chut Documentation

Release 0.3

Gael Pasgrimaud

Nov 08, 2018

Contents

Chut!

Chut is a small tool to help you to interact with shell pipes and commands.

Basically it will help to write some shell script in python

This is more like a toy than a real tool but... It may be useful sometimes.

It's tested with py2.6+ and py3.2+:

Full documentation can be found here

CHAPTER 1

Quick quick start

Get the chutify script:

```
$ wget https://raw.github.com/gawel/chut/master/docs/_static/binaries/chutify
$ chmod +x chutify
```

Write a console script:

```
$ cat << EOF > myscript.py
from chut import *

__version__ = '0.1'

@console_script
def mycmd(args):
    """Usage: %prog [options] <directory>

    Print all chut scripts found in <directory>

    Options:

    %options
    """
    for filename in find('-name *.py') | grep('@console_script'):
        print(filename)
EOF
```

Run `chutify` in development mode:

```
$ ./chutify --devel
chmod +x bin/mycmd
```

And use/debug the newly created script:

```
$ ./bin/mycmd -h
```

When your script is ready for production then generate the standalone version:

```
$ ./chutify
chmod +x dist/scripts/mycmd
```

Also have a look at the examples.

CHAPTER 2

Installation

Using pip:

```
$ pip install chut
```

This will also install docopt and allow you to use the `@console_script` decorator.

Another option is to get chutify standalone version:

```
$ wget https://raw.github.com/gawel/chut/master/docs/_static/binaries/chutify
$ chmod +x chutify
```

CHAPTER 3

Quick start

Import the shell:

```
>>> import chut as sh
```

Get a file content if it contains “Chut”:

```
>>> grep_chut = sh.cat('README.rst') | sh.grep('Chut')
>>> if grep_chut:
...     print(grep_chut | sh.head("-n1"))
Chut!
```

Redirect output to a file:

```
>>> ret = (grep_chut | sh.head("-n1")) > '/tmp/chut.txt'
>>> ret.succeeded
True
>>> print(sh.cat('/tmp/chut.txt'))
Chut!
```

Or to stdout:

```
>>> sh.cat('/tmp/chut.txt') > 1
Chut!
```

Redirect stdout to stderr:

```
>>> sh.cat('/tmp/chut.txt') > 2
Chut!
```

Run many command with a pool of processes:

```
>>> [ret.succeeded for ret in sh.ls.map(['.', ['-l', '/tmp']])]
[True, True]
```

Use docopt to write a console script. This script will take an iface as argument and return a code 1 if no address is found:

```
>>> @sh.console_script
... def got_inet_addr(args):
...     """Usage: got_inet_addr <iface>"""
...     if sh.ifconfig(args['<iface>']) | sh.grep('inet addr:'):
...         return 1
```

CHAPTER 4

Documentation

4.1 Chut's basics

4.1.1 About imports

You can import chut and use commands:

```
>>> import chut as sh
>>> sh.ls()
'ls'
```

You can import sudo:

```
>>> from chut import sudo
```

But you can also import some specific commands:

```
>>> from chut import cat, grep, gzip, gunzip
>>> from chut.sudo import ifconfig
```

Or import "all" commands. Where "all" is a unexaustive set of commands:

```
>>> from chut import *
```

See *Imports*

4.1.2 Exceptions

The `cd` command use python `os.chdir()`

Some commands do not use a pipe by default. This mean that they are executed immediately. See *Imports*

By default a command is piped. But you can avoid this:

```
>>> sh.ls(pipe=False)
'ls'
```

By default a command do not use a shell. But if you need you can use one:

```
>>> sh.ls(shell=True)
'ls'

>>> sh.ls(sh=True)
'ls'
```

4.1.3 Aliases

You can define some aliases:

```
>>> sh.aliases['ll'] = '/usr/local/bin/ls -l'
>>> sh.aliases['python'] = '/opt/python3/bin/python3'
>>> print(repr(sh.ll('.')))
'/usr/local/bin/ls -l .'
>>> print(repr(sh.python('-c "import sys"')))
"/opt/python3/bin/python3 -c 'import sys'"
```

4.1.4 Environ

Chut use a copy of `os.environ` but you can modify values:

```
>>> from chut import env
>>> env.path = '/usr/bin:/usr/local/bin'
>>> env.path
['/usr/bin', '/usr/local/bin']
>>> env.path = ['/usr/bin', '/usr/local/bin']
>>> env.path
['/usr/bin', '/usr/local/bin']
>>> env.path += ['bin']
>>> env.path
['/usr/bin', '/usr/local/bin', 'bin']
```

Only `path` return a list. Other values return a string.

You can also pass a copy to your commands:

```
>>> env = sh.env.copy()
>>> sh.cat('-', env=env)
'cat -'
```

The environment can also be temporarily modified with a "with" statement. In this example, "HOME" is modified only inside the "with" block and restored at the end:

```
>>> with sh.env(HOME="/home/foo"):
...     str(sh.echo("$HOME", sh=True))
...
'/home/foo'
```

4.1.5 The test command

You can use the test command:

```
>>> from chut import test

>>> # test -f chut/recipe.py
>>> bool(test.f('chut/recipe.py'))
True

>>> # test -x chut/recipe.py
>>> if test.x('chut/recipe.py'):
...     print('chut/recipe.py is executable')
```

4.1.6 Logging

Chut provide logging facility:

```
>>> import sys
>>> log = sh.logopts(fmt='brief', stream=sys.stdout)
>>> log.info('info message')
```

When logging is configured you can use those simple functions:

```
>>> from chut import debug, info, error
>>> info('info message')
>>> debug('debug message')
>>> error('error message')
```

Notice that if you use `%options` in a `console_script` docstring then you don't need to use `logopts`. The decorator will do the job for you.

4.1.7 Run a large amount of processes

You can use the *chut.Pipe.map()* method to run a large amount of commands with the same binary. Arguments must be a list of string or list:

```
>>> results = sh.ls.map(['.', ['-l', '.']])
>>> [res.succeeded for res in results]
[True, True]
```

4.1.8 Debugging

You can print your pipe:

```
>>> print(repr(cat('README.txt') | grep('Chut')))
'cat README.txt | grep Chut'
```

You can also activate logging:

```
>>> sh.set_debug()
>>> print(cat('README.rst') | grep('Chut') | sh.head('-n1'))
Popen(['cat', 'README.rst'], **{...})
```

(continues on next page)

(continued from previous page)

```
Popen(['grep', 'Chut'], **{...})
Popen(['head', '-n1'], **{...})
Chut!
```

Cheers.

4.2 Pipes

A *Pipe* is a group of one or more command. By default all commands are lazy. It mean that they are not executed until you want a result (output).

You can always check what a *Pipe* will execute with `print(repr(pipe))`

4.2.1 The pipe context manager

A context manager can help you to check for some errors:

```
>>> from chut import cat, grep

>>> with sh.pipes(cat('fff') | grep('fff')) as p:
...     print(p)
Traceback (most recent call last):
...
OSError: cat: fff: No such file or directory
```

Basically it checks return codes at the end of the pipe

4.2.2 Use predefined pipe

Define an pipe:

```
>>> chut = cat('README.rst') | grep('chut')
```

And use it:

```
>>> chut | sh.head('-n1')
'cat README.rst | grep chut | head -n1'
```

The original defined pipe stay as this (everything is copied):

```
>>> chut
'cat README.rst | grep chut'
```

You can also extract parts of the pipe using slices:

```
>>> chut[1:]
'grep chut'
```

4.2.3 Use python !!

You can use some python code ad the end of the pipe (and only at the end):

```
>>> @sh.wraps
... def check_chut(stdin):
...     for line in stdin:
...         if line.startswith(b'Chut'):
...             yield b'Chut rocks!\n'
...             break

>>> with sh.pipes(cat('README.rst') | check_chut) as cmd:
...     for line in cmd:
...         print(line)
Chut rocks!
```

4.2.4 Access binaries outside of PATH

You can use `pipe` to get some binaries:

```
>>> sh.pipe('/opt/foo/bin/bar --help')
'/opt/foo/bin/bar --help'
```

You can also use getitem:

```
>>> sh['/opt/foo/bin/bar']('--help')
'/opt/foo/bin/bar --help'
```

Or getattr...:

```
>>> getattr(sh, '/opt/bar/bin/foo')('--help')
'/opt/bar/bin/foo --help'
```

4.3 Sudo

You can for sure use sudo:

```
>>> from chut import sudo
>>> sudo.ls() | sudo.grep('chut')
'/usr/bin/sudo -s ls | /usr/bin/sudo -s grep chut'
```

Sudo wont work with ssh except if it does not require a password on the server side.

4.4 SSH

The ssh command take a host first and is gziped by default:

```
>>> import chut as sh
>>> from chut import gzip
>>> from chut import ssh
>>> srv1 = ssh('gawel@srv')
>>> srv1.ls('~')
"ssh gawel@srv 'ls ~'"
```

For example you can backup your mysql database locally:

```
>>> srv1.mysqldump('db | gzip') | gzip
"ssh gawel@srv 'mysqldump db | gzip' | gzip"
```

Or on another server:

```
>>> srv2 = ssh('gawel@srv2')
>>> srv1(sh.mysqldump('db') | gzip | srv2('gunzip > ~/backup.db'))
'ssh gawel@srv "mysqldump db | gzip | ssh gawel@srv2 \'gunzip > ~/backup.db\'"'
```

You can use your ssh instance to get some remote file:

```
>>> sh.rsync(srv1.join('~/p0rn'), '.', pipe=True)
'rsync gawel@srv:~/p0rn .'
```

4.5 Internals

4.5.1 Imports

Those elements are imported when you use `from chut import *`:

```
from subprocess import PIPE
from subprocess import STDOUT
from copy import deepcopy
from ConfigObject import ConfigObject
from contextlib import contextmanager

try:
    from fabric import api as fabric
except ImportError:
    HAS_FABRIC = False
else:  # pragma: no cover
    if 'nosetests' in sys.argv[0]:
        HAS_FABRIC = False
    else:
        HAS_FABRIC = True
```

Also noticed that commands which don't use pipes are listed here.

4.5.2 Pipe

class chut.**Pipe**(**args*, ***kwargs*)

A pipe object. Represent a set of one or more commands.

bg()

Run processes in background. Return the last piped Popen object

failed

True if one or more process failed

classmethod map(*args*, *pool_size=None*, *stop_on_failure=False*, ***kwargs*)

Run a batch of the same command and manage a pool of processes for you

returncodes

A list of return codes of all processes launched by the pipe

stderr
combined stderr of all processes

stdout
standard output of the pipe. A file descriptor or an iteraror

succeeded
True if all processes succeeded

4.5.3 Environ

class chut.**Environ**
Manage os.environ

copy() → a shallow copy of D

4.5.4 Input

You can use a python string as input:

```
>>> print(sh.stdin(b'gawel\nfoo') | grep('gawel'))
gawel
```

The input can be a file but the file is not streamed by `stdin()`. Notice that the file must be open in binary mode (`rb`):

```
>>> print(sh.stdin(open('README.rst', 'rb'))
...                | grep('Chut') | sh.head('-n1'))
Chut!
```

class chut.**Stdin**(*value*)
Used to inject some data in the pipe

4.5.5 Output

You can get the output as string (see *Stdout*):

```
>>> output = str(cat('README.rst') | grep('Chut'))
>>> output = (cat('README.rst') | grep('Chut'))()
```

As an iterator (iterate over each lines of the output):

```
>>> chut_stdout = cat('README.rst') | grep('Chut') | sh.head('-n1')
```

And can use some redirection:

```
>>> ret = chut_stdout > '/tmp/chut.txt'
>>> ret.succeeded
True
>>> print(cat('/tmp/chut.txt'))
Chut!

>>> ret = chut_stdout >> '/tmp/chut.txt'
>>> ret.succeeded
```

(continues on next page)

(continued from previous page)

```
True
>>> print(cat('/tmp/chut.txt'))
Chut!
Chut!
```

Parentheses are needed with >> (due to the way the python operator work):

```
cat('README.rst') | grep >> '/tmp/chut.txt' # wont work
(cat('README.rst') | grep) >> '/tmp/chut.txt' # work
```

class chut.**Stdout**
A string with extra attributes:

- succeeded
- failed
- stdout
- stderr

4.5.6 Ini files

chut.**ini**(*filename*, ***defaults*)
Load a .ini file in a ConfigObject. Dont raise if the file does not exist

Example:

```
>>> from chut import ini
>>> config = ini('/tmp/chut.ini')
>>> config.my = dict(key='value')
>>> config.write()
>>> config = ini('/tmp/chut.ini')
>>> print(config.my.key)
value
```

4.6 Generate scripts

Chut allow you to generate some standalone scripts. Scripts will include chut and docopts (and a few other modules if you want) encoded in base64.

4.6.1 How it works

Write a file with a function in it:

```
>>> ch.stdin(b'''
... import chut as ch
... @ch.console_script
... def my_script(arguments):
...     """Usage: %prog [-h]
...
...     -h, --help    Print this help
...     """
```

(continues on next page)

(continued from previous page)

```
...     print('Hello world')
... ''') > 'myscript.py'
''
```

Then run chutify on it:

```
>>> print(ch.chutify('myscript.py', combine_stderr=True))
chmod +x dist/scripts/my-script
```

And check the result in `dist/scripts`:

```
>>> bool(test.x('dist/scripts/my-script'))
True

>>> print(ch.pipe('dist/scripts/my-script'))
Hello world

>>> print(ch.pipe('dist/scripts/my-script', '-h'))
Usage: my-script [-h]

-h, --help    Print this help
```

4.7 Using chut with Fabric

Chut contains some helpers to generate and use chut scripts with Fabric.

class chut.**Fab**

chutifab(**args*)
Generate chut scripts contained in location

run(*script*, **args*, ***kwargs*)
Upload a script and run it. `*args` are used as command line arguments. `**kwargs` are passed to *fabric*'s *run*

sudo(*script*, **args*, ***kwargs*)
Upload a script and run it using sudo. `*args` are used as command line arguments. `**kwargs` are passed to *fabric*'s *sudo*

Here is a sample `fabfile.py`

```
# -*- coding: utf-8 -*-
from fabric.api import env
from chut import fab

env.forward_agent = True

fab.chutifab()


def upgrade():
    fab.run('rfsync', '-h')
    fab.sudo('rfsync', '-h')
```

4.8 Examples

4.8.1 example

```
# -*- coding: utf-8 -*-
__doc__ = """Generate this page"""
from chut import *  # noqa

TEMPLATE = '''
%(binary)s
=========================================================

.. literalinclude:: ../chut/examples/%(filename)s
   :language: python

Get standalone `%(binary)s <_static/binaries/%(binary)s>`_

'''


@console_script
def example(args):
    fd = open('docs/examples.rst', 'w')
    fd.write((
        '==========================\n'
        'Examples\n'
        '==========================\n\n'))
    for filename in sorted(find('chut/examples -name *.py')):
        try:
            scripts = list(grep('-A1 -E @.*console_script', filename))
        except OSError:
            continue
        if not scripts:
            continue
        filename = path.basename(filename)
        scripts = [s[4:].split('(')[0] for s in scripts if s[0] != '@']
        binary = scripts[0].replace('_', '-')
        fd.write(TEMPLATE % dict(filename=filename, binary=binary))
    fd.close()
```

Get standalone example

4.8.2 github-clone

```
# -*- coding: utf-8 -*-
from chut import *


@console_script
def github_clone(args):
    """Usage: %prog [<user>]

    Clone all github repository for a user using ssh
    """
    user = args['<user>'] or 'gawel'
```

(continues on next page)

(continued from previous page)

```
    page = wget('-O- https://github.com/%s?tab=repositories' % user)
    for line in page | grep('href="/%s/' % user) | grep('-v title='):
        repo = line.split('"')[1]
        name = repo.split('/', 1)[-1]
        if not test.d(name) and name not in ('followers', 'following'):
            try:
                git('clone', 'git@github.com:%s' % repo.strip('/')) > 1
            except OSError:
                pass

if __name__ == '__main__':
    github_clone()
```

Get standalone github-clone

4.8.3 rfsync

```
# -*- coding: utf-8 -*-
from chut import *  # NOQA
import sys


@console_script(fmt='msg')
def rfsync(args):
    """
    Usage: %prog [-p] <host>:<path> [-- <find_options>...]
           %prog [-] [<destination>] [-- <rsync_options>...]
           %prog -h

    Find some files on a remote server and sync them on a local directory using
    rsync

    Examples:

        $ rfsync gawel@example.com:~/ -- -name "*.avi" | rfsync
        $ rfsync gawel@example.com:~/ -- -size +100M | rfsync ~/Movies -- -q

    """
    remote = args.get('<host>:<path>')
    if remote not in (None, '-') and ':' in remote:
        host, p = remote.split(':')
        srv = ssh(host)
        options = []
        for a in args.get('<find_options>', []) or []:
            if '*' in a:
                a = '"%s"' % a
            options.append(a)
        options = ' '.join(options)
        done = set()
        for line in srv.find(p, options, shell=True):
            line = line.strip()
            if args['-p']:
                line = path.dirname(line)
            if line not in done:
                done.add(line)
```

(continues on next page)

(continued from previous page)

```
                print(srv.join(line))
    else:
        destination = args['<host>:<path>'] or args['<destination>'] or '.'
        destination = path.expanduser(path.expandvars(destination))
        options = ' '.join(args.get('<rsync_options>', [])) or '-aP'
        targets = sys.stdin.readlines()
        targets = [t.strip('\n/') for t in targets]
        targets = [t.strip('/ ') for t in targets if t.strip('/ ')]
        targets = sorted(set(targets))
        if not targets:
            return 1
        if '-q' not in options:
            info('$ rsync %s \\\n\t%s \\\n\t%s',
                 options,
                 ' \\\n\t'.join(targets),
                 destination)
        rsync(options, ' '.join(targets), destination, shell=True) > 1
```

Get standalone rfsync

4.8.4 ssh-copy-id

```
from chut import * # NOQA


@console_script
def ssh_copy_id(args):
    """
    Usage: %prog <host>
           %prog <pubkey> <host>
    """
    stdin = None
    pubkey = args['<pubkey>']

    if not pubkey and env.SSH_AUTH_SOCK:
        ret = str(sh['ssh-add']('-L', combine_stderr=True))
        if ret.succeeded:
            stdin = stdin(ret.strip() + '\n')

    if stdin is None:
        if not pubkey:
            pubkey = path.expanduser('~/.ssh/id_rsa.pub')
        if not test.e(pubkey):
            print('Cant find a valid key')
            return 1
        stdin = cat(pubkey)

    srv = ssh(args['<host>'])
    if stdin | srv(("umask 077; test -d .ssh || mkdir .ssh;"
                    "cat >> .ssh/authorized_keys")):
        print('Key added to %s' % (args['<host>'],))
        print('Trying to cat %s~/.ssh/authorized_keys...' % srv)
        srv.cat('~/.ssh/authorized_keys') | tail('-n1') > 1
        return 0
    print('Failed to add key')
```

(continues on next page)

(continued from previous page)

```
    return 1

if __name__ == '__main__':
    ssh_copy_id()
```

Get standalone ssh-copy-id

4.8.5 ssh-mount

```
# -*- coding: utf-8 -*-
from chut import *  # noqa

__version__ = "0.17"


@console_script(fmt='msg')
def ssh_mount(args):
    """
    Usage: %prog [options] [<server>] [<mountpoint>]

    Use sshfs/fusermount to mount/umount your remote servers

    By default ~ is mounted but you can set the mountpoints in ~/.ssh/sshfs:

    [mountpoints]
    myserver = ./path

    Options:
    -u, --umount        Umount
    %options
    """
    env.lc_all = 'C'
    server = args['<server>']
    if not server:
        if not args['--umount']:
            sh.mount() | grep('--color=never fuse.sshfs') > 1
        else:
            for line in sh.mount() | grep('--color=never fuse.sshfs'):
                dirname = line.split(' ')[2]
                info('umount %s', dirname)
                sh.fusermount('-u', dirname) > 0
        return 0
    dirname = path.expanduser('~/mnt/%s' % server)
    if sh.mount() | grep(dirname):
        info('umount %s', dirname)
        sh.fusermount('-u', dirname) > 0
    if args['--umount']:
        return 0
    if not test.d(dirname):
        mkdir('-p', dirname)
    cfg = ini('~/.ssh/sshfs')
    mountpoint = '%s:%s' % (
        server,
        args['<mountpoint>'] or cfg['mountpoints'][server] or '.'
    )
```

(continues on next page)

(continued from previous page)

```
    info('mount %s to %s', mountpoint, dirname)
    sh.sshfs(mountpoint, dirname) > 1
```

Get standalone ssh-mount

4.8.6 safe-upgrade

```
# -*- coding: utf-8 -*-
from chut import * # noqa


@console_script
def safe_upgrade(args):
    """Usage: %prog

    Update && upgrade a debian based system
    """
    sudo.aptitude('update') > 1
    sudo.aptitude('safe-upgrade') > 1
```

Get standalone safe-upgrade

4.8.7 translate

```
# -*- coding: utf-8 -*-
from chut import *  # NOQA
import atexit
import six
import sys
import os

__version__ = "0.17"

__doc__ = """
Usage: %prog [options] [-] [<text>...]

This script use chut and casperjs to build an interactive translator

Examples:

    $ echo "hello" | translate -
    $ translate -l fr:en bonjour
    $ translate -i

Options:

    -l LANGS, --langs=LANGS     Langs [default: en:fr]
    -i, --interactive           Translate line by line in interactive mode
    -h, --help                  Show this help
    %options
"""

SCRIPT = six.b("""
```

(continues on next page)

(continued from previous page)

```
system = require('system')
require('casper').create()
  .start('http://translate.google.com/#' + system.env['TR_PAIR'], function(){
      this.fill('form#gt-form', {text: system.env['TR_TEXT']}, false)})
  .waitForSelector('span.hps', function() {
      this.echo(this.fetchText('#result_box'))
      results = this.evaluate(function() {
          results = document.querySelectorAll('table.gt-baf-table tr')
          return Array.prototype.map.call(results, function(e) {
              if (!/colspan/.exec(e.innerHTML))
                  return e.innerText.replace(/\\n/,': ')
              else
                  return ''
          })
      })
      this.echo(results.join(''))
}).run()
""")


@console_script(doc=__doc__)
def translate(args):
    if not which('casperjs'):
        print('You must install casperjs first')
        return 1
    env.tr_pair = args['--langs'].replace(':', '|')
    script = str(mktemp('--tmpdir translate-XXXX.js'))
    atexit.register(rm(script))
    stdin(SCRIPT) > script

    def show_result():
        try:
            for line in sh.casperjs(script):
                if line:
                    if ':' in line:
                        line = '- ' + line
                    print(line)
        except:
            pass

    if args['--interactive'] or not (args['-'] or args['<text>']):
        import readline
        hist = os.path.join(os.path.expanduser('~'), '.translate_history')
        if test.f(hist):
            readline.read_history_file(hist)
        atexit.register(readline.write_history_file, hist)
        while True:
            try:
                tr_text = six.moves.input('%s: ' % env.tr_pair)
                tr_text = tr_text.strip()
            except KeyboardInterrupt:
                return
            else:
                if tr_text == 'q':
                    return
                elif tr_text == 's':
                    env.tr_pair = '|'.join(reversed(env.tr_pair.split('|')))
```

(continues on next page)

(continued from previous page)

```
                elif tr_text.startswith(('lang ', 'l ')):
                    tr_pair = tr_text.split(' ', 1)[1]
                    env.tr_pair = tr_pair.replace(':', '|').strip()
                elif tr_text:
                    env.tr_text = tr_text
                    show_result()
    elif args['-']:
        env.tr_text = sys.stdin.read()
    elif args['<text>']:
        env.tr_text = ' '.join(args['<text>'])
    show_result()

if __name__ == '__main__':
    translate()
```

Get standalone translate

4.8.8 vlserie

```
# -*- coding: utf-8 -*-
from chut import *  # noqa
import shutil
import sys
import re

__version__ = "0.17"

_episode = re.compile(
    r'[^0-9]+(?P<s>[0-9]+)\s*(x|e|episode)\s*(?P<e>[0-9]+)[^0-9]+')


def extract_numbers(f):
    m = _episode.search(f.lower())
    if m:
        m = m.groupdict()
        return int(m['s']), int(m['e'])


@console_script(fmt='brief')
def vlserie(args):
    """
    Usage: %prog [options] [<season> <episode>]

    Play the serie contained in the current folder. File names should be
    formated like SXXEXX. Also load subtitles if any.

    Store the latest play in ~/.vlserie. So you dont have to remember it
    yourself.

    Require vlc or mplayer.

    Options:

    -s TIME, --start=TIME   Start at (float)
    -l, --latest            Play latest instead of next
```

(continues on next page)

(continued from previous page)

```
    -f, --freeplayer        Play in freeplayer
    --loop                  Loop over episodes
    %options
    """

    config = ini('~/.vlserie')
    config.write()

    player = config.player.binary or 'vlc'
    player_opts = config[player]
    if env.display:
        options = player_opts.xoptions
    else:
        options = player_opts.fboptions
    if args['--start']:
        options += ' --start-time ' + args['--start']
    debug('Using %s player', player)

    pwd = path.abspath('.')

    def play(filename, episode):
        filename = path.abspath(filename)
        dirname, filename = path.split(filename)
        cd(dirname)
        if args['--freeplayer']:
            cmdline = (
                "%s %r --play-and-exit --sout "
                "'#transcode{vcodec=mp2v,vb=4096,scale=1,audio-sync,soverlay}:"
                "duplicate{dst=std{access=udp,mux=ts,dst=212.27.38.253:1234}}'"
            ) % (options, filename)
        elif player in ('vlc', 'cvlc'):
            cmdline = (
                '%s -f --play-and-exit --qt-minimal-view %r'
            ) % (options, filename)
        elif player == 'mplayer':
            cmdline = '%s -fs %r' % (options, filename)
        else:
            error('Unknown player %r', player)
            sys.exit(1)
        srts = find(pwd, '-iregex ".*%s\(x\|E\)%02i.*srt"' % episode,
                    shell=True)
        for srt in sorted(srts):
            if '  ' in srt:
                new = srt.replace('  ', ' ')
                shutil.move(srt, new)
                srt = new
            if player in ('vlc', 'cvlc'):
                cmdline += ' --sub-file %r' % srt
            elif player == 'mplayer':
                cmdline += ' -sub %r' % srt
        subs = find(pwd, '-iregex ".*%s\(x\|E\)%02i.*sub"' % episode,
                    shell=True)
        for sub in sorted(subs):
            if player == 'mplayer':
                sub = sub.lstrip('./')
                cmdline += ' -vobsub %r' % sub[:-4]
        cmd = sh[player](cmdline, combine_stderr=True, shell=True)
```

(continues on next page)

(continued from previous page)

```
        info(repr(cmd))
        serie = config[pwd]
        serie.latest = filename
        config.write()
        try:
            cmd > 1
        except OSError:
            pass
        if not args['--loop']:
            sys.exit(0)

    serie = config[pwd]

    filenames = find(
        ('. -iregex '
         '".*[0-9]+\s*\(e\|x\|episode\)\s*[0-9]+.*'
         '\(avi\|wmv\|mkv\|mp4\)"'),
        shell=True)
    filenames = [path.abspath(f) for f in filenames]
    filenames = sorted([(extract_numbers(f), f) for f in filenames])
    filenames = [(e, f) for e, f in filenames if f is not None]

    if args['<season>']:
        episode = int(args['<season>']), int(args['<episode>'])
        filenames = [(x, f) for x, f in filenames if x >= episode]
    elif serie.latest:
        episode = extract_numbers(serie.latest.lower())
        if args['--latest']:
            filenames = [(x, f) for x, f in filenames if x >= episode]
        else:
            filenames = [(x, f) for x, f in filenames if x > episode]

    for episode, filename in filenames:
        play(filename, episode)


@console_script(fmt='brief')
def freeplayer(args):
    """Usage: %prog [options] [<stream>]

    -s      Serve freeplayer page
    """
    if args["-s"]:
        with open('/tmp/settings.html', 'w') as fd:
            fd.write(settings)
        nc('-l -p 8080 -q 1 < /tmp/settings.html', shell=True) > 0
        info('freeplayer initialized')
        return
    stream = args['<stream>']
    if stream.startswith('https://'):
        stream.replace('https://', 'http://')
    cmdline = (
        "%r --play-and-exit --sout "
        "'#transcode{vcodec=mp2v,vb=4096,scale=1,audio-sync,soverlay}:"
        "duplicate{dst=std{access=udp,mux=ts,dst=212.27.38.253:1234}}'"
    ) % stream
    cmd = sh['vlc'](cmdline, combine_stderr=True, shell=True)
```

(continues on next page)

(continued from previous page)

```
    info(repr(cmd))
    cmd > 1

settings = '''HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
<html><body background="ts://127.0.0.1"></body></html>'''
```

Get standalone vlserie

4.8.9 webapp

```
# -*- coding: utf-8 -*-
from chut import * # noqa
requires('webob', 'waitress')
import webob


def application(environ, start_response):
    req = webob.Request(environ)
    resp = webob.Response()
    resp.text = req.path_info
    return resp(environ, start_response)


@console_script
def webapp(args):
    """
    Usage: %prog [--upgrade-deps]
    """
    serve(application, port=4000)
```

Get standalone webapp

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index