrpo.pyrpo.**Repository**(*fpath*)
Bases: `object`

Abstract Repository class from which VCS-specific implementations derive

label
str

vcs name (e.g. "hg")

prefix
str

vcs folder name (e.g. ".hg")

preparse
callable

pre-processing function for vcs log output

fsep
str

field separator / record delimiter

lsep
str

log output record separator / delimiter

fields
list of tuples

(colname, vcs formatter, postprocess_callable)

clone_cmd
str

name of commandline clone command (e.g. "clone")

__str__()

Returns //fpath‘‘

Return type str: ‘‘label

__unicode__()

Returns //fpath‘‘

Return type str: ‘‘label

branch()
Determine the branch name of the working directory of this Repository

Returns branch name

Return type str

cfg_file

checkout_branch_cmd = 'checkout'

checkout_rev_cmd = 'checkout -r'

clone_cmd = 'clone'

ctime

Returns strftime-formatted ctime (creation time) of fpath

Return type str

current_id()
Determine the current revision identifier for the working directory of this Repository

Returns revision identifier

Return type str

diff()
Run the repository diff command to compare working directory with 'tip'

Returns stdout output of Repository diff command

Return type str

eggname

Returns basename of repository path (e.g. for pip_report)

Return type str

fields = []

find_symlinks
Find symlinks within fpath

Returns `path -> link`

Return type str

fsep = ' ||| '

full_report()
Show origin, last_commit, status, and parsed complete log history for this repository

Yields *str* – report lines

gitmodule_report()

Yields *str* – .gitmodules configuration lines for this repository

hgsub_report()

Yields *str* – .hgsubs line for this repository

incoming_cmd = 'incoming'

itersplit_to_fields(*_str*)
Split (or parse) repository log output into fields

Returns self._tuple(*values)

Return type tuple

json_report()

label = None

last_commit
Get and parse the most recent Repository revision

Returns Repository log tuple

Return type tuple

lately(*count=15*)
Show `count` most-recently modified files by mtime

Yields *tuple* – (strftime-formatted mtime, self.fpath-relative file path)

log(*n=None*, ***kwargs*)
Run the repository log command

Returns output of log command

Return type str

log_iter(*maxentries=None*, *template=None*, ***kwargs*)
Run the repository log command, parse, and yield log tuples

Yields *tuple* – self._tuple

lsep = ' |..|'

mtime

Returns strftime-formatted mtime (modification time) of fpath

Return type str

new_branch_cmd = 'branch'

origin_report()

Yields str: `label -- //fpath = remote_url`

outgoing_cmd = 'outgoing'

pip_report()
Show editable pip-requirements line necessary to clone this repository

Yields *str* – pip-requirements line necessary to clone this repository

prefix = None

preparse = None

pull_cmd = 'pull'

push_cmd = 'push'

read_cfg_file()

recreate_remotes_shellcmd()

Yields *str* – shell command blocks to recreate repo config

relpath
Determine the relative path to this repository

Returns relative path to this repository

Return type str

remote_url()
Determine the primary remote url for this Repository

Returns primary remote url for this Repository

Return type str

repo_abspath_cmd = '--repo-path'

sh(*cmd*, *ignore_error=False*, *cwd=None*, **args*, ***kwargs*)
Run a command with the current working directory set to self.fpath

Returns stdout output of wrapped call to `sh` (`subprocess.Popen`)

Return type str

sh_report(*full=True*, *latest=False*)
Show shell command necessary to clone this repository

If there is no primary remote url, prefix-comment the command

Keyword Arguments

- **full** (*bool*) – also include commands to recreate branches and remotes
- **latest** (*bool*) – checkout repo.branch instead of repo.current_id

Yields *str* – shell command necessary to clone this repository

status()
Run the repository status command and return stdout

Returns stdout output of Repository status command

Return type str

status_report()

Yields *str* – sh_report, last_commit, and repository status output

str_report()

Yields *str* – pretty-formatted representation of `self.to_dict`

to_dict()
Cast this Repository to a dict

Returns this Repository as a dict

Return type dict

to_json()

to_json_dict()

classmethod to_normal_url(*url*)

Parameters **url** (*str*) – repository URL (potentially with hg schemes)

Returns normal repository URL (un-hg-schemes-ed repository URL)

Return type str

unique_id()
Determine a "unique id" for this repository

Returns a "unique id" for this repository

Return type str

class pyrpo.pyrpo.**SvnRepository**(*fpath*)
Bases: *pyrpo.pyrpo.Repository*

SVN Repository subclass

label
str

vcs name (e.g. "hg")

prefix
str

vcs folder name (e.g. ".hg")

preparse
callable

pre-processing function for vcs log output

fsep
str

field separator / record delimiter

lsep
str

log output record separator / delimiter

fields
list of tuples

(colname, vcs formatter, postprocess_callable)

clone_cmd
str

name of commandline clone command (e.g. "clone")

current_id()
Determine the current revision identifier for the working directory of this Repository

Returns revision identifier (`svn info | grep "^Revision:  "`)

Return type str

diff()
Run the repository diff command to compare working directory with 'tip'

Returns stdout output of `svn diff`

Return type str

fields = (('noderev', None, None), ('author', None, None), ('datestr', None, None), ('changecount', None, None), ('desc',

fsep = '|'

label = 'svn'

last_commit
Get and parse the most recent Repository revision

Returns Repository log tuple

Return type tuple

log(*n=None*, *template=None*, ***kwargs*)
Run the repository log command

Returns output of log command (`svn log -l <n> <--kwarg=value>`)

Return type str

lsep = '——\n'

prefix = '.svn'

remote_url
Determine the primary remote url for this Repository

Returns primary remote url for this Repository (`svn info | grep "^Repository Root:"`)

Return type str

search_upwards(*fpath=None*, *repodirname='.svn'*, *upwards={}*)
Traverse filesystem upwards, searching for .svn directories with matching UUIDs (Recursive)

Parameters

- **fpath** (*str*) – file path to search upwards from
- **repodirname** (*str*) – directory name to search for (`.svn`)
- **upwards** (*dict*) – dict of already-searched directories

example

repo/.svn repo/dir1/.svn repo/dir1/dir2/.svn

>> search_upwards('repo/') << 'repo/' >> search_upwards('repo/dir1') << 'repo/' >> search_upwards('repo/dir1/dir2') << 'repo/'

repo/.svn repo/dirA/ repo/dirA/dirB/.svn

>> search_upwards('repo/dirA') << 'repo/' >> search_upwards('repo/dirA/dirB') >> 'repo/dirB')

status
Run the repository status command and return stdout

Returns stdout output of `svn status` command

Return type str

template = None

unique_id
Determine a "unique id" for this repository

Returns Repository UUID of this repository

Return type str

class pyrpo.pyrpo.**cached_property**(*func*, *name=None*, *doc=None*)
Bases: object

Decorator that converts a function into a lazy property. The function wrapped is called the first time to retrieve the result and then that calculated result is used the next time you access the value:

```
class Foo(object):

    @cached_property
    def foo(self):
        # calculate something important here
        return 42
```

The class must have a *__dict__* (e.g. be a subclass of object)

Copyright BSD

see: https://github.com/mitsuhiko/werkzeug/blob/master/werkzeug/utils.py

pyrpo.pyrpo.**do_repo_report**(*repos*, *report='full'*, *output=<open file '<stdout>', mode 'w'>*, **args*, ***kwargs*)
Do a repository report: call the report function for each Repository

Parameters

- **repos** (*iterable*) – iterable of Repository instances
- **report** (*string*) – report name
- **output** (*writeable*) – output stream to print to

Yields Repository subclass

pyrpo.pyrpo.**do_tortoisehg_report**(*repos*, *output*)
Generate a thg-reporegistry.xml file from a list of repos and print to output

Parameters

- **repos** (*iterable*) – iterable of Repository subclass instances
- **output** (*writeable*) – output stream to which THG XML will be printed

pyrpo.pyrpo.**dtformat**(*time*)

pyrpo.pyrpo.**find_find_repos**(*where*, *ignore_error=True*)
Search for repositories with GNU find

Parameters

- **where** (*str*) – path to search from
- **ignore_error** (*bool*) – if False, raise Exception when the returncode is not zero.

Yields Repository subclass instance

pyrpo.pyrpo.**find_unique_repos**(*where*)
Search for repositories and deduplicate based on `repo.fpath`

Parameters **where** (*str*) – path to search from

Yields Repository subclass

pyrpo.pyrpo.**get_option_parser**()
Build an `optparse.OptionParser` for pyrpo commandline use

pyrpo.pyrpo.**itersplit**(*s*, *sep=None*)
Split a string by `sep` and yield chunks

Parameters

- **s** (*str-type*) – string to split
- **sep** (*str-type*) – delimiter to split by

Yields *generator of strings* – chunks of string s

pyrpo.pyrpo.**itersplit_to_fields**(*str_*, *fsep=' ||| '*, *revtuple=None*, *fields=[]*, *preparse=None*)
Itersplit a string into a (named, if specified) tuple.

Parameters

- **str** (*str*) – string to split
- **fsep** (*str*) – field separator (delimiter to split by)
- **revtuple** (*object*) – namedtuple (or class with a `._fields` attr) (optional)
- **fields** (*list of str*) – field names (if revtuple is not specified)
- **preparse** (*callable*) – function to parse str with before itersplitting

Returns fields as a tuple or revtuple, if specified

Return type tuple or revtuple

pyrpo.pyrpo.**listdir_find_repos**(*where*)
Search for repositories with a stack and `os.listdir`

Parameters **where** (*str*) – path to search from

Yields Repository subclass instance

pyrpo.pyrpo.**main**()
pyrpo.main: parse commandline options with optparse and run specified reports

pyrpo.pyrpo.**sh**(*cmd*, *ignore_error=False*, *cwd=None*, **args*, ***kwargs*)
Execute a command with subprocess.Popen and block until output

Parameters

- **cmd** (*tuple or str*) – same as subprocess.Popen args
- **ignore_error** (*bool*) – if False, raise an Exception if p.returncode is not 0
- **cwd** (*str*) – path to current working directory

Returns command execution stdout

Return type str

Raises Exception – if ignore_error is true and returncode is not zero

---

Note: this executes commands with `shell=True`: careful with shell-escaping.

4.1.3 Module contents

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/westurner/pyrpo/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

5.1.4 Write Documentation

pyrpo could always use more documentation, whether as part of the official pyrpo docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/westurner/pyrpo/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *pyrpo* for local development.

1. Fork the *pyrpo* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pyrpo.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pyrpo
$ cd pyrpo/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pyrpo tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/westurner/pyrpo/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_pyrpo
```

CHAPTER 6

Credits

- Wes Turner – https://github.com/westurner

CHAPTER 7

History

```
git log --format='* %s [%h]' master..develop
# [ ... ]
```

7.1 0.2.1 (2015-05-24)

- BUG: pyrpo.py: sh_full report: `cat > %r << EOF` [91d5fb7]
- BUG,CLN: pyrpo.py: logname='pyrpo' [e029abe]
- BLD: Makefile: pull, push, BROWSERCMD lookups [59cbc66]
- BLD: Makefile: twine [d636e15]

7.2 0.2.0 (2015-04-25)

- Development: https://github.com/westurner/pyrpo/commits/develop
- Master: https://github.com/westurner/pyrpo/commits/master

7.3 0.1.0 (2014-05-12)

- First release on PyPI.
- Re-packaged from https://github.com/westurner/dotfiles/blob/2813e4ad/scripts/repos.py

CHAPTER 8

License

Copyright (c) 2014, Wes Turner All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of pyrpo nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

p

Symbols

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U