dotfiles Documentation

Release 0.6.4

Wes Turner

November 14, 2014

Contents

CHAPTER 1

Dotfiles

GitHub | Documentation | ReadTheDocs

1.1 Goals

- Streamline frequent workflows
- Configure Bash, ZSH, and Vim
- Configure Python, pip, virtualenv, virtualenvwrapper
- Configure IPython
- Configure Gnome
- Configure i3wm
- Support Debian Linux, Ubuntu Linux, OSX
- Document commands and keyboard shortcuts
- Create a productive working environment

1.2 Usage

- `scripts/bootstrap_dotfiles.sh` installs symlinks in `$HOME` (such as `~/.bashrc -> ${__DOTFILES}/etc/bashrc`)
- `etc/bashrc` sources `etc/bash/00-bashrc.before.sh`
- `etc/bash/00-bashrc.before.sh` sources a documented, ordered sequence of Bash scripts
- `etc/zsh/00-zshrc.before.sh` sources a documented, ordered sequence of ZSH scripts

See: Usage and Venv for documentation.

1.2.1 Examples

Installing the dotfiles

```
# clone and install dotfiles and dotvim
# with venv-style paths (_SRC, _APP, _WRD)

# WORKON_HOME       -- base path for virtualenvs [virtualenvwrapper]
WORKON_HOME="~/wrk/.ve"
test -d ${WORKON_HOME} || mkdir -p ${WORKON_HOME}

# VIRTUAL_ENV_NAME  -- basename for VIRTUAL_ENV [venv]
VIRTUAL_ENV_NAME="dotfiles"

# VIRTUAL_ENV       -- current virtualenv prefix [virtualenv]
VIRTUAL_ENV="${WORKON_HOME}/${VIRTUAL_ENV_NAME}"
_SRC="$[VIRTUAL_ENV}/src"

# _APP              -- basename of current working directory [venv]
_APP="dotfiles"
# _WRD              -- working directory [venv]
_WRD="${_SRC}/${_APP}"      # ${WORKON_HOME}/dotfiles/src/dotfiles

git clone https://github.com/westurner/dotfiles $_WRD
git clone https://github.com/westurner/dotvim ${_WRD}/etc/vim
cd $_WRD                    # cdw [venv]

__DOTFILES="${HOME}/.dotfiles"
ln -s $_WRD $__DOTFILES
ls ~/.dotfiles

$_WRD/scripts/bootstrap_dotfiles.sh -h
$_WRD/scripts/bootstrap_dotfiles.sh -I      # or: make install

# TODO: move / replicate this in bootstrap_dotfiles.sh
```

1.2.2 Bash

Source: https://github.com/westurner/dotfiles/tree/master/etc/bash
Documentation: https://westurner.github.io/dotfiles/usage.html#bash

```
# There should be a symlink from ~/.dotfiles
# to the current dotfiles repository.
# All dotfiles symlinks are relative to ${__DOTFILES}.
__DOTFILES="${HOME}/.dotfiles"
ls -ld $__DOTFILES || ln -s $_WRD $__DOTFILES

# There should be symlinks for each dotfile: e.g.
# ln -s ~/.dotfiles/etc/.bashrc ~/.bashrc
# ln -s ~/.dotfiles/etc/vim/vimrc ~/.vimrc
# ln -s ~/.dotfiles/etc/vim ~/.vim
# Create these symlinks with bootstrap_dotfiles.sh
$_WRD/scripts/bootstrap_dotfiles.sh -S      # or: make install_symlinks

source ~/.bashrc
# source dotfiles/etc/bash/00-bashrc.before.sh

dotfiles_status  # print dotfiles environment variables
ds               # print dotfiles environment variables
```

```
dotfiles_reload  # source ${__DOTFILES}/etc/bash/00-bashrc.before.sh
dr               # source ${__DOTFILES}/etc/bash/00-bashrc.before.sh
```

1.2.3 vimrc

Source: https://github.com/westurner/dotvim
Documentation: https://westurner.github.io/dotfiles/usage.html#vim

Vim configuration should be cloned to `${__DOTFILES}/etc/vim`.

```
make dotvim_clone dotvim_install
```

1.3 Installation

1.3.1 Requirements

Project requirements are installed by bootstrap_dotfiles.sh and, optionally, also the Makefile.

- *Bash*
- *Git*
- *Python* (*Pip*)

1.3.2 Install the dotfiles

Source: https://github.com/westurner/dotfiles
Documentation: https://westurner.github.io/dotfiles/

The bootstrap_dotfiles.sh shell script clones the dotfiles git repository and installs the `dotfiles` Python package.

Create a *Virtualenv* with *Virtualenvwrapper* named "`dotfiles`":

```
[sudo] pip install virtualenvwrapper
source $(which 'virtualenvwrapper.sh')
mkvirtualenv dotfiles
mkdir $VIRTUAL_ENV/src
cd $VIRTUAL_ENV/src
```

Install the dotfiles (symlink dotfiles into `$HOME`, install the dotfiles package, and install additional helpful packages):

```
git clone ssh://git@github.com/westurner/dotfiles && cd dotfiles
# Install and symlink dotfiles and dotvim
scripts/bootstrap_dotfiles.sh -I -R

# (Optional) Install dotfiles scripts into ~/.local/bin (pip --user)
scripts/bootstrap_dotfiles.sh -I -u
```

Note: See the Installing the dotfiles example, which uses venv-style paths.

1.3.3 Upgrade the dotfiles

```
# Check for any changes to symlinked dotfiles
cd ~/.dotfiles && git status && git diff

# Pull and upgrade dotfiles and dotvim (later)
scripts/bootstrap_dotfiles.sh -U
```

1.4 Further Dotfiles Resources

- https://dotfiles.github.io/
- https://westurner.github.io/wiki/workflow
- https://westurner.github.io/dotfiles/

Next: https://westurner.github.io/dotfiles/usage.html

CHAPTER 2

Usage

- *Install the dotfiles* with bootstrap_dotfiles.sh
- Develop with the Makefile (*Make*)
- Shell with Bash (*Bash*, *ZSH*)
- Edit text files with Vim (*Vim*)
- Manage windows on *Linux* platforms with I3wm (*I3wm*)
- *Script* all the things

2.1 bootstrap_dotfiles.sh

https://github.com/westurner/dotfiles/blob/master/scripts/bootstrap_dotfiles.sh

`bash scripts/bootstrap_dotfiles.sh -h`:

```
$ bash ../scripts/bootstrap_dotfiles.sh -h
## dotfiles_bootstrap -- a shell wrapper for cloning and installing
## Usage: bootstrap_dotfiles.sh <actions> <options>
#
## Actions
#  -I   --  install the dotfiles
#  -S   --  symlink dotfiles into place
#  -U   --  update and upgrade dotfiles
#  -R   --  pip install -r requirements-all.txt
#  -G   --  install gitflow and hubflow
#  -C   --  check
#  -h   --  print this help message
#
## Options
#  -u   --  pip install --user (modified for other actions)
#  -d   --  show debugging info (set -x)
```

2.2 Makefile

https://github.com/westurner/dotfiles/blob/master/Makefile

make help:

```
$ cd .. && make help
dotfiles Makefile
#################
help         -- print dotfiles help
help_vim     -- print dotvim make help
help_vim_txt -- print dotvim help as rst
help_i3      -- print i3wm configuration
help_setuppy -- print setup.py help
help_txt     -- print setup.py help as rst

install      -- install dotfiles and dotvim [in a VIRTUAL_ENV]
upgrade      -- upgrade dotfiles and dotvim [in a VIRTUAL_ENV]

install_user -- install dotfiles and dotvim (with 'pip --user')
upgrade_user -- upgrade dodtfiles and dotfile (with 'pip --user')

pip_upgrade pip              -- upgrade pip
pip_install_requirements_all -- install all pip requirements

install_gitflow -- install gitflow from github
install_hubflow -- install hubflow from github

install_brew_formulas  -- install brew formulas
update_brew_list       -- overwrite etc/brew/brew.list

clean  -- remove .pyc, .pyo, __pycache__/ etc
edit   -- edit the project main files README.rst
test   -- run tests
build  -- build a python sdist

docs      -- build sphinx documentation
gh-pages  -- overwrite the gh-pages branch with docs/_build/html
push      -- git push
pull      -- git pull
```

2.3 Bash

https://github.com/westurner/dotfiles/blob/master/etc/.bashrc
https://github.com/westurner/dotfiles/blob/master/etc/bash/00-bashrc.before.sh

make help_bash_rst:

```
#### etc/bash/00-bashrc.before.sh
## 00-bashrc.before.sh     -- bash dotfiles configuration root
#  source ${__DOTFILES}/etc/bash/00-bashrc.before.sh    -- dotfiles_reload()
#
  #  dotfiles_reload()  -- (re)load the bash configuration
  #  $__DOTFILES (str): -- path to the dotfiles symlink (~/.dotfiles)
  #
  ## 01-bashrc.lib.sh           -- useful bash functions (paths)
  #  lspath()           -- list every file along $PATH
  #  realpath()         -- readlink -f (python os.path.realpath)
  #  walkpath()         -- list every directory along ${1:-"."}
```

```
#
## 02-bashrc.platform.sh      -- platform things
#  detect_platform()  -- set $__IS_MAC or $__IS_LINUX
#
## 04-bashrc.TERM.sh          -- set $TERM and $CLICOLOR
#
## 05-bashrc.dotfiles.sh      -- dotfiles
#  $__DOTFILES (str): -- path to the dotfiles symlink (~/.dotfiles)
#  dotfiles_status()  -- print dotfiles variables
#  ds()               -- print dotfiles variables
#
## 06-bashrc.completion.sh    -- configure bash completion
#
#  python: pip, virtualenv, virtualenvwrapper
#  $PROJECT_HOME (str): path to project directory (~/wrk)
#  $WORKON_HOME  (str): path to virtualenvs directory (~/wrk/.ve)
#  $VIRTUAL_ENV  (str): path to current $VIRTUAL_ENV
#
## 07-bashrc.python.sh        -- python
#  _setup_anaconda()  -- setup anaconda paths (manual)
#  _setup_pyenv()     -- setup pyenv paths (manual)
#
## 07-bashrc.virtualenvwrapper.sh -- virtualenvwrapper
#
## 08-bashrc.gcloud.sh        -- gcloud: Google Cloud SDK
#  _setup_google_cloud()  -- setup google cloud paths
#
## 10-bashrc.venv.sh          -- venv: virtualenvwrapper extensions
#  $__PROJECTSRC     (str): script to source (${PROJECT_HOME}/.projectsrc.sh)
#  $VIRTUAL_ENV_NAME (str): basename of current $VIRTUAL_ENV
#  $_APP             (str): $VIRTUAL_ENV/src/${_APP}
#  we() -- workon a new venv
#     $1: VIRTUAL_ENV_NAME [$WORKON_HOME/${VIRTUAL_ENV_NAME}=$VIRTUAL_ENV]
#     $2: _APP (optional; defaults to $VIRTUAL_ENV_NAME)
#     we dotfiles
#     we dotfiles etc/bash; cdw; ds; ls
#
#
## 11-bashrc.venv.pyramid.sh  -- venv-pyramid: pyramid-specific config
#
## 20-bashrc.editor.sh        -- $EDITOR configuration
#  $_EDIT_ (str): cmdstring to open $@ (file list) in current editor
#  $EDITOR (str): cmdstring to open $@ (file list) in current editor
## 20-bashrc.vimpagers.sh     -- $PAGER configuration
#  $PAGER (str): cmdstring to run pager (less/vim)
#
## 30-bashrc.usrlog.sh        -- $_USRLOG configuration
#  $_USRLOG (str): path to .usrlog command log
#  stid       -- set $TERM_ID to a random string
#  stid $name -- set $TERM_ID to string
#  note       -- add a dated note to $_USRLOG [_usrlog_append]
#  usrlogv    -- open usrlog with vim:   $VIMBIN + $_USRLOG
#  usrlogg    -- open usrlog with gmvim: $GUIVIMBIN + $_USRLOG
#  usrloge    -- open usrlog with editor:$EDITOR + $_USRLOG
#  ut         -- tail $_USRLOG
#  ug         -- egrep current usrlog: egrep $@ $_USRLOG
#  ugall      -- egrep $@ $__USRLOG ${WORKON_HOME}/*/.usrlog
#  ugrin      -- grin current usrlog: grin $@ $_USRLOG
```

```
  #  ugrinall   -- grin $@  $__USRLOG ${WORKON_HOME}/*/.usrlog
  #  lsusrlogs  -- ls -tr   $__USRLOG ${WORKON_HOME}/*/.usrlog
  #
  ## 30-bashrc.xlck.sh          -- screensaver, (auto) lock, suspend
  #
  ## 40-bashrc.aliases.sh       -- aliases
  ## 42-bashrc.commands.sh      -- example commands
  #
  ## 50-bashrc.bashmarks.sh     -- bashmarks: local bookmarks
  #
  ## 70-bashrc.repos.sh         -- repos: $__SRC repos, docs
  #
  ## 99-bashrc.after.sh         -- after: cleanup
    # dr()  -- dotfiles_reload
    # ds()  -- print dotfiles_status()


#### etc/bash/01-bashrc.lib.sh
### bashrc.lib.sh
## bash
    # echo_args         -- echo $@ (for checking quoting)
    # function_exists() -- check whether a bash function exists
    # add_to_path       -- prepend a directory to $PATH
    #   instead of:
    #       export PATH=$dir:$PATH
    #       add_to_path $dir
    # lightpath()       -- display $PATH with newlines
    # lspath()          -- list files in each directory in $PATH
    # lspath_less()     -- lspath with less (color)
## file paths
    # realpath()        -- print absolute path (os.path.realpath) to $1
    #                      note: OSX does not have readlink -f
    # path()            -- realpath()
    # walkpath()        -- walk down path $1 and $cmd each component
    #   $1: path (optional; default: pwd)
    #   $2: cmd  (optional; default: 'ls -ald --color=auto')
    # ensure_symlink()  -- create or update a symlink to $2 from $1
    #                      if $2 exists, backup with suffix $3
    # ensure_mkdir()    -- create directory $1 if it does not yet exist


#### etc/bash/02-bashrc.platform.sh
### bashrc.platform.sh
    # detect_platform() -- set $__IS_MAC or $__IS_LINUX according to $(uname)


#### etc/bash/04-bashrc.TERM.sh
### bashrc.TERM.sh
    # configure_TERM            -- configure the $TERM variable (man terminfo)
    #   $1: (optional; autodetects if -z)
    # configure_TERM_CLICOLOR   -- configure $CLICOLOR and $CLICOLOR_256
    #   CLICOLOR=1
    # configure_TERM when sourced


#### etc/bash/05-bashrc.dotfiles.sh
### bashrc.dotfiles.sh
    # dotfiles_add_path()       -- add ${__DOTFILES}/scripts to $PATH
```

```
    # dotfiles_status()         -- print dotfiles_status
    # ds()                      -- print dotfiles_status
    # log_dotfiles_state()      -- save current environment to logfiles
    # dotfiles_initialize()     -- virtualenvwrapper initialize
    # dotfiles_postmkvirtualenv -- virtualenvwrapper postmkvirtualenv
    # dotfiles_preactivate()    -- virtualenvwrapper preactivate
    # dotfiles_postactivate()   -- virtualenvwrapper postactivate
    # dotfiles_predeactivate()  -- virtualenvwrapper predeactivate
    # dotfiles_postdeactivate() -- virtualenvwrapper postdeactivate


#### etc/bash/06-bashrc.completion.sh
### bashrc.completion.sh
    # _configure_bash_completion()  -- configure bash completion
    #                               note: `complete -p` lists completions


#### etc/bash/07-bashrc.python.sh
### bashrc.python.sh
    # pypath()              -- print python sys.path and site config
    # _setup_python()       -- configure $PYTHONSTARTUP
    # _setup_pip()          -- set $PIP_REQUIRE_VIRTUALENV=false
## Pyenv
    # _setup_pyvenv()       -- set $PYENV_ROOT, add_to_path, and pyenv venvw
## Conda / Anaconda
    # _setup_anaconda()     -- set $ANACONDA_ROOT, add_to_path
    # workon_conda()        -- workon a conda + venv project
    # wec()                 -- workon a conda + venv project
    #                       note: tab-completion only shows regular virtualenvs
    # mkvirtualenv_conda()  -- mkvirtualenv and conda create
    # rmvirtualenv_conda()  -- rmvirtualenv conda
    #   TODO
    # mkvirtualenv_conda_if_available() -- mkvirtualenv_conda OR mkvirtualenv
    # workon_conda_if_available()       -- workon_conda OR we OR workon


#### etc/bash/07-bashrc.virtualenvwrapper.sh
### bashrc.virtualenvwrapper.sh
# sudo apt-get install virtualenvwrapper || sudo pip install virtualenvwrapper
    # _setup_virtualenvwrapper()    -- configure $VIRTUALENVWRAPPER_*
    # lsvirtualenvs()       -- list virtualenvs in $WORKON_HOME
    # lsve()                -- list virtualenvs in $WORKON_HOME
    # backup_virtualenv()   -- backup VIRTUAL_ENV_NAME $1 to [$2]
    # backup_virtualenvs()  -- backup all virtualenvs in $WORKON_HOME to [$1]
    # rebuild_virtualenv()      -- rebuild a virtualenv, leaving pkgs in place
    # rebuild_virtualenvs()     -- rebuild all virtualenvs in $WORKON_HOME


#### etc/bash/08-bashrc.gcloud.sh
### bashrc.gcloud.sh
    # _setup_google_cloud() -- configure gcloud $PATH and bash completions


#### etc/bash/10-bashrc.venv.sh
### bashrc.venv.sh
#   note: most of these aliases and functions are overwritten by `we`
## Variables
    # __PROJECTSRC -- path to local project settings script
```

```
    # __SRC        -- path/symlink to local repository ($__SRC/hg $__SRC/git)
    # PATH="~/.local/bin:$PATH" (if not already there)
    # _VENV       -- path to local venv config script (executable)
## Functions
    # venv $@   -- call $_VENV $@
    # venv -h   -- print venv --help
    # venv -b   -- print bash configuration
    # venv -p   -- print IPython configuration as JSON
    # _venv <args> -- call $_VENV -E $@ (for the current environment)
    # we()         -- workon a virtualenv and load venv (TAB-completion)
    #  param $1: $VIRTUAL_ENV_NAME ("dotfiles")
    #  param $2: $_APP ("dotfiles") [default: $1)
    #   ${WORKON_HOME}/${VIRTUAL_ENV_NAME}  # == $VIRTUAL_ENV
    #   ${VIRTUAL_ENV}/src                  # == $_SRC
    #   ${_SRC}/${VIRTUAL_ENV_NAME}         # == $_WRD
    #  examples:
    #   we dotfiles
    #   we dotfiles dotfiles
## cd functions
    # cdb()     -- cd $_BIN
    # cde()     -- cd $_ETC
    # cdh()     -- cd $HOME
    # cdl()     -- cd $_LIB
    # cdlog()   -- cd $_LOG
    # cdp()     -- cd $PROJECT_HOME
    # cdph()    -- cd $PROJECT_HOME
    # cdpylib() -- cd $_PYLIB
    # cdpysite()-- cd $_PYSITE
    # cds()     -- cd $_SRC
    # cdv()     -- cd $VIRTUAL_ENV
    # cdve()    -- cd $WORKON_HOME
    # cdvar()   -- cd $_VAR
    # cdw()     -- cd $_WRD
    # cdwrd     -- cd $_WRD
    # cdwh      -- cd $WORKON_HOME
    # cdwrk()   -- cd $WORKON_HOME
    # cdww()    -- cd $_WWW
    # cdwww()   -- cd $_WWW
## Grin search
# virtualenv / virtualenvwrapper
    # grinv()   -- grin $VIRTUAL_ENV
    # grindv()  -- grind $VIRTUAL_ENV
# venv
    # grins()   -- grin $_SRC
    # grinds()  -- grind $_SRC
    # grinw()   -- grin $_WRD
    # grin-()   -- grin _WRD
    # grindw()  -- grind $_WRD
    # grind-()  -- grind $_WRD
    # grindctags()      -- generate ctags from grind (in ./tags)
    # grindctagssys()   -- generate ctags from grind --sys-path ($_WRD/tags)
    # grindctagsw()     -- generate ctags from (cd $_WRD; grind) ($_WRD/tags)
    # grindctagss()     -- generate ctags from (cd $_SRC; grind) ($_SRC/tags)
    # _load_venv_aliases()  -- load venv aliases
    #   note: these are overwritten by `we` [`source <(venv -b)`]
    # ssv()     -- supervisord   -c ${_SVCFG}
    # sv()      -- supervisorctl -c ${_SVCFG}
    # svd()     -- supervisorctl -c ${_SVCFG} restart && sv tail -f dev
```

```
    # svt()     -- supervisorctl -c "${_SVCFG}" tail -f
    # hgw()     -- hg -R  ${_WRD}
    # hg-()     -- hg -R  ${_WRD}
    # gitw()    -- git -C ${_WRD}
    # git-()    -- git -C ${_WRD}
    # serve-()  -- ${_SERVE_}
    # shell-()  -- ${_SHELL_}
    # test-()   -- cd ${_WRD} && python setup.py test
    # testr-()  -- reset; cd ${_WRD} && python setup.py test
    # makew()   -- cd $_WRD && make $@
    # make-()   -- cd $_WRD && make $@
    # mw()      -- cd $_WRD && make $@
    # _venv_set_prompt()    -- set PS1 with $WINDOW_TITLE, $VIRTUAL_ENV_NAME,
    #                          and ${debian_chroot}
    # _venv_ensure_paths()  -- create FSH paths in ${1} or ${VIRTUAL_ENV}


#### etc/bash/11-bashrc.venv.pyramid.sh
### bashrc.venv.pyramid.sh
    # workon_pyramid_app()  -- $VIRTUAL_ENV_NAME [$_APP] [open_terminals]


#### etc/bash/20-bashrc.editor.sh
### bashrc.editor.sh
    # setup_editor()    -- configure ${EDITOR}
    #   VIMBIN  (str):   /usr/bin/vim
    #   GVIMBIN (str):   /usr/bin/gvim
    #   MVIMBIN (str):   /usr/local/bin/mvim
    #   GUIVIMBIN (str): $GVIMBIN || $MVIMBIN || ""
    #   EDITOR  (str):   $VIMBIN -f || $GUIVIMBIN -f
    #   EDITOR_ (str):   $EDITOR || $GUIVIMBIN $VIMCONF --remote-tab-silent
    #   VIMCONF (str):   --servername ${VIRTUAL_ENV_NAME:-'EDITOR'}
    #   SUDO_EDITOR (str): $EDITOR
    # ggvim()   -- ${EDITOR} $@ 2>&1 >/dev/null
    # edits()   -- open $@ in ${GUIVIMBIN} --servername $1
    # e()       -- ${EDITOR_} $@      [ --servername $VIRTUAL_ENV_NAME ]
    # edit()    -- ${EDITOR_} $@      [ --servername $VIRTUAL_ENV_NAME ]
    # editcfg() -- ${EDITOR_} ${_CFG} [ --servername $VIRTUAL_ENV_NAME ]
    # sudoe()   -- EDITOR=${SUDO_EDITOR} sudo -e
    # sudoe()   -- EDITOR=${SUDO_EDITOR} sudo -e


#### etc/bash/29-bashrc.vimpagers.sh
### bashrc.vimpagers.sh
    # _configure_lesspipe() -- (less <file.zip> | lessv)
    # vimpager() -- call vimpager
    # lessv()   -- less with less.vim and vim (g:tinyvim=1)
    # lessg()   -- less with less.vim and gvim / mvim
    # lesse()   -- less with current venv's vim server
    # manv()    -- view manpages in vim
    # mang()    -- view manpages in gvim / mvim
    # mane()    -- open manpage with venv's vim server


#### etc/bash/30-bashrc.usrlog.sh
### bashrc.usrlog.sh
    # _USRLOG (str): path to .usrlog userspace shell command log
    # stid()      -- set $TERM_ID to a random string
```

```
    # stid $name  -- set $TERM_ID to string
    # note()      -- add a dated note to $_USRLOG [_usrlog_append]
    # usrlogv()   -- open usrlog with vim:   $VIMBIN + $_USRLOG
    # usrlogg()   -- open usrlog with gmvim: $GUIVIMBIN + $_USRLOG
    # usrloge()   -- open usrlog with editor:$EDITOR + $_USRLOG
    # ut()        -- tail $_USRLOG
    # ug()        -- egrep current usrlog: egrep $@ $_USRLOG
    # ugall()     -- egrep $@ $__USRLOG ${WORKON_HOME}/*/.usrlog
    # ugrin()     -- grin current usrlog: grin $@ $_USRLOG
    # ugrinall()  -- grin $@  $__USRLOG ${WORKON_HOME}/*/.usrlog
    # lsusrlogs() -- ls -tr   $__USRLOG ${WORKON_HOME}/*/.usrlog
    # _setup_usrlog()   -- source ${__DOTFILES}/etc/usrlog.sh
    # usrlogv() -- open $_USRLOG w/ $VIMBIN (and skip to end)
    # usrlogg() -- open $_USRLOG w/ $GUIVIMBIN (and skip to end)
    # usrloge() -- open $_USRLOG w/ $EDITOR_ [ --servername $VIRTUAL_ENV_NAME ]


#### etc/bash/30-bashrc.xlck.sh
### 30-bashrc.xlck.sh
## xlck     -- minimal X screensaver
    # xlck
    # xlck -I  --  (I)nstall xlck (apt-get)
    # xlck -U  --  check stat(U)s (show xautolock processes on this $DISPLAY)
    # xlck -S  --  (S)tart xlck (start xautolock on this $DISPLAY)
    # xlck -P  --  sto(P) xlck (stop xautolock on this $DISPLAY)
    # xlck -R  --  (R)estart xlck
    # xlck -M  --  suspend to ra(M) (and lock)
    # xlck -D  --  suspend to (D)isk (and lock)
    # xlck -L  --  (L)ock
    # xlck -X  --  shutdown -h now
    # xlck -h  --  help
    # xlck_status_all()             -- pgrep 'xautolock|xlock|i3lock', ps ufw
    # xlck_status_this_display()    -- show status for this $DISPLAY
    # _setup_xlck() -- source ${__DOTFILES}/etc/xlck.sh (if -z __IS_MAC)


#### etc/bash/40-bashrc.aliases.sh
### bashrc.aliases.sh
    #  _load_aliases()  -- load aliases
    # chmodr   -- 'chmod -R'
    # chownr   -- 'chown -R'
    # grep     -- 'grep --color=auto'
    # egrep    -- 'egrep --color=auto'
    # fgrep    -- 'fgrep --color=auto'
    # grindp   -- 'grind --sys.path'
    # grinp    -- 'grin --sys-path'
    # fumnt    -- 'fusermount -u'
    # ga       -- 'git add'
    # gl       -- 'git log --pretty=format:"%h : %an : %s" --topo-order --graph'
    # gs       -- 'git status'
    # gd       -- 'git diff'
    # gds      -- 'git diff -p --stat'
    # gc       -- 'git commit'
    # gco      -- 'git checkout'
    # gdc      -- 'git diff --cached'
    # gsl      -- 'git stash list'
    # gsn      -- 'git stash save'
    # gss      -- 'git stash save'
```

```
    # gitr      -- 'git remote -v'
    # hgl       -- 'hg glog --pager=yes'
    # hgs       -- 'hg status'
    # hgd       -- 'hg diff'
    # hgds      -- 'hg diff --stat'
    # hgdl      -- 'hg diff --color=always | less -R'
    # hgc       -- 'hg commit'
    # hgu       -- 'hg update'
    # hgq       -- 'hg qseries'
    # hgqd      -- 'hg qdiff'
    # hgqs      -- 'hg qseries'
    # hgqn      -- 'hg qnew'
    # hgr       -- 'hg paths'
    # __IS_MAC
        # la        -- 'ls -A -G'
        # ll        -- 'ls -alF -G'
        # ls        -- 'ls -G'
        # lt        -- 'ls -altr -G'
    # else
        # la        -- 'ls -A --color=auto'
        # ll        -- 'ls -alF --color=auto'
        # ls        -- 'ls --color=auto'
        # lt        -- 'ls -altr --color=auto'
    # __IS_LINUX
        # psx       -- 'ps uxaw'
        # psf       -- 'ps uxawf'
        # psxs      -- 'ps uxawf --sort=tty,ppid,pid'
        # psxh      -- 'ps uxawf --sort=tty,ppid,pid | head'
        # psh       -- 'ps uxaw | head'
        # psc       -- 'ps uxaw --sort=-pcpu'
        # psch      -- 'ps uxaw --sort=-pcpu | head'
        # psm       -- 'ps uxaw --sort=-pmem'
        # psmh      -- 'ps uxaw --sort=-pmem | head'
    # __IS_MAC
        # psx       -- 'ps uxaw'
        # psf       -- 'ps uxaw' # no -f
        # psh       -- 'ps uxaw | head'
        # psc       -- 'ps uxaw -c'
        # psch      -- 'ps uxaw -c | head'
        # psm       -- 'ps uxaw -m'
        # psmh      -- 'ps uxaw -m | head'
    # shtop     -- 'sudo htop'
    # t         -- 'tail'
    # tf        -- 'tail -f'
    # xclipc    -- 'xclip -selection c'


#### etc/bash/42-bashrc.commands.sh
### bashrc.commands.sh
# usage: bash -c 'source bashrc.commands.sh; funcname <args>'
    # chown-me()         -- chown -Rv user
    # chown-me-mine()    -- chown -Rv user:user && chmod -Rv go-rwx
    # chown-sme()        -- sudo chown -Rv user
    # chown-sme-mine()   -- sudo chown -Rv user:user && chmod -Rv go-rwx
    # chmod-unumask()    -- recursively add other+r (files) and other+rx (dirs)
    # new-sh()           -- create and open a new shell script at $1
    # diff-dirs()        -- list differences between directories
    # diff-stdin()       -- diff the output of two commands
```

```
    # wopen()           -- open path/URI/URL $1 in a new browser tab
    #                      see: scripts/x-www-browser
    # find-largefiles() -- find files larger than size (default: +10M)
    # find-pdf()        -- find pdfs and print info with pdfinfo
    # find-lately()     -- list and sort files in paths $@ by ISO8601 mtime
    #                      stderr > lately.$(date).errors
    #                      stdout > lately.$(date).files
    #                      stdout > lately.$(date).sorted
    #                      note:
    # find-setuid()     -- find all setuid and setgid files
    #                      stderr > find-setuid.errors
    #                      stdout > find-setuid.files
    # find-startup()    -- find common startup files in common locations
    # find-ssl()        -- find .pem and .db files and print their metadata
    # find-dpkgfile()   -- search dpkgs with apt-file
    # find-dpkgfiles()  -- sort dpkg /var/lib/dpkg/info/<name>.list
    # deb-chksums()     -- check dpkg md5 checksums with md5sums
    # deb-mkrepo()      -- create dpkg Packages.gz and Sources.gz from dir ${1}
    # mnt-chroot-bind() -- bind mount linux chroot directories
    # mnt-cifs()        -- mount a CIFS mount
    # mnt-davfs()       -- mount a WebDAV mount
    # lsof-sh()         -- something like lsof
    # lsof-net()        -- lsof the network things
    # net-stat()        -- print networking information
    # ssh-prx()         -- SSH SOCKS
    # strace-()         -- strace with helpful options
    # strace-f()        -- strace -e trace=file + helpful options
    # strace-f-noeno()  -- strace -e trace=file | grep -v ENOENT
    # hgst()            -- hg diff --stat, hg status, hg diff


#### etc/bash/50-bashrc.bashmarks.sh
### bashrc.bashmarks.sh
## bashmarks
    # l()  -- list bashmarks
    # s()  -- save bashmarks as $1
    # g()  -- goto bashmark $1
    # p()  -- print bashmark $1
    # d()  -- delete bashmark $1
    # lsbashmarks() -- list Bashmarks (e.g. for NERDTree)
    # see also: ${__DOTFILES}/scripts/nerdtree_to_bashmarks.py


#### etc/bash/70-bashrc.repos.sh
### 70-bashrc.repos.sh
#
#
#
# Use Cases
# * Original: a bunch of commands that i was running frequently
#   before readthedocs (and hostthedocs)
# * local mirrors (manual, daily?)
#   * no internet, outages
#   * push -f
#   * (~offline) Puppet/Salt source installs
#     * bandwidth: testing a recipe that pulls a whole repositor(ies)
# * what's changed in <project>'s source dependencies, since i looked last
#
```

```
# Justification
# * very real risks for all development projects
#   * we just assume that GitHub etc. are immutable and forever
#
# Features (TODO) [see: pyrpo]
# * Hg <subcommands>
# * Git <subcommands>
# * Bzr <subcommands>
# * periodic backups / mirroring
# * gitweb / hgweb
# * mirror_and_backup <URL>
# * all changes since <date> for <set_of_hg-git-bzr-svn_repositories>
# * ideally: transparent proxy
#   * +1: easiest
#   * -1: pushing upstream
#
# Caveats
# * pasting / referencing links which are local paths
# * synchronization lag
# * duplication: $__SRC/hg/<pkg> AND $VIRTUAL_ENV/src/<pkg>
#
    #  setup_dotfiles_docs_venv -- create default 'docs' venv
    #  setup_dotfiles_src_venv -- create default 'src' venv
    #
    #   __SRC_HG=${WORKON_HOME}/src/src/hg
    #   __SRC_GIT=${WORKON_HOME}/src/src/git
    #
    #  Hg runs hg commands as user hg
    #  Git runs git commands as user git
    #
    #  Hgclone will mirror to $__SRC_HG
    #  Gitclone will mirror to $__SRC_GIT
    #
    #
# __SRC_GIT_REMOTE_URI_PREFIX   -- default git remote uri prefix
# __SRC_GIT_REMOTE_NAME         -- name for git remote v
# __SRC_HG_REMOTE_URI_PREFIX    -- default hg remote uri prefix
# __SRC_HG_REMOTE_NAME          -- name for hg paths
    ## Create a new hosted repository with gitolite-admin
    #  $1   -- repo [user/]name (e.g. westurner/dotfiles)
    ## push a git repository to local git storage
    #  $1   -- repo [user/]name (e.g. westurner/dotfiles)
    #  $2   -- path of local repo (e.g. ~/wrk/.ve/dotfiles/src/dotfiles)
    ## Create a new hosted repository with mercurial-ssh
    ## push a hg repository to local git storage
    #  $1   -- repo [user/]name (e.g. westurner/dotfiles)
    #  $2   -- path of local repo (e.g. ~/wrk/.ve/dotfiles/src/dotfiles)
    #    fixperms ${path}
    #  host_docs    -- build and host documentation in a local directory
    #   param $1: <project_name>
    #   param $2: [<path>]
    #   param $3: [<docs/Makefile>]
    #   param $4: [<docs/conf.py>]
    # * log documentation builds
    # * build a sphinx documentation set with a Makefile and a conf.py
    # * rsync to docs webserver
    # * set permissions
    # this is not readthedocs.org
```

```
    # note: you must manually install packages into the
    # local 'docs' virtualenv'
                        # TODO: prompt?
        # >> 'html_theme = "_-_"
        # << 'html_theme = 'default'


#### etc/bash/99-bashrc.after.sh
```

2.4 Vim

https://github.com/westurner/dotfiles/blob/master/etc/.vimrc
https://github.com/westurner/dotvim/blob/master/vimrc
https://github.com/westurner/dotvim/blob/master/vimrc.full.bundles.vimrc

`make help_vim_rst:`

```
   # etc/vim/vimrc
" .vimrc
"
" Vim Reference
" ---------------
"  %          --  current filename
"  %:p        --  current filepath
"  $VIMRUNTIME      --  /{colors,syntax,macros}
"  ListMappings     --  list commented mappings
"  :map             --  list actual mappings
"  :scriptnames     --  list scripts and plugins
"  :set             --  list all nondefault options
"  e <path>         --  open file
"  e <pa...><tab>   --  open file with wildmenu completion
"  \e [...] <enter> --  open file
"  :tabnew <path>   --  open file
"  :read filename|  --  insert filename at cursor
"  :read !cmd       --  insert cmd output at cursor
"  :%! [cmd]        --  buffer > stdin > [cmd] > stdout => buffer.replace
"
"  [n]G             --  goto line #
"  g <C-g>          --  whereami
"  u                --  undo
"  ^r               --  redo
"  :%s:\(.*\):+\1:g --  Regex
"
" Modes
"  i                --  insert
"  I                --  insert at beginning of line
"  a                --  append
"  A                --  append at end of line
"  v                --  visual
"  c-v              --  visual block
"  ;;               --  command
"  <Esc>            --  command
"
" Vim Marks
```

```
"  m[a-z]{1}        --  set mark
"  `[a-z]{1}        --  goto mark
"  '[a-z]{1}        --  goto mark
"
" Macros
"  q[a-z]{1}        --  start recording
"  q                --  stop recording
"  @[a-z]{1}        --  replay macro
"  @@               --  repeat macro
"  q2<seq><esc>q;@2 --  record macro to 2 and repeat
"
" Searching
"  /<pattern>       --  search for term
"  *                --  search for term under cursor next
"  n                --  next search ocurrence
"  #                --  search for term under cursor previous
"  N                --  previous search ocurrence
"
"  :[l][vim]grep <pattern> <file>
"
"  :cl   :ll        --  list list
"  :copen :lopen    --  open list
"  :cw   :lw        --  toggle show list
"  :ccl[ose] :lcl   --  close list
"  :cn   :ln        --  next <Enter>
"  :cp   :lp        --  prev <Enter>
"  :cc!  :lc [nr]   --  jump to [nr]
"  :cfir :cla       --  first, last
"
" Yanking and Pasting
"  y[a-z]           --  yank to buffer [a-z]
"  p[a-z]           --  paste from buffer [a-z]
"  ]p               --  paste to level
"
" Indenting/Shifting Blocks
"  [n]<             --  shift block left
"  [n]>             --  shift block right
"
"
" Folding
"  :help Fold       --  also usr_28
"  :set nofen       --  stop folding
"  zf               --  create fold
"  zo               --  fold open
"  zO               --  fold open recursive
"  zc               --  fold close
"  zC               --  fold close recursive
"  zx               --  undo manual fold actions
"  zX               --  undo manual fold actions and recompute
"  zR               --  open all folds
"
" Etiquette
"  <leader> i       --  toggle unprintables
"  <leader> sd      --  toggle highlight EOL whitespace
"  <leader> sc      --  clear highlighting
"
" set window title to vim title (display full path)
"  :ListMappings    --  list .vimrc mapping comments
```

```
"  <space> -- <leader>
"  ,    --  <leader> == <comma>
"  ;;   --  <esc> == double semicolon
"  jk   --  <esc>
"  98   --  <esc> == 98
"  :;   --  <esc> == colon semicolon
"  :;   --  <esc> == colon semicolon
" Quickfix
"  <leader> q              --  toggle quicklist
"  <leader> n              --  next quicklist item
"  <leader> l              --  toggle location list
" Workaround vim lp:#572863
" Code Folding
" UTF-8
" TODO XXX
" Code Indenting
" Searching
"  set colorcolumn=0    --  clear color column
" Turn Off Visual Bell
" WildMenu
" Spell Checking
"  <leader> sp           --  toggle spellcheck
"  shift-<enter>        --  insert new line w/o changing mode
    " no error bells
    " Jump to last position
    " remove trailing whitespace
    " filetype extensions
    "    if &previewwindow
    "        exec 'setlocal winheight='.&previewheight
    "    endif
    " Auto completion
    "  CTRL-<space>     --  autocomplete menu
    "  CTRL-<tab>       --  autocomplete menu
    " close vim if the only window left open is a NERDTree
    " Open NERDTree automatically if no files were specified
" Drag and Drop
"   :help drag-n-drop
"   shift-<drop>    --  cd to file's directory
"   ctrl-<drop>     --  split new window for file
"   <drop>          --  open file or paste path at cursor
" Fonts
"  :PatchFont      -- set the font
" GUI Menubar
"  :HideMenubar    -- hide GUI menubar
"  :ShowMenubar    -- show GUI menubar
"   :Set256         -- set 256 colors (for console vim)
"   :Set88          -- set 88 colors (for console vim)
" GUI
    "  Remove gui scrollbars
    "  ctrl-z   --  undo
    "  alt-z    --  undo
    "  ctrl-r   --  redo
    "  alt-r    --  redo
    "  ctrl-X   --  cut
    "  alt-x    --  cut
    "  ctrl-c   --  copy
    "  alt-c    --  copy
        " always call Set256.
```

```
        " if this causes problems with older terminals
        " :Set88
" autocmd! Syntax * syn match ExtraWhitespace /\s\+$\| \+\ze\t/
"  <leader> sd              --  match EOL whitespace
"  <leader> sc              --  clear search highlighting
"  ctrl-q       --  close
"  <leader> i   --  toggle show invisibles
"  ,cd          --  :cd %:p:h
"  T            --  wrap paragram
" Keep search matches in the middle of the window.
"  <leader> [    --  toggle cursorline and cursorcolumn
"  <leader> hm   --  set horizontal line mark
"  <leader> hv   --  set vertical column mark
"  <leader> c    --  clear virt marks
"  Tab          --  Indent Line
" map <Tab>         >gb
"  Shift-Tab    --  Dedent Line
"  ctrl-t       --  Indent Current Line
"  ctrl-d       --  Dedent Current Line
"  >            --  Visual Indent Block
"  <            --  visual dedent block
" Alternative using Tab/Shift-Tab (for gvim).
"  tab          --  shift right
"  Shift-tab    --  shift left
" vnoremap <Tab>    >gv
"  ctrl-f       --  find
"  ctrl-alt-A   --  copy all
" Paste
"  shift-insert --  paste
"  ctrl-S       --  Save
"  ctrl-Alt-W   --  Close
"  ctrl-Home    --  Goto line one
"  ctrl-End     --  Goto line :-1
" PgUp/PgDn
"  K    --  PageUp
"  J    --  PageDown
" Pgup/Down are actually 2*<c-U>
" Buffer Nav
"  ctrl-a       --  move to beginning of line (^)
"  ctrl-e       --  move to end of line ($)
" Window Nav        (window-move-cursor)
"  ctrl-j       --  cursor window down
"  ctrl-u       --  cursor window down
"  ctrl-k       --  cursor window up
"  ctrl-i       --  cursor window up
"  ctrl-l       --  cursor window right
"  ctrl-h       --  cursor window left
" Window Resize     (window-resize)
"  ctrl-w _     --  maximize window height
"  ctrw-w 1_    --  minimize window height
"  ctrl-w |     --  maximize window width
"  ctrl-w 1|    --  minimize window width
"  ctrl-w =     --  equalize window sizes
"  [n]ctrl-w >  --  expand width
"  [n]ctrl-w <  --  contract width
"  [n]ctrl-w +  --  increase height
"  [n]ctrl-w -  --  reduce height
"  ctrl-w o     --  minimze all other windows
```

```
" Window Movement (window-move)
" Window Up
"  <leader> wk  --  move window up
"  ctrl-wi      --  move window up
"  <leader> wi  --  move window up
" Window Right
"  <leader> wl  --  move window right
" Window Down
"  <leader> wj  --  move window down
"  ctrl-wu      --  move window down
"  <leader> wu  --  move window down
" Window Left
"  <leader> wj  --  move window left
" Window Rotate
"  ctrl-w R     --  rotate window up
"  ctrl-w r     --  rotate window down
" Tab Movement (tab-page-commands)
"  ctrl-Alt-h   --  previous tab
"  Alt-u        --  previous tab
"  ctrl-Alt-l   --  next tab
"  Alt-i        --  next tab
" Man.vim          --  view manpages in vim
"  :Man man        --  view manpage for 'man'
"  <leader> o      --  Open uri under cursor
    "  :Ack <term> <path>
    "  <leader>a        --  Ack
" Grin              -- Find in Python
" Ctags
"  ctrl-[           --  go to tag under cursor
"  ctrl-T           --  go back  #TODO
" sh: ctags -R -f ~/.vim/tags/python-$PYVER.ctags $PYLIBDIR
" Use :make to see syntax errors. (:cn and :cp to move around, :dist to see
" all errors)
" Colors
"  :PatchColors     --  load local colorizing postsets
" List highlight colors
" Python
"  Wrap at 72 chars for comments.
"  read virtualenv's site-packages to vim path
" TODO: python regex current buffer
" :help if_pyth
" Tabsetting functions
"  :Fourtabs    -- set to four (4) soft tabs (Default)
" Default to fourtabs
"  :Threetabs   -- set to three (3) soft tabs (e.g. RST)
"  :Twotabs     -- set to two (2) soft tabs
"  :CurrentBuffer -- display number of current buffer
"  <leader> 2   -- diffget from bufnr 2
"  <leader> 3   -- diffget from bufnr 3
"  <leader> 4   -- diffget from bufnr 4
"  <leader> 2   -- diffget from bufnr 2
"  <leader> 3   -- diffget from bufnr 3
"  <leader> 4   -- diffget from bufnr 4
"  :Striptrailingwhitespace -- strip spaces at the end of lines
" Adjust font-size
"  <C-Up>   -- increase font size
"  <C-Down> -- decrease font size
"  <F3>     -- insert ReST date heading
```

```
" Trac


   # etc/vim/vimrc.full.bundles.vimrc
" Bundle            -- Vim bundle manager [help bundle]
" :BundleList          - list configured plugins
" :BundleInstall(!)    - install (update) plugins
" :BundleSearch(!) foo - search (or refresh cache first) for foo
" :BundleClean(!)      - confirm (or auto-approve) removal of unused plugins
"
" The Bundle URLs are intentionally complete https URLs
" * grep '^Bundle \'' vimrc.bundles
" * sed -i 's\https://github.com/\ssh://git@github.com/\g'
" Info.vim          -- vim infopages in vim [help info]
"  :Info sed        --  view infopage for 'sed'
"  <Space>          --  Scroll forward (page down).
"  <Backspace>      --  Scroll backward (page up).
"  <Tab>            --  Move cursor to next hyperlink within this node.
"  <Enter>,<C-]>    --  Follow hyperlink under cursor.
"  ;,<C-T>          --  Return to last seen node.
"  .,>              --  Move to the "next" node of this node.
"  p,<              --  Move to the "previous" node of this node.
"  u                --  Move "up" from this node.
"  d                --  Move to "directory" node.
"  t                --  Move to the Top node.
"  <C-S>            --  Search forward within current node only.
"  s                --  Search forward through all nodes for a specified
"  string.
"  q                --  Quit browser.
"
" Signify   -- show git/hg file changes in gutter [help signify]
" Fugitive      -- Git commands and statusline display [help fugitive]
" Lawrencium    -- Hg commands [help lawrencium]
" NERDTree      -- File browser [help NERDTree]
"  <Leader>e         --  toggle NERDTree
"  ctrl-e            --  toggle NERDTree
"  <Leader>E         --  open nerdtree to current file (:NERDTreeFind %:p:h)
"  ctrl-E            --  open nerdtree to current file (:NERDTreeFind %:p:h)
"  I                 --  toggle view hidden files
"  B                 --  toggle view bookmarks
"  cd                --  set vim CWD to selected dir
"  C                 --  refocus view to selected dir
"  o                 --  open
"  r                 --  refresh dir
"  R                 --  refresh root
"  t                 --  open in new tab
"  T                 --  open in new tab silently
"  u                 --  up a dir
"  U                 --  up a dir and leave open
"  x                 --  close node
"  X                 --  close all nodes recursive
"  ?                 --  toggle help
" FindInNERDTree    -- NERDTRee show current file [help NERDTreeFind]
"  <c-b>             --  toggle BufExplorer
"  ?                 --  toggle BufExplorer help
"  <leader>b         --  toggle BufExplorer
" CtrlP             -- file/buffer/mru finder [help ctrlp]
"  <C-p>             -- CtrlP (fuzzy matching)
```

```
" Syntastic         -- syntax highlighting [help syntastic]
" NERDCommenter     -- commenting [help NERDCommenter]
"  ,cm              --  minimal comment
"  ,cs              --  sexy comment
"  ,c<space>        --  toggle comment
" UltiSnips         --  syntax-specific snippets [help ultisnips]
"  snippetname<C-CR>    --  insert snippet
"  <tab>            --  next placeholder
"  <tab>            --  prev placeholder
"  ~/.vim/snippets-ulti/python:
"   climain         --  new cli script
"   setuppy         --  new setup.py script
" NeoComplCache -- code completion [help neocomplcache]
" unstack.vim   -- parse and open stacktrace paths [help unstack]
"  <leader> s   -- parse part/all of a stacktrace
" accordion.vim -- work w/ a number of vsplits at once [help accordion]
" ViM Airline   -- helpful statusbar information w/ vimscript [help airline]
    " base16, wombat, luna
    " base16, wombat, luna
" EasyMotion    -- easy visual motions [help easymotion]
"  <Leader>m-w/e    --  search forward (beg/end of word)
"  <Leader>m-b      --  search backward
"  <Leader>m-j      --  search line down
"  <Leader>m-k      --  search line up
" Jellybeans    -- a good colorscheme w/ sensible diff highlighting
"  :colorscheme jellybeans -- switch to the jellybeans colorscheme
" Vim-misc      -- functions for colorscheme-switcher and vim-session
" Vim Colorscheme Switcher [help colorscheme-switcher]
"  <F8>         -- cycle colors forward
"  <Shift><F8>  -- cycle colors reverse
" HiColors
"  call HiTest() -- print highlighting colors
" Pasting       -- make paste work normally [help paste]
" Vim Room      -- focus just the relevant text [help vimroom]
" VOoM Outline Viewer   -- view outlines of code and text [help voom]
"  VOoM modes:  html, markdown, python, rest,
"               thevimoutliner, txt2tags,
"               viki, vimwiki, wiki
"  :Voom [<format>] -- open Voom outline tab
"  :Voom rest       -- open ReStructuredText outline
"  ggg?G
" TagBar        -- source tag browser [help tagbar]
"  <leader> t   -- toggle TagBar"
" Vim Session   -- save and restore sessions between exits [help session]
"  :SaveSession <name>  -- save a session
"  :OpenSession <name>  -- open a saved session
"  :Restart             -- SaveSession restart && exit
"  :OpenSession restart -- open the 'restart' saved session
" Vim Unimpaired        --  moving between buffers [help unimpaired]
"  [a      :previous
"  ]a      :next
"  [A      :first
"  ]A      :last
"  [b      :bprevious
"  ]b      :bnext
"  [B      :bfirst
"  ]B      :blast
"  [l      :lprevious
```

```
"  ]l      :lnext
"  [L      :lfirst
"  ]L      :llast
"  [<C-L>  :lpfile
"  ]<C-L>  :lnfile
"  [q      :cprevious
"  ]q      :cnext
"  [Q      :cfirst
"  ]Q      :clast
"  [<C-Q>  :cpfile (Note that <C-Q> only works in a terminal if you disable
"  ]<C-Q>  :cnfile flow control: stty -ixon)
"  [t      :tprevious
"  ]t      :tnext
"  [T      :tfirst
"  ]T      :tlast
" Ack.vim       -- ack through files (instead of grep) [help ack]
" :Ack [options] PATTERN [directory]    -- search for pattern
" :AckAdd [options] PATTERN [directory] -- add a search pattern
" :AckWindow [options] PATTERN          -- search all visible buffers"
" vim-surround  -- paired tag wrappings [help surround]
"  ds   -- delete surroundings
"  cs   -- change surroundings
"
" csapprox      -- adapt gvim colorschemes for terminal vim [help csapprox]
" UndoTree      -- visualize vim undotree
"  <F5>     -- Toggle UndoTree (? for help)
" vim-nginx -- nginx ftdetect, indent, and syntax
" n3.vim    -- N3/Turtle RDF Syntax
" SPARQL    -- SPARQL syntax
" Python-mode       -- Python [help pymode]
"  :help pymode
"  [[    --  Jump to previous class or function
"  ]]    --  Jump to next class or function
"  [M    --  Jump to previous class or method
"  ]M    --  Jump to next class or method
"  aC    --  Select a class. Ex: vaC, daC, yaC, caC
"  iC    --  Select inner class. Ex: viC, diC, yiC, ciC
"  aM    --  Select a function or method. Ex: vaM, daM, yaM, caM
"  iM    --  Select inner function or method. Ex: viM, diM, yiM, ciM
"  g:pymode_python = { 'python', 'python3', 'disable' }
"  :PymodeLintToggle    -- toggle lint checking
"  :PymodeLintAuto      -- autofix current buffer pep8 errors
" - auto-show an error window
" - show lint signs
" - run lint on write
"  let g:pymode_lint_ignore = ""
"  let g:pymode_lint_select = ""
"  Pymode lint line annotation symbols
"   XX = TODO
"   CC = COMMENT
"   RR = VISUAL
"   EE = ERROR
"   II = INFO
"   FF = PYFLAKES
"  <F7>     -- set debugger breakpoints
"  auto lookup breakpoint cmd (pdb, ipdb, pudb)"
"  Searches upward for a .ropeproject file (that should be .vcs-ignored)
"  :PymodeRopeNewProject    -- Create a new .ropeproject in CWD
```

```
"  :PymodeRopeRegenerate    -- Regenerate rope project cache
"  <C-c>d       -- show docs for current function w/ pymode
"  rope for autocompletion
"  <C-Space>    -- rope autocomplete
"  <leader> j       --  :RopeGotoDefinition
"  <C-c> ro     -- organize Python imports; drop unused (:PymodeRopeAutoImport)
"  :PymodeRopeUndo  -- Undo last project changes
"  :PymodeRopeRedo  -- Redo last project changes
"  <C-c> rr     -- rope rename
" vim-virtualenv    -- Python virtualenv [help virtualenv]
"  :help
"  :VirtualEnvDeactivate
"  :VirtualEnvList
"  :VirtualEnvActivate <name>
"  :VirtualEnvActivate <TAB>
" Sort python imports
"  :PyFixImports    --  sort import statements
" Pytest.vim    -- py.test red/green results [help pytest]
"  :Pytest clear    -- reset pytest globals
"  :Pytest file     --  pytest file
"  :Pytest class    --  pytest class
"  :Pytest method   --  pytest method
"  :Pytest {...} --pdb  -- pytest file/class/method with pdb
"  <leader>tf       --  pytest file
"  <leader>tc       --  pytest class
"  <leader>tm       --  pytest method
"  " cycle through test errors
"  <leader>tn       --  pytest next error
"  <leader>tp       --  pytest prev error
"  <leader>te       --  pytest error
" Pyrex         -- Pyrex syntax
" Jinja         -- Jinja Templates syntax
" clickbable.vim -- click-able links
" Riv.vim   -- ReStructuredText [help riv]
"  :RivIntro
"  :RivQuickStart
"  :RivPrimer
"  :RivSpecification
"  :RivCheatSheet
" Salt      -- Salt syntax
" Trac      -- Trac [help trac]
" webapi-vim -- vim web API [help webapi[-{html, http, json, xml}]]
" gist-vim  -- Create a gist.github.com [help gist-vim]
" github-issues.vim     -- autocomplete, CRUD GitHub issues [help Gissues]
" All of your Bundles must be added before the following line
"


   # etc/vim/vimrc.tinyvim.bundles.vimrc
" Bundle            -- Vim bundle manager [help bundle]
" :BundleList          - list configured plugins
" :BundleInstall(!)    - install (update) plugins
" :BundleSearch(!) foo - search (or refresh cache first) for foo
" :BundleClean(!)      - confirm (or auto-approve) removal of unused plugins
"
" The Bundle URLs are intentionally complete https URLs
" * grep '^Bundle \'' vimrc.bundles
" * sed -i 's\https://github.com/\ssh://git@github.com/\g'
```

```
" Info.vim          -- vim infopages in vim [help info]
"  :Info sed        --  view infopage for 'sed'
"  <Space>          --  Scroll forward (page down).
"  <Backspace>      --  Scroll backward (page up).
"  <Tab>            --  Move cursor to next hyperlink within this node.
"  <Enter>,<C-]>    --  Follow hyperlink under cursor.
"  ;,<C-T>          --  Return to last seen node.
"  .,>              --  Move to the "next" node of this node.
"  p,<              --  Move to the "previous" node of this node.
"  u                --  Move "up" from this node.
"  d                --  Move to "directory" node.
"  t                --  Move to the Top node.
"  <C-S>            --  Search forward within current node only.
"  s                --  Search forward through all nodes for a specified
"  string.
"  q                --  Quit browser.
"
" Signify   -- show git/hg file changes in gutter [help signify]
" NERDTree      -- File browser [help NERDTree]
"  <Leader>e         --  toggle NERDTree
"  ctrl-e            --  toggle NERDTree
"  <Leader>E         --  open nerdtree to current file (:NERDTreeFind %:p:h)
"  ctrl-E            --  open nerdtree to current file (:NERDTreeFind %:p:h)
"  I                 --  toggle view hidden files
"  B                 --  toggle view bookmarks
"  cd                --  set vim CWD to selected dir
"  C                 --  refocus view to selected dir
"  o                 --  open
"  r                 --  refresh dir
"  R                 --  refresh root
"  t                 --  open in new tab
"  T                 --  open in new tab silently
"  u                 --  up a dir
"  U                 --  up a dir and leave open
"  x                 --  close node
"  X                 --  close all nodes recursive
"  ?                 --  toggle help
" FindInNERDTree   -- NERDTRee show current file [help NERDTreeFind]
"  <c-b>            --  toggle BufExplorer
"  ?                --  toggle BufExplorer help
"  <leader>b        --  toggle BufExplorer
" CtrlP             -- file/buffer/mru finder [help ctrlp]
"  <C-p>            -- CtrlP (fuzzy matching)
" Syntastic         -- syntax highlighting [help syntastic]
" EasyMotion    -- easy visual motions [help easymotion]
"  <Leader>m-w/e    --  search forward (beg/end of word)
"  <Leader>m-b      --  search backward
"  <Leader>m-j      --  search line down
"  <Leader>m-k      --  search line up
" Jellybeans    -- a good colorscheme w/ sensible diff highlighting
"  :colorscheme jellybeans -- switch to the jellybeans colorscheme
" Vim-misc      -- functions for colorscheme-switcher and vim-session
" Vim Colorscheme Switcher [help colorscheme-switcher]
"  <F8>         -- cycle colors forward
"  <Shift><F8>  -- cycle colors reverse
" vim-nginx -- nginx ftdetect, indent, and syntax
" n3.vim    -- N3/Turtle RDF Syntax
" SPARQL    -- SPARQL syntax
```

```
" Pyrex         -- Pyrex syntax
" Jinja         -- Jinja Templates syntax
" Salt      -- Salt syntax
" All of your Bundles must be added before the following line
"
```

2.5 I3wm

https://github.com/westurner/dotfiles/blob/master/etc/.i3/config

make help_i3_rst:

```
#
#  #  Default location: ~/.i3/config
#  #  List commented command shortcuts with::
#
#  #     cat ~/.i3/config | egrep '(^(\s+)?###+ |^(\s+)?#  )'
#
# #!/bin/sh
# ### .i3/config requirements
#
# ## Ubuntu (12.04)
# # MUST
# apt-get install i3 i3status xautolock xlockmore i3lock
# hg clone https://github.com/westurner/dotfiles ~/.dotfiles  # etc/xlck.sh
#
# # SHOULD
# apt-get install gnome-terminal network-manager-gnome thunar pulseaudio-utils
# apt-get install feh                  # wallpaper
# apt-get install xfce4-screenshooter  # screenshots
# mkdir -p ~/pictures/screens          # screenshots
# apt-get install xbacklight           # brightness
#
# # COULD
# apt-get install vim-gnome            # scratchpad
# add-apt-repository ppa:kilian/f.lux  # f.lux
# apt-get update                       # f.lux
# apt-get install fluxgui              # http://justgetflux.com
#
# ## References
# * http://i3wm.org/docs/userguide.html
# * https://faq.i3wm.org/question/1425/variable-substitution/
# * i3-config-wizard
#
# ## Notes
# * grab keyboard mappings: xev | grep keycode
# Set i3 modifier keys to variables
#  # set $mod to <alt>    (<Alt_L> and <Alt_R>)
#  # set $mod2 to <Super> (<Super_L> and <Super_R>)
# font for window titles. ISO 10646 = Unicode
# Pango requires i3 version >= TODO
# reload the configuration file
#  <alt><shift> c   -- reload i3 configuration
# restart i3 inplace (preserves your layout/session, can be used to upgrade i3)
#  <alt><shift> r   -- restart i3 (session preserving)
```

```
# exit i3 (logs you out of your X session)
#  <super><shift> l  -- exit i3 (close all and logout of X session)
#  <alt><shift> q   -- close focused window
#  # Hide edge borders
#  # Get WM_CLASS with $(xprop WM_CLASS)
# set $set_xwallpaper feh --bg-fill --no-xinerama ~/wallpaper.png
#  - Set X background
#  - Set X wallaper to (~/wallpaper.png)
#  - Start screensaver
# - Launch network applet (optional)
# exec_always --no-startup-id $network_applet  # (optional)
# see also: nmcli
#  <super> l        -- lock screen
#  <XF86PowerOff>   -- exit
#  <XF86Sleep>      -- suspend
#  <XF86MonBrightnessUp>      -- brightness up
#  <XF86MonBrightnessDown>    -- brightness down
#  <XF86AudioRaiseVolume>   -- volume up
#  <XF86AudioLowerVolume>   -- volume down
#  <alt> x      -- run command
#  <super> r    -- run command
#  <super> e    -- launch browser
#  <alt><shift> g   -- launch editor
#  <alt><shift> b   -- launch browser
#  <alt><shift> t   -- launch terminal
#  <super> t        -- launch terminal
#  <alt> <enter>    -- launch terminal
#  XF86Calculator   -- launch calculator
#  <alt><shift> w          -- launch network manager applet (see also: $(nmcli))
#  <PrintScr>       -- screenshot (full screen)
#  <alt> <PrintScr> -- screenshot (current window)
#  <alt> v      -- focus nearest: editor
#  <alt> t      -- focus nearest: terminal
#  <alt> b      -- focus nearest: browser
#  <alt> [         -- start xflux
#  <alt> ]         -- stop xflux
#  <alt><shift> ]  -- reset gamma to 1.0
#  <alt><shift> [  -- xgamma -bgamma 0.6 -ggamma 0.9 -rgamma 0.9
#  <alt><shift> \  -- xgamma -bgamma -0.4 -ggamma 0.4 -rgamma 0.9
#  <alt> <space>            -- toggle focus mode: tiling / floating
#  <alt><shift> <space>     -- toggle tiling/floating mode for focused window
#  <alt> <backspace>        -- toggle tiling/floating mode for focused window
#  <alt> <mouse>            -- drag floating window to position
#  # Note: popups will be hidden below fullscreened windows
#  <alt><shift> f   -- fullscreen
#  # popup during fullscreen exits fullscreen
# Split next window
#  <alt><shift> h   -- split [next] window horizontally
#  <alt><shift> v   -- split [next] window vertically
#  <alt> w          -- tabbed window layout
#  <alt> e          -- Default window layout
#  <alt> s          -- stacked window layout
#  <alt> a          -- focus parent container
#  <alt><shift> a   -- focus child container
#  <alt> Up     -- focus up
#  <alt> Down   -- focus down
#  <alt> Left   -- focus left
#  <alt> Right  -- focus right
```

```
#  <alt> h      -- focus left
#  <alt> j      -- focus down
#  <alt> k      -- focus up
#  <alt> l      -- focus right
# Toggle between previous and current workspace
#  <alt> 0-9        -- switch to workspace N  (repeat to return)
#  <super> 0-9      -- switch to workspace N  (repeat to return)
#  <alt> <F_n>      -- switch to workspace N (repeat to return)
#  <alt> <Keypad_n> -- switch to workspace N (repeat to return)
#  <super> Left     -- move to previous workspace
#  <super> Right    -- move to next workspace
#  <super> Up       -- move to second most recently focused workspace
#  <super> Left     -- move container to previous workspace
#  <super> Right    -- move container to next workspace
#  <super> Up       -- move container to second most recently focused workspace
#  <alt><shift> Up      -- move window up
#  <alt><shift> Down    -- move window down
#  <alt><shift> Left    -- move window left
#  <alt><shift> Right   -- move window right
#  <alt><shift> h       -- move window left
#  <alt><shift> j       -- move window down
#  <alt><shift> k       -- move window up
#  <alt><shift> l       -- move window right
#  <alt><shift>  [N: 0-9]   -- move to workspace N
#  <alt><shift> [KP_N: 0-9] -- move to workspace N
#  <super> [KP_N: 0-9] -- move to workspace N
#  <super><shift> Left  -- move workspace to left
#  <super><shift> Right -- move workspace to right
#  <alt><shift> <minus>     -- make the currently focused window a scratchpad
#  <alt> <minus>            -- show/hide and cycle through scratchpad windows
#  <alt><shift> s           -- start scratchpad editor
#  <alt> <XF86Favorites>    -- start scratchpad editor
#  <XF86Favorites>          -- show the $scratchpad_editor_selector
#  <alt> <backspace>        -- toggle tiling/floating mode for focused window
# see above.
#  # on (re)load, move $scratchpad_editor_selector windows to scratchpad
#  <alt> r      -- enter resize mode
    # These bindings trigger as soon as you enter the resize mode
    #
    # They resize the border in the direction you pressed, e.g.
    # when pressing left, the window is resized so that it has
    # more space on its left
    # same bindings, but for the arrow keys
    #  Left         -- grow left
    #  <shift> Left     -- shrink left
    #  Down             -- grow down
    #  <shift> Down     -- shrink down
    #  Up               -- grow up
    #  <shift> Up       -- shrink up
    #  Right            -- grow right
    #  <shift> Right    -- shrink right
    #  h            -- grow left
    #  <shift> h    -- shrink left
    #  j            -- grow down
    #  <shift> j    -- shrink down
    #  k            -- grow up
    #  <shift> k    -- shrink up
    #  l            -- grow right
```

```
    #  <shift> l    -- shrink right
    # back to normal: Enter or Escape
    #  <enter>  -- exit resize mode
    #  <esc>    -- exit resize mode
# color defines for zenburn styled i3 derived from:
# https://faq.i3wm.org/question/2071/how-can-i-change-look-of-windows/?answer=2075
# set some nice colors      border     background  text
#  # display i3bar with i3status
    # $ xrandr-tool outputs
```

2.6 Scripts

https://github.com/westurner/dotfiles/tree/master/scripts

In `scripts/`

bashmarks_to_nerdtree.sh Convert *bashmarks* shortcut variables starting with `DIR_` to NERDTreeBookmarks format:

```
l
./bashmarks_to_nerdtree.sh | tee ~/.NERDTreeBookmarks
```

bootstrap_dotfiles.sh* Clone, update, and install dotfiles in `$HOME`

See: bootstrap_dotfiles.sh

compare_installed.py Compare packages listed in a debian/ubuntu APT `.manifest` with installed packages.

See: https://github.com/westurner/pkgsetcomp

gittagstohgtags.sh Convert `git` tags to `hgtags` format

pulse.sh Setup, configure, start, stop, and restart `pulseaudio`

setup_mathjax.py Setup `MathJax`

setup_pandas_notebook_deb.sh Setup `IPython Notebook`, `Scipy`, `Numpy`, `Pandas` with Ubuntu packages and pip

setup_pandas_notebook.sh Setup `Brew`, `IPython Notebook`, `scipy`, `numpy`, and pandas on OSX

setup_scipy_deb.py Install and symlink `scipy`, `numpy`, and `matplotlib` from `apt`

deb_deps.py Work with debian dependencies

deb_search.py Search for a debian package

build_docs.py Build sets of sphinx documentation projects

greppaths.py Grep

lsof.py lsof subprocess wrapper

mactool.py MAC address tool

optimizepath.py Work with PATH as an ordered set

passwordstrength.py Gauge password strength

pipls.py Walk and enumerate a pip requirements file

pycut.py Similar to `coreutils`' `cut`: split line-based files into fields. See: *pyline.py* (`pyline 'w[1:2]'`)

py_index.py Create a python package index HTML file for a directory of packages. (`.egg`, `.zip`, `.tar.gz`, `tgz`)

pyline.py Similar to `sed` and `awk`: Execute python expressions over line-based files.

See: https://github.com/westurner/pyline

pyren.py Skeleton regex file rename script

See: https://github.com/westurner/pyleset

pyrpo.py Wrap version control system commandline interfaces

See: https://github.com/westurner/pyrpo

usrlog.py Search through `.usrlog` files

x-www-browser Launch browser tabs for each argument (OSX, Linux, webbrowser)

CHAPTER 3

Venv

Venv makes working with *Virtualenvwrapper*, *Bash*, and *IPython* very easy.

There are three parts to "`venv`":

- 10-bashrc.venv.sh
- dotfiles.venv.ipython_config.py
- dotfiles.venv.ipython_magics.py

10-bashrc.venv.sh (*docs*) configures variables like `$VIRTUAL_ENV_NAME`, `$_SRC`, and `$_WRD`; and functions like `we()` and `e()` for *Bash* (and *ZSH*).

dotfiles.venv.ipython_config.py (`dotfiles.venv.ipython_config`) provides a shell command (`venv`) called by `we()` for generating shell configuration for a *Virtualenv* and configures *IPython*.

dotfiles.venv.ipython_magics.py (`dotfiles.venv.ipython_magics`) configures the same `cd` commands and `ds` command defined in `10-bashrc.venv.sh` and `ipython_config.py` for *IPython*.

3.1 Quickstart

```
# print shell configuration for a (hypothetical) dotfiles virtualenv
venv dotfiles --bash

# print shell configuration for the current ${VIRTUAL_ENV} [and ${_WRD}]
venv -E --bash

# run a command within a virtualenv
venv dotfiles -x bash

# workon a virtualenvwrapper virtualenv (we) (source <(venv -E --bash))
we dotfiles

# workon ${WORKON_HOME}/dotfiles/src/otherproject (echo $_APP $_WRD)
we dotfiles otherproject
```

3.2 Usage

3.2.1 Shell Command

```
$ python ../src/dotfiles/venv/ipython_config.py --help
Usage: ipython_config.py [-b|--print-bash] [-t] [-e] [-E<virtualenv>] [appname]

dotfiles.venv.ipython_config.py

Options:
  -h, --help            show this help message and exit
  -E, --from-shell-environ
  -p, --print, --print-json, --json
                        Print venv configuration as JSON
  -b, --bash, --print-bash, --zsh, --print-zsh
                        Print venv configuration for Bash, ZSH
  -x RUN_COMMAND, --cmd=RUN_COMMAND, --command=RUN_COMMAND
                        Run a command in a venv-configured shell
  -t, --terminals, --open-terminals
                        Open terminals within the venv [gnome-terminal]
  -e, --editors, --open-editors
                        Open an editor with venv._project_files
                        [$PROJECT_FILES]
  --platform=PLATFORM   Platform to generate configuration for
  -v, --verbose
  -q, --quiet
  -T, --test

Copyright 2014 Wes Turner. New BSD License
```

3.2.2 Python API

A `dotfiles.venv.ipython_config.Venv` object builds a `dotfiles.venv.ipython_config.Env` OrderedDict (`.env`) with `$VIRTUAL_ENV`-relative paths and environment variables in a common filesystem hierarchy and an OrderedDict of command aliases (`.aliases`), which can be serialized to a bash script (`venv --bash`), JSON (`venv --print`), and IPython configuration.

```
from dotfiles.venv.ipython_config import Venv
venv = Venv(from_environ=True)
venv.print()
venv.bash_env()

venv.configure_sys()
venv.configure_ipython()

assert venv.virtualenv  == venv.env['VIRTUAL_ENV']
assert venv.appname     == venv.env['_APP']

print(venv.env['_WRD'])     # working directory
print(venv.aliases['e'])    # edit with --servername $_APP
```

3.3 Example Venv Configuration

3.3.1 Shell Configuration

venv dotfiles --bash:

```
$ python ../src/dotfiles/venv/ipython_config.py dotfiles --bash \
| sed "s,${HOME},~,g"
export VIRTUAL_ENV='~/wrk/.ve/dotfiles'
export VIRTUAL_ENV_NAME='dotfiles'
export _BIN='~/wrk/.ve/dotfiles/bin'
export _ETC='~/wrk/.ve/dotfiles/etc'
export _ETCOPT='~/wrk/.ve/dotfiles/etc/opt'
export _HOME='~/wrk/.ve/dotfiles/home'
export _ROOT='~/wrk/.ve/dotfiles/root'
export _LIB='~/wrk/.ve/dotfiles/lib'
export _PYLIB='~/wrk/.ve/dotfiles/lib/python2.7'
export _PYSITE='~/wrk/.ve/dotfiles/lib/python2.7/site-packages'
export _MNT='~/wrk/.ve/dotfiles/mnt'
export _MEDIA='~/wrk/.ve/dotfiles/media'
export _OPT='~/wrk/.ve/dotfiles/opt'
export _SBIN='~/wrk/.ve/dotfiles/sbin'
export _SRC='~/wrk/.ve/dotfiles/src'
export _SRV='~/wrk/.ve/dotfiles/srv'
export _TMP='~/wrk/.ve/dotfiles/tmp'
export _USR='~/wrk/.ve/dotfiles/usr'
export _USRBIN='~/wrk/.ve/dotfiles/usr/bin'
export _USRINCLUDE='~/wrk/.ve/dotfiles/usr/include'
export _USRLIB='~/wrk/.ve/dotfiles/usr/lib'
export _USRLOCAL='~/wrk/.ve/dotfiles/usr/local'
export _USRSBIN='~/wrk/.ve/dotfiles/usr/sbin'
export _USRSHARE='~/wrk/.ve/dotfiles/usr/share'
export _USRSRC='~/wrk/.ve/dotfiles/usr/src'
export _VAR='~/wrk/.ve/dotfiles/var'
export _VARCACHE='~/wrk/.ve/dotfiles/var/cache'
export _VARLIB='~/wrk/.ve/dotfiles/var/lib'
export _VARLOCK='~/wrk/.ve/dotfiles/var/lock'
export _LOG='~/wrk/.ve/dotfiles/var/log'
export _VARMAIL='~/wrk/.ve/dotfiles/var/mail'
export _VAROPT='~/wrk/.ve/dotfiles/var/opt'
export _VARRUN='~/wrk/.ve/dotfiles/var/run'
export _VARSPOOL='~/wrk/.ve/dotfiles/var/spool'
export _VARTMP='~/wrk/.ve/dotfiles/var/tmp'
export _WWW='~/wrk/.ve/dotfiles/var/www'
export _USRLOG='~/wrk/.ve/dotfiles/.usrlog'
export HISTFILE='~/wrk/.ve/dotfiles/.bash_history'
export HISTSIZE=1000000
export HISTFILESIZE=1000000
export PAGER='/usr/bin/less -R'
export _APP='dotfiles'
export _WRD='~/wrk/.ve/dotfiles/src/dotfiles'
export VIMBIN='/usr/bin/vim'
export GVIMBIN=None
export MVIMBIN=None
export GUIVIMBIN=None
export VIMCONF='--servername dotfiles'
export _EDIT_='/usr/bin/vim -f'
```

```
export EDITOR_='/usr/bin/vim -f'
export _IPSESSKEY='~/wrk/.ve/dotfiles/src/.sessionkey'
export _NOTEBOOKS='~/wrk/.ve/dotfiles/src/notebooks'
export _IPQTLOG='~/wrk/.ve/dotfiles/.ipqt.log'
export _WRD_SETUPY='~/wrk/.ve/dotfiles/src/dotfiles/setup.py'
export _TEST_='(cd "${_WRD}" && python "${_WRD_SETUPY}" test)'
export _CFG='~/wrk/.ve/dotfiles/etc/development.ini'
export _EDITCFG_='/usr/bin/vim -f ~/wrk/.ve/dotfiles/etc/development.ini'
export _SHELL_='(cd "${_WRD}" && "${_BIN}"/pshell "${_CFG}")'
export _SERVE_='(cd "${_WRD}" && "${_BIN}"/pserve --app-name=main --reload --monitor-restart "${_CFG
export PROJECT_FILES='README.rst CHANGES.rst TODO.rst Makefile setup.py requirements.txt .hgignore .g
export _SVCFG='~/wrk/.ve/dotfiles/etc/supervisord.conf'
export _SVCFG_=' -c "~/wrk/.ve/dotfiles/etc/supervisord.conf"'
eval 'cdb () {
    cd "${_BIN}"/$@
}';
eval 'cde () {
    cd "${_ETC}"/$@
}';
eval 'cdh () {
    cd "${HOME}"/$@
}';
eval 'cdl () {
    cd "${_LIB}"/$@
}';
eval 'cdlog () {
    cd "${_LOG}"/$@
}';
eval 'cdp () {
    cd "${PROJECT_HOME}"/$@
}';
eval 'cdph () {
    cd "${PROJECT_HOME}"/$@
}';
eval 'cdpylib () {
    cd "${_PYLIB}"/$@
}';
eval 'cdpysite () {
    cd "${_PYSITE}"/$@
}';
eval 'cds () {
    cd "${_SRC}"/$@
}';
eval 'cdv () {
    cd "${VIRTUAL_ENV}"/$@
}';
eval 'cdve () {
    cd "${VIRTUAL_ENV}"/$@
}';
eval 'cdvar () {
    cd "${_VAR}"/$@
}';
eval 'cdwh () {
    cd "${WORKON_HOME}"/$@
}';
eval 'cdwrk () {
    cd "${WORKON_HOME}"/$@
}';
```

```
eval 'cdw () {
    cd "${_WRD}"/$@
}';
eval 'cd- () {
    cd "${_WRD}"/$@
}';
eval 'cdww () {
    cd "${_WWW}"/$@
}';
eval 'cdwww () {
    cd "${_WWW}"/$@
}';
eval 'cdhelp () {
    set | grep "^cd.*()" | cut -f1 -d" " #$@
}';
eval 'edit- () {
    ${_EDIT_} $@
}';
alias gvim-='${VIMBIN} -f'
eval 'ipskey () {
    (python -c "import os; print os.urandom(128).encode(\"base64\")" > "${_IPSESSKEY}" ) && chmod 060
}';
eval 'ipnb () {
    ipython notebook --secure --Session.keyfile="${_IPSESSKEY}" --notebook-dir="${_NOTEBOOKS}" --deep
}';
eval 'ipqt () {
    ipython qtconsole --secure --Session.keyfile="${_IPSESSKEY}" --logappend="${_IPQTLOG}" --deep-rel
}';
eval 'grinv () {
    grin --follow $@ "${VIRTUAL_ENV}"
}';
eval 'grindv () {
    grind --follow $@ --dirs "${VIRTUAL_ENV}"
}';
eval 'grins () {
    grin --follow $@ "${_SRC}"
}';
eval 'grinds () {
    grind --follow $@ --dirs "${_SRC}"
}';
alias test-='(cd "${_WRD}" && python "${_WRD_SETUPY}" test)'
alias testr-='reset && (cd "${_WRD}" && python "${_WRD_SETUPY}" test)'
alias nose-='(cd "${_WRD}" && nosetests)'
eval 'grinw () {
    grin --follow $@ "${_WRD}"
}';
eval 'grin- () {
    grin --follow $@ "${_WRD}"
}';
eval 'grindw () {
    grind --follow $@ --dirs "${_WRD}"
}';
eval 'grind- () {
    grind --follow $@ --dirs "${_WRD}"
}';
alias hgv-='hg view -R "${_WRD}"'
alias hgl-='hg -R "${_WRD}" log'
eval 'editcfg () {
```

```
    "${_EDITCFG_}" $@
}';
alias serve-='(cd "${_WRD}" && "${_BIN}"/pserve --app-name=main --reload --monitor-restart "${_CFG}")
alias shell-='(cd "${_WRD}" && "${_BIN}"/pshell "${_CFG}")'
eval 'e () {
    ${_EDIT_} $@
}';
eval 'editp () {
    $GUIVIMBIN $VIMCONF $PROJECT_FILES $@
}';
eval 'makewrd () {
    (cd "${_WRD}" && make $@)
}';
eval 'make- () {
    (cd "${_WRD}" && make $@)
}';
eval 'mw () {
    (cd "${_WRD}" && make $@)
}';
alias ssv='supervisord -c "${_SVCFG}"'
alias sv='supervisorctl -c "${_SVCFG}"'
alias svt='sv tail -f'
alias svd='supervisorctl -c "${_SVCFG}" restart dev && supervisorctl -c "${_SVCFG}" tail -f dev'
```

3.3.2 JSON Configuration

`venv dotfiles --print`:

```
$ python ../src/dotfiles/venv/ipython_config.py dotfiles --print \
| sed "s,${HOME},~,g"
{
  "env": {
    "VIRTUAL_ENV": "~/wrk/.ve/dotfiles",
    "VIRTUAL_ENV_NAME": "dotfiles",
    "_BIN": "~/wrk/.ve/dotfiles/bin",
    "_ETC": "~/wrk/.ve/dotfiles/etc",
    "_ETCOPT": "~/wrk/.ve/dotfiles/etc/opt",
    "_HOME": "~/wrk/.ve/dotfiles/home",
    "_ROOT": "~/wrk/.ve/dotfiles/root",
    "_LIB": "~/wrk/.ve/dotfiles/lib",
    "_PYLIB": "~/wrk/.ve/dotfiles/lib/python2.7",
    "_PYSITE": "~/wrk/.ve/dotfiles/lib/python2.7/site-packages",
    "_MNT": "~/wrk/.ve/dotfiles/mnt",
    "_MEDIA": "~/wrk/.ve/dotfiles/media",
    "_OPT": "~/wrk/.ve/dotfiles/opt",
    "_SBIN": "~/wrk/.ve/dotfiles/sbin",
    "_SRC": "~/wrk/.ve/dotfiles/src",
    "_SRV": "~/wrk/.ve/dotfiles/srv",
    "_TMP": "~/wrk/.ve/dotfiles/tmp",
    "_USR": "~/wrk/.ve/dotfiles/usr",
    "_USRBIN": "~/wrk/.ve/dotfiles/usr/bin",
    "_USRINCLUDE": "~/wrk/.ve/dotfiles/usr/include",
    "_USRLIB": "~/wrk/.ve/dotfiles/usr/lib",
    "_USRLOCAL": "~/wrk/.ve/dotfiles/usr/local",
    "_USRSBIN": "~/wrk/.ve/dotfiles/usr/sbin",
    "_USRSHARE": "~/wrk/.ve/dotfiles/usr/share",
    "_USRSRC": "~/wrk/.ve/dotfiles/usr/src",
```

```
    "_VAR": "~/wrk/.ve/dotfiles/var",
    "_VARCACHE": "~/wrk/.ve/dotfiles/var/cache",
    "_VARLIB": "~/wrk/.ve/dotfiles/var/lib",
    "_VARLOCK": "~/wrk/.ve/dotfiles/var/lock",
    "_LOG": "~/wrk/.ve/dotfiles/var/log",
    "_VARMAIL": "~/wrk/.ve/dotfiles/var/mail",
    "_VAROPT": "~/wrk/.ve/dotfiles/var/opt",
    "_VARRUN": "~/wrk/.ve/dotfiles/var/run",
    "_VARSPOOL": "~/wrk/.ve/dotfiles/var/spool",
    "_VARTMP": "~/wrk/.ve/dotfiles/var/tmp",
    "_WWW": "~/wrk/.ve/dotfiles/var/www",
    "_USRLOG": "~/wrk/.ve/dotfiles/.usrlog",
    "HISTFILE": "~/wrk/.ve/dotfiles/.bash_history",
    "HISTSIZE": 1000000,
    "HISTFILESIZE": 1000000,
    "PAGER": "/usr/bin/less -R",
    "_APP": "dotfiles",
    "_WRD": "~/wrk/.ve/dotfiles/src/dotfiles",
    "VIMBIN": "/usr/bin/vim",
    "GVIMBIN": null,
    "MVIMBIN": null,
    "GUIVIMBIN": null,
    "VIMCONF": "--servername dotfiles",
    "_EDIT_": "/usr/bin/vim -f",
    "EDITOR_": "/usr/bin/vim -f",
    "_IPSESSKEY": "~/wrk/.ve/dotfiles/src/.sessionkey",
    "_NOTEBOOKS": "~/wrk/.ve/dotfiles/src/notebooks",
    "_IPQTLOG": "~/wrk/.ve/dotfiles/.ipqt.log",
    "_WRD_SETUPY": "~/wrk/.ve/dotfiles/src/dotfiles/setup.py",
    "_TEST_": "(cd \"${_WRD}\" && python \"${_WRD_SETUPY}\" test)",
    "_CFG": "~/wrk/.ve/dotfiles/etc/development.ini",
    "_EDITCFG_": "/usr/bin/vim -f ~/wrk/.ve/dotfiles/etc/development.ini",
    "_SHELL_": "(cd \"${_WRD}\" && \"${_BIN}\"/pshell \"${_CFG}\")",
    "_SERVE_": "(cd \"${_WRD}\" && \"${_BIN}\"/pserve --app-name=main --reload --monitor-restart \"$
    "PROJECT_FILES": "README.rst CHANGES.rst TODO.rst Makefile setup.py requirements.txt .hgignore .g
    "_SVCFG": "~/wrk/.ve/dotfiles/etc/supervisord.conf",
    "_SVCFG_": " -c \"~/wrk/.ve/dotfiles/etc/supervisord.conf\""
  },
  "aliases": {
    "cdb": "cd \"${_BIN}\"/%l",
    "cde": "cd \"${_ETC}\"/%l",
    "cdh": "cd \"${HOME}\"/%l",
    "cdl": "cd \"${_LIB}\"/%l",
    "cdlog": "cd \"${_LOG}\"/%l",
    "cdp": "cd \"${PROJECT_HOME}\"/%l",
    "cdph": "cd \"${PROJECT_HOME}\"/%l",
    "cdpylib": "cd \"${_PYLIB}\"/%l",
    "cdpysite": "cd \"${_PYSITE}\"/%l",
    "cds": "cd \"${_SRC}\"/%l",
    "cdv": "cd \"${VIRTUAL_ENV}\"/%l",
    "cdve": "cd \"${VIRTUAL_ENV}\"/%l",
    "cdvar": "cd \"${_VAR}\"/%l",
    "cdwh": "cd \"${WORKON_HOME}\"/%l",
    "cdwrk": "cd \"${WORKON_HOME}\"/%l",
    "cdw": "cd \"${_WRD}\"/%l",
    "cd-": "cd \"${_WRD}\"/%l",
    "cdww": "cd \"${_WWW}\"/%l",
    "cdwww": "cd \"${_WWW}\"/%l",
```

```
    "cdhelp": "set | grep \"^cd.*()\" | cut -f1 -d\" \" #%l",
    "edit-": "${_EDIT_} %l",
    "gvim-": "/usr/bin/vim -f",
    "ipskey": "(python -c \"import os; print os.urandom(128).encode(\\\"base64\\\")\" > \"${_IPSESSK
    "ipnb": "ipython notebook --secure --Session.keyfile=\"${_IPSESSKEY}\" --notebook-dir=\"${_NOTEBO
    "ipqt": "ipython qtconsole --secure --Session.keyfile=\"${_IPSESSKEY}\" --logappend=\"${_IPQTLOG
    "grinv": "grin --follow %l \"${VIRTUAL_ENV}\"",
    "grindv": "grind --follow %l --dirs \"${VIRTUAL_ENV}\"",
    "grins": "grin --follow %l \"${_SRC}\"",
    "grinds": "grind --follow %l --dirs \"${_SRC}\"",
    "test-": "(cd \"${_WRD}\" && python \"${_WRD_SETUPY}\" test)",
    "testr-": "reset && (cd \"${_WRD}\" && python \"${_WRD_SETUPY}\" test)",
    "nose-": "(cd \"${_WRD}\" && nosetests)",
    "grinw": "grin --follow %l \"${_WRD}\"",
    "grin-": "grin --follow %l \"${_WRD}\"",
    "grindw": "grind --follow %l --dirs \"${_WRD}\"",
    "grind-": "grind --follow %l --dirs \"${_WRD}\"",
    "hgv-": "hg view -R \"${_WRD}\"",
    "hgl-": "hg -R \"${_WRD}\" log",
    "editcfg": "\"${_EDITCFG_}\" %l",
    "serve-": "(cd \"${_WRD}\" && \"${_BIN}\"/pserve --app-name=main --reload --monitor-restart \"${_
    "shell-": "(cd \"${_WRD}\" && \"${_BIN}\"/pshell \"${_CFG}\")",
    "e": "${_EDIT_} %l",
    "editp": "$GUIVIMBIN $VIMCONF $PROJECT_FILES %l",
    "makewrd": "(cd \"${_WRD}\" && make %l)",
    "make-": "(cd \"${_WRD}\" && make %l)",
    "mw": "(cd \"${_WRD}\" && make %l)",
    "ssv": "supervisord -c \"${_SVCFG}\"",
    "sv": "supervisorctl -c \"${_SVCFG}\"",
    "svt": "sv tail -f",
    "svd": "supervisorctl -c \"${_SVCFG}\" restart dev && supervisorctl -c \"${_SVCFG}\" tail -f dev'
  }
}
```

CHAPTER 4

Tools

4.1 Packages

Wikipedia: https://en.wikipedia.org/wiki/Package_(package_management_system)

A software package is an archive of files with a manifest that lists the files included. Often, the manifest contains file checksums and a *signature*.

Many packaging tools make a distinction between source and/or binary packages.

Some packaging tools provide configuration options for:

- Scripts to run when packaging
- Scripts to run at install time
- Scripts to run at uninstal time
- Patches to apply to the "*vanilla*" source tree, as might be obtained from a version control repository

There is a package maintainer whose responsibilities include:

- Testing new *upstream* releases
- *Vetting* changes from release to release
- *Repackaging* upstream releases
- *Signing* new package releases

Packaging lag refers to how long it takes a package maintainer to repackage upstream releases for the target platform(s).

4.1.1 Apt

Wikipedia: https://en.wikipedia.org/wiki/Advanced_Packaging_Tool
Homepage: http://alioth.debian.org/projects/apt
Docs: https://wiki.debian.org/Apt
Docs: https://www.debian.org/doc/manuals/debian-reference/ch02.en.html
Docs: https://www.debian.org/doc/manuals/apt-howto/
Docs: https://wiki.debian.org/SecureApt
Source: git git://anonscm.debian.org/git/apt/apt.git

IRC: irc://irc.debian.org/debian-apt

APT ("Advanced Packaging Tool") is the core of Debian package management.

An APT package repository serves *Dpkg* packages.

An APT package repository can be accessed from a local filesystem or over a network protocol ("apt transports") like HTTP, HTTPS, RSYNC, FTP, and BitTorrent.

An example of APT usage (e.g. to maintain an updated *Ubuntu Linux* system):

```
apt-get update
apt-get upgrade
apt-get dist-upgrade

apt-cache show bash
apt-get install bash

apt-get --help
man apt-get
man sources.list
```

4.1.2 Bower

Wikipedia: https://en.wikipedia.org/wiki/Bower_(software)
Homepage: https://www.bower.io/
Source: https://github.com/bower/bower

Bower is "a package manager for the web" (*JavaScript* packages) built on *NPM*.

4.1.3 DEB

Wikipedia: https://en.wikipedia.org/wiki/Deb_(file_format)

DEB is the Debian software package format.

DEB packages are built with *Dpkg* and often hosted in an *Apt* package repository.

4.1.4 Dpkg

Wikipedia: https://en.wikipedia.org/wiki/Dpkg
Homepage: http://wiki.debian.org/Teams/Dpkg
Docs: https://en.wikipedia.org/wiki/Debian_build_toolchain
Docs: https://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics.en.html
Docs: https://www.debian.org/doc/manuals/debian-faq/ch-pkgtools.en.html
Docs:

Dpkg is a collection of tools for creating and working with *DEB* packages.

4.1.5 Homebrew

Wikipedia: https://en.wikipedia.org/wiki/Homebrew_(package_management_software)
Homepage: http://brew.sh/

Homebrew is a package manager (`brew`) for *OS X*.

4.1.6 NPM

Wikipedia: https://en.wikipedia.org/wiki/Npm_(software)
Homepage: https://www.npmjs.org/
Source: https://github.com/npm/npm

NPM is a *JavaScript* package manager created for *Node.js*.

Bower builds on NPM.

4.1.7 NuGet

Wikipedia: https://en.wikipedia.org/wiki/NuGet
Homepage: https://www.nuget.org/

- Package Repositories (chocolatey):
  - https://chocolatey.org/
- Linux/Mac/Windows: No / No / Yes

4.1.8 Portage

Wikipedia: https://en.wikipedia.org/wiki/Portage_(software)
Homepage: http://wiki.gentoo.org/wiki/Project:Portage

- Build recipes with flag sets
- Package Repositories (portage)

4.1.9 Ports

Wikipedia: https://en.wikipedia.org/wiki/Ports_collection
Homepage: https://www.freebsd.org/ports/

Sources and Makefiles designed to compile software packages for particular distributions' kernel and standard libraries on a particular platform.

4.1.10 RPM

Wikipedia: https://en.wikipedia.org/wiki/RPM_Package_Manager

- Install with `rpm`, `yum`
- Build with tools like `rpmbuild` and `fpm`
- Python: build with `bdist_rpm`, `fpm`
- List contents:

```
less ~/path/to/local.rpm   # requires lesspipe to be configured
```

- Package Repositories (yum):
  - Local: directories of packages and metadata
  - Network: HTTP, HTTPS, RSYNC, FTP

4.1.11 Python Packages

Homepage: https://pypi.python.org/pypi
Docs: https://packaging.python.org/en/latest/
Docs: https://packaging.python.org/en/latest/peps.html
Docs: https://packaging.python.org/en/latest/projects.html

A Python Package is a collection of source code and package data files.

- Python packages have dependencies: they depend on other packages
- Python packages can be served from a package index
- *PyPI* is the community Python Package Index
- A Python package is an archive of files (`.zip` (`.egg`, `.whl`), `.tar`, `.tar.gz`,) containing a `setup.py` file containing a version string and metadata that is meant for distribution.
- An source dist (`sdist`) package contains source code (every file listed in or matching a pattern in a `MANIFEST.in` text file).
- A binary dist (`bdist`, `bdist_egg`, `bdist_wheel`) is derived from an sdist and may be compiled and named for a specific platform.
- sdists and bdists are defined by a `setup.py` file which contains a call to a `distutils.setup()` or `setuptools.setup()` function.
- The arguments to the `setup.py` function are things like `version`, `author`, `author_email`, and `homepage`; in addition to package dependency strings required for the package to work (`install_requires`), for tests to run (`tests_require`), and for optional things to work (`extras_require`).
- A package dependency string can specify an exact version (==) or a greater-than (>=) or less-than (<=) requirement for each package.
- Package names are looked up from an index server (`--index`), such as *PyPI*, and or an HTML page (`--find-links`) containing URLs containing package names, version strings, and platform strings.
- `easy_install` (*Setuptools*) and *Pip* can install packages from: the local filesystem, a remote index server, or a local index server.

- `easy_install` and `pip` read the `install_requires` (and `extras_require`) attributes of `setup.py` files contained in packages in order to resolve a dependency graph (which can contain cycles) and install necessary packages.

Distuils

Docs: https://docs.python.org/2/distutils/

Distutils is a collection of tools for common packaging needs.

Distutils is included in the Python standard library.

Setuptools

Wikipedia: https://en.wikipedia.org/wiki/Setuptools
Docs: https://pythonhosted.org/setuptools/
Source: hg https://bitbucket.org/pypa/setuptools
PyPI: http://pypi.python.org/pypi/setuptools

Setuptools is a *Python package* for working with other *Python Packages*.

- Setuptools builds upon *Distuils*
- Setuptools is widely implemented
- Most Python packages are installed by setuptools (by *Pip*)
- Setuptools can be installed by downloading `ez_setup.py` and then running `python ez_setup.py`; or, setuptools can be installed with a system package manager (apt, yum)
- Setuptools installs a script called `easy_install` which can be used to install packages from the local filesystem, a remote index server, a local index server, or an HTML page
- `easy_install pip` installs *Pip* from PyPI
- Like `easy_install`, *Pip* installs python packages, with a number of additional configuration options
- Setuptools can build *RPM* and *DEB* packages from python packages, with some extra configuration:

```
``python setup.py bdist_rpm --help``
``python setup.py --command-packages=stdeb.command bdist_deb --help``
```

Pip

Wikipedia: https://en.wikipedia.org/wiki/Pip_(package_manager)
Homepage: http://www.pip-installer.org/
Docs: http://www.pip-installer.org/en/latest/user_guide.html
Docs: https://pip.readthedocs.org/en/latest/
Source: git https://github.com/pypa/pip
Pypi: https://pypi.python.org/pypi/pip
IRC: #pypa
IRC: #pypa-dev

Pip is a tool for installing, upgrading, and uninstalling *Python* packages.

```
pip help
pip help install
pip --version

sudo apt-get install python-pip
pip install --upgrade pip

pip install libcloud
pip install -r requirements.txt
pip uninstall libcloud
```

- Pip stands upon *Distuils* and *Setuptools*.
- Pip retrieves, installs, upgrades, and uninstalls packages.
- Pip can list installed packages with `pip freeze` (and `pip list`).
- Pip can install packages as 'editable' packages (`pip install -e`) from version control repository URLs which must begin with `vcs+`, end with `#egg=<usuallythepackagename>`, and may contain an `@vcstag` tag (such as a branch name or a version tag).
- Pip installs packages as editable by first cloning (or checking out) the code to `./src` (or `${VIRTUAL_ENV}/src` if working in a *Virtualenv*) and then running `setup.py develop`.
- Pip configuration is in `${HOME}/.pip/pip.conf`.
- Pip can maintain a local cache of downloaded packages, which can lessen the load on package servers during testing.
- Pip skips reinstallation if a package requirement is already satisfied.
- Pip requires the `--upgrade` and/or `--force-reinstall` options to be added to the `pip install` command in order to upgrade or reinstall.
- At the time of this writing, the latest stable pip version is `1.5.6`.

Warning: With *Python* 2, pip is preferable to *Setuptools*'s `easy_install` because pip installs `backports.ssl_match_hostname` in order to validate `HTTPS` certificates (by making sure that the certificate hostname matches the hostname from which the DNS resolved to).
Cloning packages from source repositories over `ssh://` or `https://`, either manually or with `pip install -e` avoids this concern.
There is also a tool called *Peep* which requires considered-good SHA256 checksums to be specified for every dependency listed in a `requirements.txt` file.
For more information, see: http://legacy.python.org/dev/peps/pep-0476/#python-versions

Pip Requirements File Plaintext list of packages and package URIs to install.

Requirements files may contain version specifiers (`pip >= 1.5`)

Pip installs Pip Requirement Files:

```
pip install -r requirements.txt
pip install --upgrade -r requirements.txt
pip install --upgrade --user --force-reinstall -r requirements.txt
```

An example `requirements.txt` file:

```
# install pip from the default index (PyPI)
pip
--index=https://pypi.python.org/simple --upgrade pip
```

```
# Install pip 1.5 or greater from PyPI
pip >= 1.5

# Git clone and install pip as an editable develop egg
-e git+https://github.com/pypa/pip@1.5.X#egg=pip

# Install a source distribution release from PyPI
# and check the MD5 checksum in the URL
https://pypi.python.org/packages/source/p/pip/pip-1.5.5.tar.gz#md5=7520581ba0687dec1ce85bd154965

# Install a source distribution release from Warehouse
https://warehouse.python.org/packages/source/p/pip/pip-1.5.5.tar.gz

# Install an additional requirements.txt file
-r requirements/more-requirements.txt
```

Peep

Source: https://github.com/erikrose/peep
PyPI: https://pypi.python.org/pypi/peep

Peep works just like *Pip*, but requires `SHA256` checksum hashes to be specified for each package in `requirements.txt` file.

PyPI

Wikipedia: https://en.wikipedia.org/wiki/Python_Package_Index
Docs: http://wiki.python.org/moin/CheeseShop
Docs: http://wiki.python.org/moin/CheeseShopDev
Homepage: https://pypi.python.org/
Source: https://bitbucket.org/pypa/pypi

PyPI is the Python Package Index.

Warehouse

Homepage: https://warehouse.python.org/
Docs: https://warehouse.readthedocs.org/
Source: https://github.com/pypa/warehouse

Warehouse is the "Next Generation Python Package Repository".

All packages uploaded to *PyPI* are also available from Warehouse.

Wheel

Docs: http://legacy.python.org/dev/peps/pep-0427/
Docs: http://wheel.readthedocs.org/en/latest/

Source: hg https://bitbucket.org/pypa/wheel/
PyPI: https://pypi.python.org/pypi/wheel

- Wheel is a newer, PEP-based standard (`.whl`) with a different metadata format, the ability to specify (JSON) digital signatures for a package within the package, and a number of additional speed and platform-consistency advantages.
- Wheels can be uploaded to PyPI.
- Wheels are generally faster than traditional Python packages.

Packages available as wheels are listed at http://pythonwheels.com/.

Conda

Docs: http://conda.pydata.org/docs/
Source: git https://github.com/conda/conda
PyPI: https://pypi.python.org/pypi/conda

- Conda installs packages written in any language; especially Python
- `conda skeleton` can automatically create conda packages both from `PyPI` and from `CPAN` (Perl)
- Conda was originally created for the Anaconda Python Distribution, which installs packages written in *Python*, R, Javascript, *Ruby*, C, Fortran
- Conda (and *Anaconda*) packages are hosted by <*https://binstar.org*>, which hosts free public and paid private Conda packages.

4.1.12 RubyGems

Wikipedia: https://en.wikipedia.org/wiki/RubyGems
Homepage: https://rubygems.org/
Docs: http://guides.rubygems.org/
Source: https://github.com/rubygems/rubygems

- RubyGems installs Ruby Gems

4.1.13 Yum

Wikipedia: https://en.wikipedia.org/wiki/Yellowdog_Updater,_Modified
Homepage: http://yum.baseurl.org/

Yum is a tool for installing, upgrading, and uninstalling *RPM* packages.

4.2 Anaconda

Wikipedia: https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)
Homepage: https://store.continuum.io/cshop/anaconda/

Docs: http://docs.continuum.io/anaconda/
Docs: http://docs.continuum.io/anaconda/pkg-docs.html

Anaconda is a maintained distribution of many popular *Python Packages*.

Anaconda works with *Conda* packages.

Note: https://en.wikipedia.org/wiki/Anaconda_(installer) (1999) is the installer for *RPM*-based *Linux* distributions; which is also written in *Python* (and *C*).

4.3 Bash

Wikipedia: https://en.wikipedia.org/wiki/Bash_(Unix_shell)
Homepage: http://www.gnu.org/software/bash/
Docs: https://www.gnu.org/software/bash/manual/
Source: git git://git.savannah.gnu.org/bash.git

Bash, the Bourne-again shell.

```
type bash
bash --help
help help
help type
apropos bash
info bash
man bash
```

- Designed to work with unix command outputs and return codes
- Functions
- Portability: sh (sh, bash, dash, zsh) shell scripts are mostly compatible
- Logging:

  ```
  set -x  # print commands and arguments
  set -v  # print source
  ```

Bash Configuration:

```
/etc/profile
/etc/bash.bashrc
/etc/profile.d/*.sh
${HOME}/.profile        /etc/skel/.profile   # PATH=+$HOME/bin  # umask
${HOME}/.bash_profile   # empty. preempts .profile
```

Linux/Mac/Windows: Almost Always / Bash 3.2 / Cygwin/Mingwin

4.4 C

Wikipedia: https://en.wikipedia.org/wiki/C_(programming_language)
Docs: http://learnxinyminutes.com/docs/c/

C is a third-generation programming language which affords relatively low-level machine access while providing helpful abstractions.

The GNU/*Linux* kernel is written in C and compiled by *GCC*.

4.5 C++

Wikipedia: https://en.wikipedia.org/wiki/C++
Docs: http://learnxinyminutes.com/docs/c++/

C++ is a third-generation programming language which adds object orientation and a standard library to *C*.

4.6 Compiz

Wikipedia: https://en.wikipedia.org/wiki/Compiz
Homepage: https://launchpad.net/compiz
Docs: http://wiki.compiz.org/
Source: bzr branch lp:compiz

Compiz is a window compositing layer for *X11* which adds lots of cool and productivity-enhancing visual capabilities.

4.7 CoreOS

Wikipedia: https://en.wikipedia.org/wiki/CoreOS
Homepage: https://coreos.com/
Docs: https://coreos.com/docs/
Source: https://github.com/coreos

CoreOS is *Linux* distribution for highly available distributed computing.

CoreOS schedules redundant *Docker* images with **fleet** and **systemd** according to configuration stored in **etcd**, a key-value store with a D-Bus interface.

4.8 Docker

Wikipedia: https://en.wikipedia.org/wiki/Docker_(software)
Homepage: https://docker.io/
Docs: http://docs.docker.io/
Source: https://github.com/dotcloud/docker

Docker is an OS virtualization project which utilizes Linux LXC Containers to partition process workloads all running under one kernel.

Limitations

- Writing to */etc/hosts*: https://github.com/dotcloud/docker/issues/2267
- Apt-get upgrade: https://github.com/dotcloud/docker/issues/3934

4.9 Docutils

Homepage: http://docutils.sourceforge.net
Docs: http://docutils.sourceforge.net/docs/
Docs: http://docutils.sourceforge.net/rst.html
Docs: http://docutils.sourceforge.net/docs/ref/doctree.html
Source: svn http://svn.code.sf.net/p/docutils/code/trunk

Docutils is a text processing system which 'parses" *ReStructuredText* lightweight markup language into a doctree which it serializes into HTML, LaTeX, man-pages, Open Document files, XML, and a number of other formats.

4.10 Fortran

Wikipedia: https://en.wikipedia.org/wiki/Fortran

Fortran (or FORTRAN) is a third-generation programming language frequently used for mathematical and scientific computing.

4.11 Filesystem Hierarchy Standard

Wikipedia: https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard
Website: http://www.linuxfoundation.org/collaborate/workgroups/lsb/fhs

The Filesystem Hierarchy Standard is a well-worn industry-supported system file naming structure.

Ubuntu and *Virtualenv* implement a Filesystem Hierarchy.

Docker layers filesystem hierarchies with aufs and now also btrfs subvolumes.

4.12 GCC

Wikipedia: https://en.wikipedia.org/wiki/GNU_Compiler_Collection
Homepage: https://gcc.gnu.org/
Docs: https://gcc.gnu.org/onlinedocs/
Source: git ssh://gcc.gnu.org/git/gcc.git

The GNU Compiler Collection started as a Free and Open Source compiler for *C*.

There are now GCC frontends for many languages, including *C++*, *Fortran*, *Java*, and *Go*.

4.13 Git

Wikipedia: https://en.wikipedia.org/wiki/Git_(software)
Homepage: http://git-scm.com/
Docs: http://git-scm.com/documentation
Docs: http://git-scm.com/book/en/
Docs: http://documentup.com/skwp/git-workflows-book
Docs: http://learnxinyminutes.com/docs/git/
Source: git https://github.com/git/git

Git is a distributed version control system for tracking a branching and merging repository of file revisions.

4.14 Gnome

Wikipedia: https://en.wikipedia.org/wiki/GNOME
Homepage: http://www.gnome.org/
Docs: https://help.gnome.org/
Source: https://git.gnome.org/browse/

- https://wiki.gnome.org/GnomeLove

4.15 Go

Wikipedia: https://en.wikipedia.org/wiki/Go_(programming_language)
Homepage: http://golang.org/
Docs: http://golang.org/doc/
Source: hg https://code.google.com/p/go/

Go is a relatively new statically-typed C-based language.

4.16 Htop

Wikipedia: https://en.wikipedia.org/wiki/Htop
Homepage: http://hisham.hm/htop/
Source: git http://hisham.hm/htop/

4.17 I3wm

Wikipedia: https://en.wikipedia.org/wiki/I3_(window_manager)
Homepage: http://i3wm.org/

Docs: http://i3wm.org/docs/
Source: git git://code.i3wm.org/i3

- http://i3wm.org/downloads/

4.18 IPython

Wikipedia: https://en.wikipedia.org/wiki/IPython
Homepage: http://ipython.org/
Docs: http://ipython.org/ipython-doc/stable/
Source: git https://github.com/ipython/ipython

- https://registry.hub.docker.com/u/ipython
- https://registry.hub.docker.com/u/jupyter
- https://github.com/jupyter

4.19 Java

Wikipedia: https://en.wikipedia.org/wiki/Java_(programming_language)
Docs: http://javadocs.org/
Docs: http://learnxinyminutes.com/docs/java/

Java is a third-generation programming language which is compiled into code that runs in a virtual machine (`JVM`) written in *C* for many different operating systems.

4.20 JavaScript

Wikipedia: https://en.wikipedia.org/wiki/JavaScript
Docs: https://en.wikipedia.org/wiki/ECMAScript
Docs: http://learnxinyminutes.com/docs/javascript/

JavaScript is a third-generation programming language designed to run in an interpreter; now specified as *ECMAScript*.

All major web browsers support Javascript.

Client-side (web) applications can be written in Javascript.

Server-side (web) applications can be written in Javascript, often with *Node.js* and *NPM* packages.

Note: Java and JavaScript are two distinctly different languages and developer ecosystems.

4.21 JSON

Wikipedia: https://en.wikipedia.org/wiki/JSON
Homepage: http://json.org/
Docs: http://learnxinyminutes.com/docs/json/

JSON is an object representation in *JavaScript* syntax which is now supported by libraries for many language.

A list of objects with `key` and `value` attributes in JSON syntax: .. code-block:: javascript

> [{ "key": "language", "value": "Javascript" }, { "key": "version", "value": 1 }, { "key": "example", "value": true },]

Machine-generated JSON is often not very readable, because it doesn't contain extra spaces or newlines. The *Python* JSON library contains a utility for parsing and indenting ("prettifying") JSON from the commandline

```
cat example.json | python -m json.tool
```

4.22 KDE

Wikipedia: https://en.wikipedia.org/wiki/KDE
Homepage: http://kde.org/
Docs: https://docs.kde.org/
Docs: https://www.kde.org/documentation/
Source: https://techbase.kde.org/Getting_Started/Sources
Source: https://techbase.kde.org/Getting_Started/Sources/Subversion
Source: https://techbase.kde.org/Development/Git
Source: https://projects.kde.org/projects

KDE is a GUI framework built on Qt.

KWin is the main KDE window manager for *X11*.

4.23 Libcloud

Homepage: https://libcloud.apache.org/
Docs: https://libcloud.readthedocs.org/
Docs: https://libcloud.readthedocs.org/en/latest/supported_providers.html
Source: git git://git.apache.org/libcloud.git
Source: git https://github.com/apache/libcloud

Apache Libcloud is a *Python* library which abstracts and unifies a large number of Cloud APIs for Compute Resources, Object Storage, Load Balancing, and DNS.

4.24 Libvirt

Wikipedia: http://libvirt.org/
Homepage: http://libvirt.org/
Docs: http://libvirt.org/docs.html
Docs: http://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.virt.html
Source: git git://libvirt.org/libvirt-appdev-guide.git

Libvirt is a system for platform virtualization with various *Linux* hypervisors.

- KVM/QEMU
- Xen
- LXC
- OpenVZ
- VirtualBox

4.25 Linux

Wikipedia: https://en.wikipedia.org/wiki/Linux
Homepage: https://www.kernel.org/
Docs: https://www.kernel.org/doc/
Source: git https://github.com/torvalds/linux

GNU/Linux is a free and open source operating system kernel written in *C*.

```
uname -a; echo "Linux"
uname -o; echo "GNU/Linux"
```

A *Linux Distribution* is a collection of *Packages* compiled to work with a GNU/Linux kernel.

4.26 Make

Wikipedia: https://en.wikipedia.org/wiki/Make_(software)
Homepage: https://www.gnu.org/software/make/
Project: https://savannah.gnu.org/projects/make/
Docs: https://www.gnu.org/software/make/manual/make.html
Source: git git://git.savannah.gnu.org/make.git

GNU Make is a classic, ubiquitous software build tool designed for file-based source code compilation.

Bash, *Python*, and the GNU/*Linux* kernel are all built with Make.

Make build task chains are represented in a `Makefile`.

Pros

- Simple, easy to read syntax

- Designed to build files on disk
- Nesting: `make -C <path> <taskname>`
- Variable Syntax: `$(VARIABLE_NAME)`
- Bash completion: `make <tab>`
- Python: Parseable with disutils.text_file Text File
- Logging: command names and values to stdout

Cons

- Platform Portability: make is not installed everywhere
- Global Variables: Parametrization with shell scripts

4.27 Mercurial

Wikipedia: https://en.wikipedia.org/wiki/Mercurial
Homepage: http://hg.selenic.org/
Docs: http://mercurial.selenic.com/guide
Source: hg http://selenic.com/hg
Source: hg http://hg.intevation.org/mercurial/crew

- http://hgbook.red-bean.com/

4.28 MessagePack

Wikipedia: https://en.wikipedia.org/wiki/MessagePack
Homepage: http://msgpack.org/

MessagePack is a data interchange format with implementations in many languages.

Salt

4.29 Node.js

Wikipedia: https://en.wikipedia.org/wiki/Node.js
Homepage: http://www.nodejs.org
Source: https://github.com/joyent/node

Node.js is a framework for *JavaScript* applications written in *C*, *C++*, and *JavaScript*.

4.30 OS X

Wikipedia: https://en.wikipedia.org/wiki/OS_X

Homepage: http://www.apple.com/osx
Docs: https://developer.apple.com/technologies/mac/
Source: https://www.apple.com/opensource/

OS X is a UNIX operating system based upon the Mach kernel from NeXTSTEP, which was partially derived from NetBSD and FreeBSD.

OS X GUI support is built from XFree86/X.org *X11*.

OS X maintains forks of many POSIX BSD and GNU tools like `bash`, `readlink`, and `find`.

Homebrew installs and maintains packages for OS X.

```
uname; echo "Darwin"
```

4.31 Packer

Homepage: http://www.packer.io/
Docs: http://www.packer.io/docs
Docs: http://www.packer.io/docs/basics/terminology.html
Source: git https://github.com/mitchellh/packer

Packer generates machine images for multiple platforms, clouds, and hypervisors from a parameterizable template.

Packer Artifact Build products: machine image and manifest

Packer Template JSON build definitions with optional variables and templating

Packer Build Task defined by a JSON file containing build steps which produce a machine image

Packer Builder Packer components which produce machine images for one of many platforms:

- VirtualBox
- Docker
- OpenStack
- GCE
- EC2
- VMware
- QEMU (KVM, Xen)
- http://www.packer.io/docs/templates/builders.html

Packer Provisioner Packer components for provisioning machine images at build time

- Shell scripts
- File uploads
- ansible
- chef
- solo
- puppet

- salt

Packer Post-Processor Packer components for compressing and uploading built machine images

4.32 Perl

Wikipedia: https://en.wikipedia.org/wiki/Perl
Homepage: http://www.perl.org/
Project: http://dev.perl.org/perl5/
Docs: http://www.perl.org/docs.html
Source: git git://perl5.git.perl.org/perl.git

Perl is a dynamically typed, C-based scripting language.

Many of the Debian system management tools are or were originally written in Perl.

4.33 Python

Wikipedia: https://en.wikipedia.org/wiki/Python_(programming_language)
Homepage: https://www.python.org/
Docs: https://docs.python.org/2/
Docs: https://docs.python.org/devguide/
Docs: https://docs.python.org/devguide/documenting.html
Docs: http://learnxinyminutes.com/docs/python/
Source: hg https://hg.python.org/cpython

Python is a dynamically-typed, C-based scripting language.

Many of the RedHat system management tools (such as *Yum*) are written in Python. Gentoo *Portage* is written in Python.

IPython, *Pip*, *Conda*, *Sphinx*, *Docutils*, *Mercurial*, *Libcloud*, *Salt*, *Tox*, *Virtualenv*, and *Virtualenvwrapper* are all written in Python.

4.33.1 Python 3

Docs: https://docs.python.org/3/
Docs: https://docs.python.org/3/howto/pyporting.html
Docs: https://docs.python.org/3/howto/cporting.html
Docs: http://learnxinyminutes.com/docs/python3/

Python 3 made a number of incompatible changes, requiring developers to update and review their Python 2 code in order to "port to" Python 3.

Python 2 will be supported in "no-new-features" status for quite some time.

Python 3 Wall of Superpowers tracks which popular packages have been ported to support Python 3: https://python3wos.appspot.com/

There are a number of projects which help bridge the gap between the two language versions:

- https://pypi.python.org/pypi/six
- http://pythonhosted.org/six/
- https://pypi.python.org/pypi/nine
- https://github.com/nandoflorestan/nine/blob/master/nine/__init__.py
- https://pypi.python.org/pypi/future
- http://python-future.org/

The Anaconda Python distribution (*Conda*) maintains a working set of packages for Python 2.6, 2.7, 3.3, and 3.4: http://docs.continuum.io/anaconda/pkg-docs.html

4.33.2 awesome-python-testing

Homepage: https://westurner.github.io/wiki/awesome-python-testing.html
Source: https://github.com/westurner/wiki/blob/master/awesome-python-testing.rest

4.34 Readline

Wikipedia: https://en.wikipedia.org/wiki/GNU_Readline
Homepage: http://tiswww.case.edu/php/chet/readline/rltop.html
Docs: http://tiswww.case.edu/php/chet/readline/readline.html
Docs: http://tiswww.case.edu/php/chet/readline/history.html
Docs: http://tiswww.case.edu/php/chet/readline/rluserman.html
Source: ftp ftp://ftp.gnu.org/gnu/readline/readline-6.3.tar.gz

- https://pypi.python.org/pypi/gnureadline

4.35 ReStructuredText

Wikipedia: https://en.wikipedia.org/wiki/ReStructuredText
Homepage: http://docutils.sourceforge.net/rst.html
Docs: http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html
Docs: http://docutils.sourceforge.net/docs/ref/rst/directives.html
Docs: http://docutils.sourceforge.net/docs/ref/rst/roles.html
Docs: http://sphinx-doc.org/rest.html

ReStructuredText (RST, ReST) is a plaintext lightweight markup language commonly used for narrative documentation and Python docstrings.

Sphinx is built on *Docutils*, which is the primary implementation of ReStructuredText.

Pandoc also supports a form of ReStructuredText.

ReStructuredText Directive Actionable blocks of ReStructuredText

```
.. include:: goals.rst

.. contents:: Table of Contents
   :depth: 3

.. include:: LICENSE
```

ReStructuredText Role RestructuredText role extensions

```
.. _anchor-name:

:ref:`Anchor <anchor-name>`
```

4.36 Ruby

Wikipedia: https://en.wikipedia.org/wiki/Ruby_(programming_language)
Homepage: https://www.ruby-lang.org/
Docs: https://www.ruby-lang.org/en/documentation/
Docs: http://learnxinyminutes.com/docs/ruby/
Source: svn http://svn.ruby-lang.org/repos/ruby/trunk

Ruby is a dynamically-typed programming language.

Vagrant is written in Ruby.

4.37 Salt

Wikipedia: https://en.wikipedia.org/wiki/Salt_(software)
Homepage: http://www.saltstack.com
Docs: https://docs.saltstack.com/en/latest/
Docs: https://docs.saltstack.com/en/latest/salt-modindex.html
Docs: https://docs.saltstack.com/en/latest/ref/states/all/index.html
Docs: https://docs.saltstack.com/en/latest/ref/clients/index.html#python-api
Docs: https://docs.saltstack.com/en/latest/topics/development/hacking.html
Docs: https://docs.saltstack.com/en/latest/glossary.html
Source: git https://github.com/saltstack/salt
Pypi: https://pypi.python.org/pypi/salt
IRC: #salt

Salt is an open source configuration management system for managing one or more physical and virtual machines running various operating systems.

Salt Top File Root of a Salt Environment (`top.sls`)

Salt Environment Folder of Salt States with a top.sls top file.

Salt Bootstrap Installer for salt master and/or salt minion

Salt Minion Daemon process which executes Salt States on the local machine.

Can run as a background daemon. Can retrieve and execute states from a salt master

Can execute local states in a standalone minion setup:

```
salt-call --local grains.items
```

Salt Minion ID Machine ID value uniquely identifying a minion instance to a Salt Master.

By default the minion ID is set to the FQDN

```
python -c 'import socket; print(socket.getfqdn())'
```

The minion ID can be set explicitly in two ways:

- /etc/salt/minion.conf:

  ```
  id: devserver-123.example.org
  ```

- /etc/salt/minion_id:

  ```
  $ hostname -f > /etc/salt/minion_id
  $ cat /etc/salt/minion_id
  devserver-123.example.org
  ```

Salt Master Server daemon which compiles pillar data for and executes commands on Salt Minions:

```
salt '*' grains.items
```

Salt SSH Execute salt commands and states over SSH without a minion process:

```
salt-ssh '*' grains.items
```

Salt Grains Static system information keys and values

- hostname
- operating system
- ip address
- interfaces

Show grains on the local system:

```
salt-call --local grains.items
```

Salt Modules Remote execution functions for files, packages, services, commands.

Can be called with salt-call

Salt States Graphs of nodes and attributes which are templated and compiled into ordered sequences of system configuration steps.

Naturally stored in `.sls` *YAML* files parsed by `salt.states.<state>.py`.

Salt States files are processed as Jinja templates (by default) they can access system-specific grains and pillar data at compile time.

Salt Renderers Templating engines (by default: Jinja) for processing templated states and configuration files.

Salt Pillar Key Value data interface for storing and making available global and host-specific values for minions: values like hostnames, usernames, and keys.

Pillar configuration must be kept separate from states (e.g. users, keys) but works the same way.

In a master/minion configuration, minions do not have access to the whole pillar.

Salt Cloud Salt Cloud can provision cloud image, instance, and networking services with various cloud providers (libcloud):

- Google Compute Engine (GCE) [KVM]
- Amazon EC2 [Xen]
- Rackspace Cloud [KVM]
- OpenStack [https://wiki.openstack.org/wiki/HypervisorSupportMatrix]
- Linux LXC (Cgroups)
- KVM

4.38 Sphinx

Wikipedia: https://en.wikipedia.org/wiki/Sphinx_(documentation_generator)
Homepage: https://pypi.python.org/pypi/Sphinx
Docs: http://sphinx-doc.org/contents.html
Docs: http://sphinx-doc.org/markup/code.html
Docs: http://pygments.org/docs/lexers/
Docs: http://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html
Source: hg https://bitbucket.org/birkenfeld/sphinx/
Pypi: https://pypi.python.org/pypi/Sphinx

Sphinx is a tool for working with *ReStructuredText* documentation trees and rendering them into HTML, PDF, LaTeX, ePub, and a number of other formats.

Sphinx extends *Docutils* with a number of useful markup behaviors which are not supported by other ReStructuredText parsers.

Most other ReStructuredText parsers do not support Sphinx directives; so, for example,

- GitHub and BitBucket do not support Sphinx but do support ReStructuredText so README.rst containing Sphinx tags renders in plaintext or raises errors.

  For example, the index page of this *Sphinx* documentation set is generated from a file named `index.rst` and referenced by `docs/conf.py`.

  - Input: https://raw.githubusercontent.com/westurner/provis/master/docs/index.rst
  - Output: https://github.com/westurner/provis/blob/master/docs/index.rst
  - Output: *ReadTheDocs*: http://provis.readthedocs.org/en/latest/

Sphinx Builder Render Sphinx *ReStructuredText* into various forms:

- HTML
- LaTeX
- PDF
- ePub

See: Sphinx Builders

Sphinx ReStructuredText Sphinx extends *ReStructuredText* with roles and directives which only work with Sphinx.

Sphinx Directive Sphinx extensions of *Docutils ReStructuredText* directives.

Most other ReStructuredText parsers do not support Sphinx directives.

```
.. toctree::

   readme
   installation
   usage
```

See: Sphinx Directives

Sphinx Role Sphinx extensions of *Docutils ReStructuredText* roles

Most other ReStructured

```
.. _anchor-name:

:ref:`Anchor <anchor-name>`
```

4.39 Tox

Homepage: https://testrun.org/tox/
Docs: https://tox.readthedocs.org
Source: hg https://bitbucket.org/hpk42/tox
Pypi: https://pypi.python.org/pypi/tox

Tox is a build automation tool designed to build and test Python projects with multiple language versions and environments in separate *virtualenvs*.

Run the py27 environment:

```
tox -v -e py27
tox --help
```

4.40 Ubuntu

Wikipedia: https://en.wikipedia.org/wiki/Ubuntu_(operating_system)
Homepage: http://www.ubuntu.com/
Docs: https://help.ubuntu.com/
Source: https://launchpad.net/ubuntu
Source: http://archive.ubuntu.com/
Source: http://releases.ubuntu.com/

4.41 Vagrant

Wikipedia: https://en.wikipedia.org/wiki/Vagrant_(software)
Homepage: http://www.vagrantup.com/

Docs: http://docs.vagrantup.com/v2/
Source: git https://github.com/mitchellh/vagrant

Vagrant is a tool for creating and managing virtual machine instances with CPU, RAM, Storage, and Networking.

- Vagrant:
  - provides helpful commandline porcelain on top of *VirtualBox* `VboxManage`
  - installs *Vagrant Boxes*

```
vagrant help
vagrant status
vagrant init ubuntu/trusty64
vagrant up
vagrant ssh
$EDITOR Vagrantfile
vagrant provision
vagrant halt
vagrant destroy
```

Vagrantfile Vagrant script defining a team of one or more virtual machines and networks.

Create a Vagrantfile:

```
vagrant init [basebox]
cat Vagrantfile
```

Start virtual machines and networks defined in the Vagrantfile:

```
vagrant status
vagrant up
```

Vagrant Box Vagrant base machine virtual machine image.

There are many baseboxes for various operating systems.

Essentially a virtual disk plus CPU, RAM, Storage, and Networking metadata.

Locally-stored and cached vagrant boxes can be listed with:

```
vagrant help box
vagrant box list
```

A running vagrant environment can be packaged into a new box with:

```
vagrant package
```

Packer generates *VirtualBox* Vagrant Boxes with a Post-Processor.

Vagrant Cloud Vagrant-hosted public Vagrant Box storage.

Install a box from Vagrant cloud:

```
vagrant init ubuntu/trusty64
vagrant up
vagrant ssh
```

Vagrant Provider A driver for running Vagrant Boxes with a hypervisor or in a cloud.

The Vagrant *VirtualBox* Provider is well-supported.

With Plugins: https://github.com/mitchellh/vagrant/wiki/Available-Vagrant-Plugins

See also: *Libcloud*.

Vagrant Provisioner Set of hooks to install and run shell scripts and configuration managment tools over `vagrant ssh`.

Vagrant up runs `vagrant provision` on first invocation of `vagrant up`.

```
vagrant provision
```

Note: Vagrant configures a default NFS share mounted at `/vagrant`.

Note: Vagrant adds a default NAT Adapter as eth0; presumably for DNS, the default route, and to ensure `vagrant ssh` connectivity.

4.42 Vim

Wikipedia: https://en.wikipedia.org/wiki/Vim_(text_editor)
Homepage: http://www.vim.org/
Docs: http://www.vim.org/docs.php
Source: hg https://vim.googlecode.com/hg/

- https://github.com/scrooloose/nerdtree
- https://github.com/westurner/dotvim

4.42.1 Vimium

Wikipedia: https://en.wikipedia.org/wiki/Vimium
Homepage: https://vimium.github.io/
Source: git https://github.com/philc/vimium

- https://chrome.google.com/webstore/detail/vimium/dbepggeogbaibhgnhhndojpepiihcmeb?hl=en

4.42.2 Vimperator

Wikipedia: https://en.wikipedia.org/wiki/Vimperator
Homepage: http://www.vimperator.org/
Source: https://github.com/vimperator/vimperator-labs

- https://addons.mozilla.org/en-US/firefox/addon/vimperator/

4.42.3 Wasavi

Homepage: http://appsweets.net/wasavi/
Docs: http://appsweets.net/wasavi/
Source: https://github.com/akahuku/wasavi

- https://chrome.google.com/webstore/detail/dgogifpkoilgiofhhhodbodcfgomelhe
- https://addons.opera.com/en/extensions/details/wasavi/
- https://addons.mozilla.org/en-US/firefox/addon/wasavi/

4.43 VirtualBox

Wikipedia: https://en.wikipedia.org/wiki/VirtualBox
Homepage: https://www.virtualbox.org/
Docs: https://www.virtualbox.org/wiki/Documentation
Source: svn svn://www.virtualbox.org/svn/vbox/trunk

Oracle VirtualBox is a platform virtualization package for running one or more guest VMs (virtual machines) within a host system.

VirtualBox:

- runs on many platforms: *Linux*, OSX, Windows
- has support for full platform NX/AMD-v virtualization
- requires matching kernel modules

Vagrant scripts VirtualBox.

4.44 Virtualenv

Homepage: http://www.virtualenv.org
Docs: http://www.virtualenv.org/en/latest/
Source: git https://github.com/pypa/virtualenv
PyPI: https://pypi.python.org/pypi/virtualenv
IRC: #pip

Virtualenv is a tool for creating reproducible *Python* environments.

Virtualenv sets the shell environment variable `$VIRTUAL_ENV` when active.

Virtualenv installs a copy of *Python*, *Setuptools*, and *Pip* when a new virtualenv is created.

A virtualenv is activated by `source`-ing `${VIRTUAL_ENV}/bin/activate`.

Paths within a virtualenv are more-or-less *FHS* standard paths, which makes virtualenv structure very useful for building chroot and container overlays.

A standard virtual environment:

```
bin/           # pip, easy_install, console_scripts
bin/activate   # source bin/activate to work on a virtualenv
include/       # (symlinks to) dev headers (python-dev/python-devel)
lib/           # libraries
lib/python2.7/distutils/
lib/python2.7/site-packages/  # pip and easy_installed packages
local/         # symlinks to bin, include, and lib
```

```
src/           # editable requirements (source repositories)

# also useful
etc/           # configuration
var/log        # logs
var/run        # sockets, PID files
tmp/           # mkstemp temporary files with permission bits
srv/           # local data
```

Virtualenvwrapper wraps virtualenv.

```
echo $PATH; echo $VIRTUAL_ENV
python -m site; pip list

virtualenv example               # mkvirtualenv example
source ./example/bin/activate    # workon example

echo $PATH; echo $VIRTUAL_ENV
python -m site; pip list

ls -altr $VIRTUAL_ENV/lib/python*/site-packages/**  # lssitepackages -altr
```

Note: *Venv* extends *Virtualenv* and *Virtualenvwrapper*.

Note: Python 3.3+ now also contain a script called **venv**, which performs the same functions and works similarly to virtualenv: https://docs.python.org/3/library/venv.html.

4.45 Virtualenvwrapper

Docs: http://virtualenvwrapper.readthedocs.org/en/latest/

Source: hg https://bitbucket.org/dhellmann/virtualenvwrapper

PyPI: https://pypi.python.org/pypi/virtualenvwrapper

Virtualenvwrapper is a tool which extends virtualenvwrapper.

Virtualenvwrapper provides a number of useful shell commands and python functions for working with and within *virtualenvs*, as well as project event scripts (e.g. `postactivate`, `postmkvirtualenv`) and two filesystem configuration variables useful for structuring development projects of any language within *virtualenvs*: `$PROJECT_HOME` and `$WORKON_HOME`.

Virtualenvwrapper is sourced into the shell:

```
# pip install --user --upgrade virtualenvwrapper
source ~/.local/bin/virtualenvwrapper.sh

# sudo apt-get install virtualenvwrapper
source /etc/bash_completion.d/virtualenvwrapper
```

Note: *Venv* extends *Virtualenv* and *Virtualenvwrapper*.

```
echo $PROJECT_HOME; echo ~/workspace             # venv: ~/wrk
cd $PROJECT_HOME                                 # venv: cdp; cdph
```

```
echo $WORKON_HOME;  echo ~/.virtualenvs          # venv: ~/wrk/.ve
cd $WORKON_HOME                                  # venv: cdwh; cdwrk

mkvirtualenv example
workon example                                   # venv: we example

cdvirtualenv; cd $VIRTUAL_ENV                    # venv: cdv
echo $VIRTUAL_ENV; echo ~/.virtualenvs/example   # venv: ~/wrk/.ve/example

mkdir src ; cd src/                              # venv: cds; cd $_SRC

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

cd src/dotfiles; cd $VIRTUAL_ENV/src/dotfiles    # venv: cdw; cds dotfiles
head README.rst

                                                 # venv: cdpylib
cdsitepackages                                   # venv: cdpysite
lssitepackages

deactivate
rmvirtualenv example

lsvirtualenvs; ls -d $WORKON_HOME                # venv: lsve; lsve 'ls -d'
```

4.46 Wayland

Wikipedia: https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)
Homepage: http://wayland.freedesktop.org/
Source:

Wayland is a display server protocol for GUI window management.

Wayland is an alternative to *X11* servers like XFree86 and X.org.

The reference Wayland implementation, Weston, is written in *C*.

4.47 YAML

Wikipedia: https://en.wikipedia.org/wiki/YAML
Homepage: http://yaml.org
Docs: http://learnxinyminutes.com/docs/yaml/

YAML ("YAML Ain't Markup Language") is a concise data serialization format.

Most *Salt* states and pillar data are written in YAML. Here's an example `top.sls` file:

```
base:
 '*':
   - openssh
 '*-webserver':
```

```
  - webserver
'*-workstation':
  - gnome
  - i3
```

4.48 ZSH

Wikipedia: https://en.wikipedia.org/wiki/Z_shell
Homepage: http://www.zsh.org/
Docs: http://zsh.sourceforge.net/Guide/zshguide.html
Docs: http://zsh.sourceforge.net/Doc/
Source: git git://git.code.sf.net/p/zsh/code

- https://github.com/robbyrussell/oh-my-zsh

4.49 X11

Wikipedia: https://en.wikipedia.org/wiki/X_Window_System
Homepage: http://www.x.org/
Docs: http://www.x.org/wiki/Documentation/
Source: git git://anongit.freedesktop.org/git/xorg/

X Window System (X, X11) is a display server protocol for window management (drawing windows on the screen).

Most UNIX and *Linux* systems utilize XFree86 or the newer X.org X11 window managers.

Gnome, *KDE*, *I3wm*, *OS X*, and *Compiz* build upon X11.

---

^top^

CHAPTER 5

Dotfiles API

5.1 Subpackages

5.1.1 dotfiles.cli package

Submodules

dotfiles.cli.cli module

dotfiles commandline interface (CLI)

dotfiles.cli.cli.**get_pkg_resource_filename**(*path=''*)
This package generates a MANIFEST.in file from version control (hg) in order to include files outside of the package source directory (src/dotfiles).

When installed as an editable or a source dist, these additional data files have paths relative to `pkg_resource.resource_filename`.

Parameters **path** (*str*) – path fragment (eg. `etc/vim`)

Returns **path** – absolute path to `path`, relative to this package install

Return type str

class dotfiles.cli.cli.**Test_dotfiles**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

test_get_pkg_resource_filename()

test_dotfiles_main()

dotfiles.cli.cli.**main**(**args*)
Main method for the `dotfiles` CLI script.

Parameters **args** (*list*) – list of arguments (if empty, read `sys.argv[1:]`)

5.1.2 dotfiles.venv package

Objectives

`dotfiles.venv.ipython_config` (ipython_config.py):

- Set variables for standard *Virtualenv* paths in the environment dict

- Define *IPython* aliases
- Serialize variables and aliases to:
  - *IPython* configuration (variables, aliases)
  - *Bash*/*ZSH* configuration (variables, aliases, functions):
    * `venv -E --bash`
    * `venv dotfiles --bash`
  - *JSON* (variables, aliases) [`--print`]
    * `venv -E --json`
    * `venv dotfiles json`

`dotfiles.venv.ipython_magics` (ipython_magics.py):

- Configure *IPython* `%magic` commands
  - `cd*` – change directories (`%cdhelp`, `%cdp`, `%cdwh`, `%cdv`, `%cds`, `%cdw`)
  - `ds` – print dotfiles_status

Configuration

Shell

For Bash/ZSH, `etc/bash/10-bashrc.venv.sh` sets:

```
# ...
_VENV="${__DOTFILES}/etc/ipython/ipython_config.py"
venv() {
    $_VENV $@
}
# ...
```

`etc/bash/10-bashrc.venv.sh` is sourced by `etc/bash/00-bashrc.before.sh`, which is sourced by `~/.bashrc` (a symlink to `${__DOTFILES}/etc/bashrc` created by *bootstrap_dotfiles.sh -S*).

IPython

To configure IPython with venv, ipython_config.py must be symlinked into `~/.ipython/profile_default` (and, optionally, ipython_magics.py into `~/.ipython/profile_default/startup/`):

```
# symlink paths relative to ${__DOTFILES}
__DOTFILES="~/.dotfiles"

# working directory (path to the dotfiles repository)
_WRD=${WORKON_HOME}/dotfiles/src/dotfiles

# created by ``bootstrap_dotfiles.sh -S``
# ln -s ${_WRD} ${__DOTFILES}
# ln -s ${__DOTFILES}/etc/bashrc ~/.bashrc
# ln -s ${_WRD}/etc/ipython/ipython_config.py \
#       ${__DOTFILES}/etc/ipython/ipython_config.py

# MANUALLY INSTALL for each IPython profile
```

```
IPY_PROFILE="profile_default"
ln -s ${__DOTFILES}/etc/ipython/ipython_config.py \
      ~/.ipython/${IPY_PROFILE}/ipython_config.py

ln -s ${__DOTFILES}/etc/ipython/ipython_magics.py \
      ~/.ipython/${IPY_PROFILE}/ipython_magics.py
```

Submodules

dotfiles.venv.ipython_config module

class dotfiles.venv.ipython_config.**Env**(**args*, ***kwds*)
Bases: collections.OrderedDict

OrderedDict of variables for/from os.environ.

classmethod from_environ(*environ*)
Build an Env from a dict (e.g. os.environ)

Parameters **environ** (*dict*) – a dict with variable name keys and values

Returns an Env environment built from the given environ dict

Return type Env

osenviron_keys = ('__DOCSWWW', '__DOTFILES', 'PAGER', 'PROJECTS', 'USRLOG', 'VIMBIN', 'GVIMBIN',

paths_to_variables(*path_*)
Replace components of a path string with variables configured in this Env.

Parameters **path** (*str*) – a path string

Returns path string containing ${VARNAME} variables

Return type str

set_standard_paths(*env*, *prefix*)
Set variables for standard paths in the environment

Parameters **prefix** (*str*) – a path prefix (e.g. $VIRTUAL_ENV or $PREFIX)

see:

- https://en.wikipedia.org/wiki/Unix_directory_structure
- https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

class dotfiles.venv.ipython_config.**IPYMock**
Bases: object

Provide a few mocked methods for testing

magic(**args*, ***kwargs*)

system(**args*, ***kwargs*)

class dotfiles.venv.ipython_config.**Test_Env**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_Env()

test_Env_from_environ()

class dotfiles.venv.ipython_config.**Test_venv**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_venv()

test_venv_appname()

test_venv_from_null_environ()

test_venv_with_environ()

test_venv_without_environ()

class dotfiles.venv.ipython_config.**Venv**(*virtualenv=None*, *appname=None*, *env=None*, *from_environ=False*, *open_editors=False*, *open_terminals=False*, *dont_reflect=True*)
Bases: object

A virtual environment configuration generator

asdict()

Returns OrderedDict(env=self.env, aliases=self.aliases)

Return type OrderedDict

bash_env(*output=<open file '<stdout>', mode 'w' at 0x7fe58dcea150>*)
Generate a source-able script for the environment variables, aliases, and functions defined by the current Venv.

Parameters **output** (*file-like*) – object to print to (print calls .write() and then .flush())

configure_ipython(**args*, ***kwargs*)
Configure IPython with Venv._configure_ipython and user_aliases from self.aliases.items().

Parameters

- **args** (*list*) – args for Venv._configure_ipython
- **kwargs** (*dict*) – kwargs for Venv._configure_ipython.

configure_sys()

Returns sys.path list from _configure_sys.

Return type list

get_user_aliases(*dont_reflect=False*)
Configure env variables and return an OrderedDict of aliases

Parameters **dont_reflect** (*bool*) – Whether to always create aliases and functions referencing $_WRD even if $_WRD doesn't exist. (default: False)

Returns dict of aliases

Return type OrderedDict

get_user_aliases_base()

Returns dict of cd command aliases

Return type OrderedDict

Note: These do not work in IPython as they run in a subshell. See: dotfiles.venv.ipython_magics.

static get_virtualenv_path(*virtualenv=None*, *from_environ=False*)
Get the path to a virtualenv

Parameters

- **virtualenv** (*str*) – a path to a virtualenv containing / OR just the name of a virtualenv in $WORKON_HOME
- **from_environ** (*bool*) – whether to try and read from os.environ["VIRTUAL_ENV"]

Returns a path to a virtualenv (for $VIRTUAL_ENV)

Return type str

open_editors()
Run self._edit_project_cmd

open_terminals()
Run self._open_terminals_cmd

system(*cmd=None*)
Call os.system with the given command string

Parameters **cmd** (*string*) – command string to call os.system with

Raises

- Exception – if cmd is None
- NotImplementedError – if cmd is a tuple

to_json(*indent=None*)

Parameters **indent** (*int*) – number of spaces with which to indent JSON output

Returns json.dumps(self.asdict())

Return type str

classmethod workon_project(*virtualenv*, ***kwargs*)

Parameters

- **virtualenv** (*str*) – a path to a virtualenv containing / OR just the name of a virtualenv in $WORKON_HOME
- **kwargs** (*dict*) – kwargs to pass to Venv (see Venv.__init__)

Returns an intialized Venv

Return type Venv

dotfiles.venv.ipython_config.**get_DEFAULT_ALIASES**(*platform=None*)

dotfiles.venv.ipython_config.**get_venv_parser**()

Returns optparse.OptionParser: options for the commandline interface

dotfiles.venv.ipython_config.**get_workon_home_default**()

Returns Best path for a virtualenvwrapper $WORKON_HOME

Return type str

dotfiles.venv.ipython_config.**in_ipython_config**()

Returns True if get_ipython is in globals()

Return type bool

dotfiles.venv.ipython_config.**ipython_imports**()
Default imports for IPython (currently unused)

dotfiles.venv.ipython_config.**ipython_main**()
Function to call if running within IPython, as determined by in_ipython_config.

dotfiles.venv.ipython_config.**main**()
Commandline main function called if __name__=="__main__"

Returns nonzero on error

Return type int

dotfiles.venv.ipython_config.**shell_quote**(*var*)
Escape single quotes and add double quotes around a given variable.

Parameters **_str** (*str*) – string to add quotes to

Returns string wrapped in quotes

Return type str

Warning: This is not safe for untrusted input and only valid in this context (os.environ).

dotfiles.venv.ipython_config.**shell_varquote**(*str_*)
Add doublequotes and shell variable brackets to a string

Parameters **str** (*str*) – string to varquote (e.g. VIRTUAL_ENV)

Returns "${VIRTUAL_ENV}"

Return type str

dotfiles.venv.ipython_magics module

IPython %magic commands

- cd aliases
- ds (dotfiles_status)
- dr (dotfiles_reload)

Installation

```
__DOTFILES="~/.dotfiles"
ipython_profile="profile_default"
ln -s ${__DOTFILES}/etc/ipython/ipython_magics.py \
    ~/.ipython/${ipython_profile}/startup/ipython_magics.py
```

dotfiles.venv.ipython_magics.**magics_class**(*cls*, **args*, ***kwargs*)

dotfiles.venv.ipython_magics.**line_magic**(*func*, **args*, ***kwargs*)

class dotfiles.venv.ipython_magics.**VenvMagics**
Bases: object

cd (*envvar*, *line*)
Change directory

Parameters

- **envvar** (*str*) – os.environ variable name
- **line** (*str*) – path to append to envvar

cdb (*line*)
cdb() – cd ${_BIN}/${@}

cde (*line*)
cde() – cd ${_ETC}/${@}

cdh (*line*)
cdh() – cd ${HOME}/${@}

cdl (*line*)
cdl() – cd ${_LIB}/${@}

cdlog (*line*)
cdlog() – cd ${_LOG}/${@}

cdp (*line*)
cdp() – cd ${PROJECT_HOME}/${@}

cdph (*line*)
cdph() – cd ${PROJECT_HOME}/${@}

cdpylib (*line*)
cdpylib() – cd ${PYLIB)/${@}

cdpysite (*line*)
cdpysite() – cd ${PYSITE}/${@}

cdsitepackages (*line*)
cdsitepackages() – cd ${_PYSITE}/${@}

cds (*line*)
cds() – cd ${_SRC}/${@}

cdv (*line*)
cdv() – cd ${VIRTUAL_ENV}/${@}

cdve (*line*)
cdve() – cd ${VIRTUAL_ENV}/${@}

cdvirtualenv (*line*)
cdvirtualenv() – cd ${VIRTUAL_ENV}/${@}

cdvar (*line*)
cdvar() – cd ${_VAR}/${@}

cdw (*line*)
cdw() – cd ${_WRD}/${@}

cdwrd (*line*)
cdwrd() – cd ${_WRD}/${@}

cdwrk (*line*)
cdwrk() – cd ${WORKON_HOME}/${@}

cdwh (*line*)
cdwh() – cd ${WORKON_HOME}/${@}

cdww(*line*)
cdww() – cd ${_WWW}/${@}

cdwww(*line*)
cdwww() – cd ${_WWW}/${@}

cdhelp(*line*)
cdhelp() – list cd commands

dotfiles_status(*line*)
dotfiles_status() – print dotfiles_status() .

ds(*line*)
ds() – print dotfiles_status() .

dotfiles_reload(*line*)
dotfiles_reload() – print NotImplemented

dr(*line*)
dr() – print NotImplemented [dotfiles_reload()]

dotfiles.venv.ipython_magics.**main**()
Register VenvMagics with IPython

CHAPTER 6

Developing

6.1 Create a virtualenv with virtualenvwrapper

Install **virtualenvwrapper** (with pip):

```
pi install --upgrade --user pip virtualenv
pip install --upgrade --user virtualenvwrapper
source $(HOME)/.local/bin/virtualenvwrapper.sh
```

Make a **virtualenv** for the **dotfiles** source with **virtualenvwrapper**:

```
mkvirtualenv dotfiles
workon dotfiles
cdvirtualenv
ls -ld **/**

mkdir -p ${VIRTUAL_ENV}/src
cdvirtualenv src
```

6.2 Install this package

- Install into `$VIRTUAL_ENV` (with pip):

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

6.3 Test and build this package

- Install into `$VIRTUAL_ENV` (manually):

```
cd ${VIRTUAL_ENV}/src
git clone https://github.com/westurner/dotfiles
hg clone https://bitbucket.org/westurner/dotfiles

cd dotfiles
ls -l ./dotfiles/**
hg paths || git remote -v && git branch -v

ls -l ./dotfiles/etc/vim/**
```

```
cd ./dotfiles/etc/vim
hg paths || git remote -v && git branch -v

cd ${VIRTUAL_ENV}/src/dotfiles
# cd $_WRD
# cdw
# pip install -e .
python setup.py develop
```

- Build

```
# sudo apt-get install make git mercurial

cd ${VIRTUAL_ENV}/src/dotfiles
echo $EDITOR
make build_tags
make edit
make test
make build
make install

# pip install -r ./requirements-all.txt
make pip_install_requirements_all
```

CHAPTER 7

Changelog

7.1 0.01

- Created dotfiles project

CHAPTER 8

License

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*

d

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

V

W

X

Y

Z